World Wide Web (2022) 25:309–333
https://doi.org/10.1007/s11280-021-00929-9

Joint service-function deployment and task scheduling in UAVFog-assisted data-driven disaster response architecture

Xianglin Wei[1] · Li Li[2] · Lingfeng Cai[1] · Chaogang Tang[3] · Suresh Subramaniam[2]

Received: 7 December 2020 / Revised: 24 May 2021 / Accepted: 12 July 2021 /
Published online: 20 August 2021

Abstract
It is critical but challenging to provide efficient information services to support disaster-response operations in disaster-hit areas. A UAVFog-assisted data-driven disaster-response architecture, which combines unmanned aerial vehicles (UAVs) and fog computing paradigm, showed many advantages in response latency and on-the-fly deployment. This paper aims to jointly optimize the deployment of service functions (SFs) and the task scheduling at UAVFog nodes to minimize the task response latency. After introducing the collaboration structure between UAVFog nodes, joint SF deployment and task scheduling is formulated as an optimization problem. Then, three algorithms are put forward to tackle the problem: 1) Dependency and topology-aware SF deployment (DeToSFD) algorithm is developed to determine the initial deployment location of each SF; 2) Context-aware greedy task scheduling (CoGTS) algorithm is put forward to schedule an arrived task; 3) Congestion-aware SF reallocation (CoSFR) algorithm is developed to reallocate SFs in case of congestion at an instance of an SF. Finally, a series of experiments are conducted to evaluate the performance of the proposed algorithms. Experimental results show that DeToSFD, CoGTS, and CoSFR could greatly reduce the task response latency of the UAVFog system in diverse parameter settings.

Keywords Unmanned aerial vehicle · Fog computing · Service functions · Scheduling

1 Introduction

Many devastating disasters can cause massive damage to human-made infrastructures and lead to injuries and the loss of lives. Organizing efficient disaster-response operations in the

This article belongs to the Topical Collection: *Special Issue on Web Information Systems Engineering 2020*
Guest Editors: Hua Wang, Zhisheng Huang, and Wouter Beek

✉ Chaogang Tang
cgtang@cumt.edu.cn

Extended author information available on the last page of the article.

disaster-hit area is a challenging task due to damaged roads, corrupted buildings, and power outage, etc. The damaged infrastructures prevent first responders and rescue teams from timely sensing, sharing, and processing the collected data. Many traditional operators of disaster rescue resort to TETRA (Terrestrial Trunked Radio), walkie-talkie, satellite, or mobile ad hoc networks for building emergency transmission service. However, these solutions face great challenges in availability, bandwidth, and latency. Moreover, several efforts have been made to adopt Unmanned Aerial Vehicles (UAVs) to provide aerial communications relay service from the sky [6]. But existing efforts usually focus on providing transmission services without paying attention to the computational demands of disaster relief operations.

To support disaster-response actions, it is necessary to incorporate machine learning components into the applications, such as injury diagnosis, route planning, video or audio recognition, etc. However, it is usually infeasible or very time-consuming to execute these computation-intensive tasks on mobile devices. Therefore, under fog/edge computing paradigm [1, 3, 21], an entity called UAVFog node is presented to provide both computation and transmission services [20]. In a UAVFog-assisted data driven architecture, multiple UAVFog nodes are deployed in the disaster-hit area. They can collaborate with each other through organizing a flying ad hoc network (FANET) to provide computation-communication integrated services. However, the abridged version of this work, i.e. [20], does not cover the service function (SF) deployment and dependent task scheduling in the system.

In this paper, we first refine the design of the UAVFog-based disaster-response architecture. The collaboration between UAVFog nodes are highlighted and detailed. Second, an SF deployment and task scheduling model is formulated. SF deployment concentrates on the service deployment in the UAVFog-assisted service architecture. On the other hand, task scheduling aims at minimizing the task response latency. Through jointly optimizing these two goals, we could benefit both the service provider and consumers. Third, three algorithms for deploying/reallocating the SFs on UAVFog nodes, and scheduling arrived tasks, are designed. Finally, the experimental results of the presented architecture and algorithms are provided to demonstrate the effectiveness of the proposals. The contributions of this paper are threefold:

1) The collaboration architecture between UAVFog nodes is introduced. Moreover, the design of a UAVFog node is detailed.
2) The joint SF deployment and task scheduling problem is formulated as an optimization problem, which considers both the response time of dependent tasks and the deployment cost of SFs. In contrast, existing efforts usually optimize these two goals separately.
3) Three algorithms for determining the initial deployment of SFs, the scheduling of arrived tasks, and the reallocation of the SFs, are designed. Moreover, a series of experiments are conducted to evaluate the performance of the proposed algorithms. Experimental results show that our proposal could greatly reduce the task response latency with different dependent task and arrival settings.

The remainder of this paper is organized as follows. Section 2 summarizes related work. In Section 3, the UAVFog-based collaborative disaster-response architecture is introduced. Section 4 formulates the service provision model; and the designed algorithms are presented in Section 5. The experimental results are illustrated in Section 6. Section 7 concludes this paper.

2 Related work

UAVs have been adopted to provide communications or computation service in diverse scenarios, where one or multiple UAV(s) are deployed to establish communication links between mobile devices.

Mozaffari et al. presented an algorithm to optimize the placement of UAVs and the transmission power of mobile device to reduce the energy consumption [17]. Al-Turjman have reviewed the applications of UAV in wireless communications [2]. To ensure the fairness of the mobile devices, Wu et al. have optimized the scheduling of the UAVs [23]. Zeng et al. have optimized the energy consumption of the UAVs with the communications constraints of the mobile devices [25].Through optimizing the trajectory of the UAV, Liu et al. have promoted the communications capacity of the network [15]. Erdelj et al. discussed the role of UAVs in different types of disasters [7]. In [13], Król et al. presented a UAV-assisted architecture that solves the communication outage issues caused by floods. Arbia et al. discussed an Internet-of-Things (IoT)-based end-to-end emergency and disaster relief system [4]. Lu et al. proposed a data transmission network by connecting smart phones [16]. However, these proposals does not consider the computation needs of mobile devices.

Through combining edge/fog computing and UAVs, a few UAV edge computing architectures have been presented. Wei et al. jointly optimized the response latency and energy consumption of the UAV edge computing system [22]. Hu et al. considered the UAV positions, time slot allocation, and task partition to minimize the energy consumption [10]. Hua et al. optimized resource allocation, task scheduling, and the trajectory of the UAV to minimize the energy consumption of the mobile devices [11]. Hu et al. aimed to minimize the sum of the maximum delay among all the users in each time slot by jointly optimizing the UAV trajectory, the ratio of offloading tasks, and the user scheduling variables [9]. Tang et al. have presented an optimal partial offloading scheme to obtain the optimal offloading ratio, local computing frequency, transmission power and edge server computing frequency [19]. However, one UAV could only serve a very small region of the disaster-hit area.

When there are many UAVs that provide services, the problem becomes much more difficulty. Cheng et al. presented the architecture of the air-ground integrated mobile edge networks [5]. Yang et al. put forward an iterative algorithm to minimize the total power via jointly optimizing user association, power control, computation capacity allocation, and location planning [24]. However, these proposals only focus on architecture design of UAV-assisted networks. Little attention has been paid to the service deployment and dependent task scheduling.

To our best knowledge, most existing UAV Edge/fog computing proposals neglected the SF deployment and dependent task scheduling problem. On the one hand, SF deployment is the basis of support disaster-response services. On the other hand, dependent task scheduling is critical for efficient service utilization in resource-shortage scenarios. In this backdrop, this paper focuses on the design of the service architecture and the algorithms for deploying SFs on UAVs and scheduling dependent tasks. A preliminary version of this paper was published and presented at the 21st International Conference on Web Information Systems Engineering (WISE2020) [20]. Compared with the preliminary version, this paper greatly extended the contents by three aspects: 1) The collaboration architecture between UAVFog nodes are detailed; 2) The SF deployment and task scheduling problem is formulated as an optimization problem; 3) Three algorithms are developed with experimental verification.

3 UAVFog-based collaborative disaster-response architecture

3.1 Computing-communication challenges in disaster scenarios

Natural disasters could severely damage the man-built infrastructures in disaster-hit area. In this paper, we assume the following challenges when conducting disaster-response operations in the disaster-hit area: 1) the access devices and the core transmission devices, including radio towers, Internet routers, and data centers could not work properly due to collapsed buildings and power outage; 2) several geographical areas are inaccessible for the first responders due to landslide, floods, damaged bridges, etc.; 3) many injured victims spread over the disaster-hit area and need medical treatment; 4) a number of portable or mobile devices, such as sensors, smart phones, or laptops, are carried by victims or rescuers, and they have disaster-response tasks to execute.

In the past, a rescuer has to rely on the emergency network for communications with very limited knowledge-support. Nowadays, data-driven analytic holds tremendous potential for assisting many aspects of disaster response, such as injury pinpointing, damage assessing, resources allocation, etc. To implement data-driven disaster-responses, raw data (including text, photos, videos, medical data, weather information, etc.) captured by the sensors are transmitted to the processing units for knowledge extraction. Then, on the processing units, the collected data are handled by machine learning algorithms which assist the decision-making. Finally, the extracted knowledge or execution results are returned to the information requesters, such as injuries, rescuers, etc., to assist their operations.

However, realizing aforementioned steps in the disaster-hit areas is formidable due to the shortage of transmission, storage, and computation resources. To tackle the transmission problem, many efforts have been made for deploying UAVs or satellite-terminals. However, the computation need, which is critical for executing the computational intensive machine learning algorithms, is usually neglected in the existing research. By adopting multiple UAVs, called UAVFog nodes as computation, transmission, and storage platforms in the disaster-hit areas [20], a quintessential UAVFog-assisted data-driven disaster response scenario is shown in Figure 1.

In this scenario, a number of UAVFog nodes are deployed to provide integrated services in the disaster-hit areas. Each UAVFog node carries sensing, computation, and transmission payload, and can serve a limited area. To maintain the connectivity of the area, UAV-to-UAV links are established between UAVFog nodes besides the UAV-to-mobile device links. Mobile devices on the ground could offload their data or computation tasks to its associated UAVFog node via air-to-ground links[8, 14]. Typically, a computation-intensive task is divided into several sub-tasks, which could be executed in multiple UAVFog nodes in a collaborative way. We assume that each sub-task could be mapped to an SF on a UAVFog node. In the following analysis, we will use sub-task and SF interchangeably without raising any confusion.

3.2 Architecture specification

From the viewpoints of function and data, Figure 2 illustrates the system architecture of UAVFog-assisted data-driven disaster-response [20].

The four-layer architecture illustrates the entities involved in each function domain. In contrast, the right-hand side of Figure 2 shows how the raw data is processed. In the data domain, raw data is generated or collected by sensors on the bottom layer, i.e. the data generation layer. These raw data, including videos, audios, images, texts, healthy records, etc.,

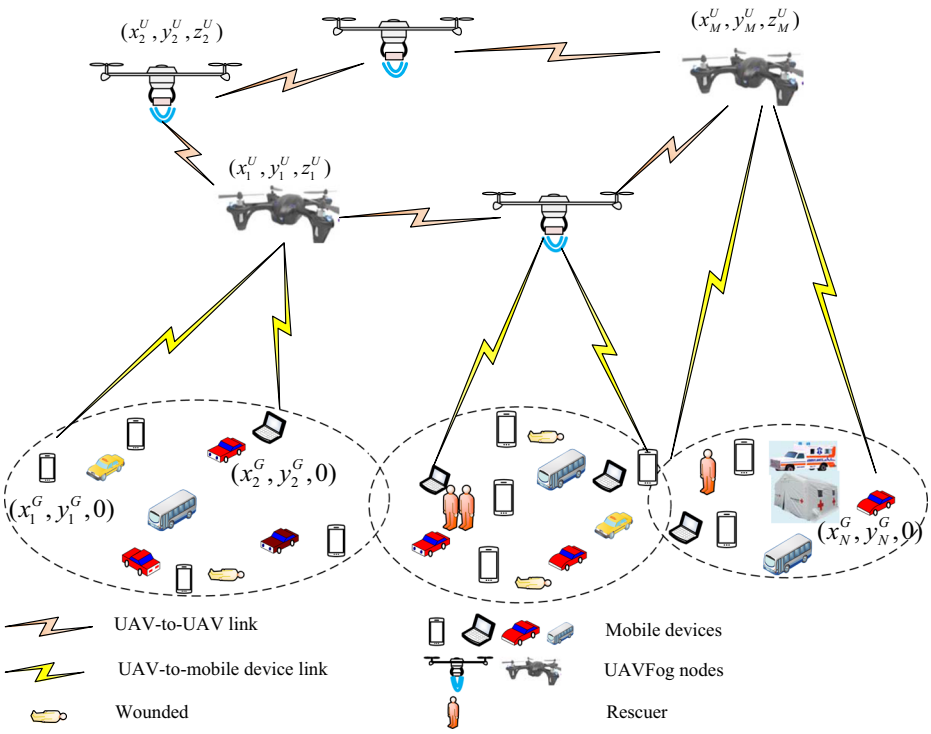


Figure 1 A typical UAVFog-assisted data-driven disaster response scenario

are usually heterogeneous with different formats and volumes. They will be transmitted by transmission devices, including UAV-carried modules, and satellites in the transmission layer. To regulate the data transmission, multiple modules, such as protocol design, bandwidth assignment, etc., are implemented in the data transmission layer. A UAVFog node, as a data processing unit, implements different types of functions or modules in the data processing layer. In the disaster response scenario, typical SFs include image recognition, text classification, etc. Diverse machine learning algorithms are implemented in the computation layer. The data output layer or the application layer interacts with the mobile devices in a machine-type-communication fashion. A detailed description of this architecture could be referred to [20]. A rescuer could use these applications in a transparent way without knowing information provider, raw data generator, and the data processing procedure.

3.3 UAVFog node structure

UAVFog nodes can accomplish multiple functionalities and provide computation services to the mobile devices in vicinity. The data collected by mobile devices can be offloaded to UAVs for processing and analysis. In this way, we can put in-place emergency response measures in time. Compared with the cloud computing paradigm, the response time can be drastically reduced.

Moreover, UAVFog nodes can work together to perform a complicated task. Usually, it can be done by deploying SFs on different UAVFog nodes. Note that, a service that can accomplish basic task is called an SF or a basic/atomic service. Correspondingly, the service

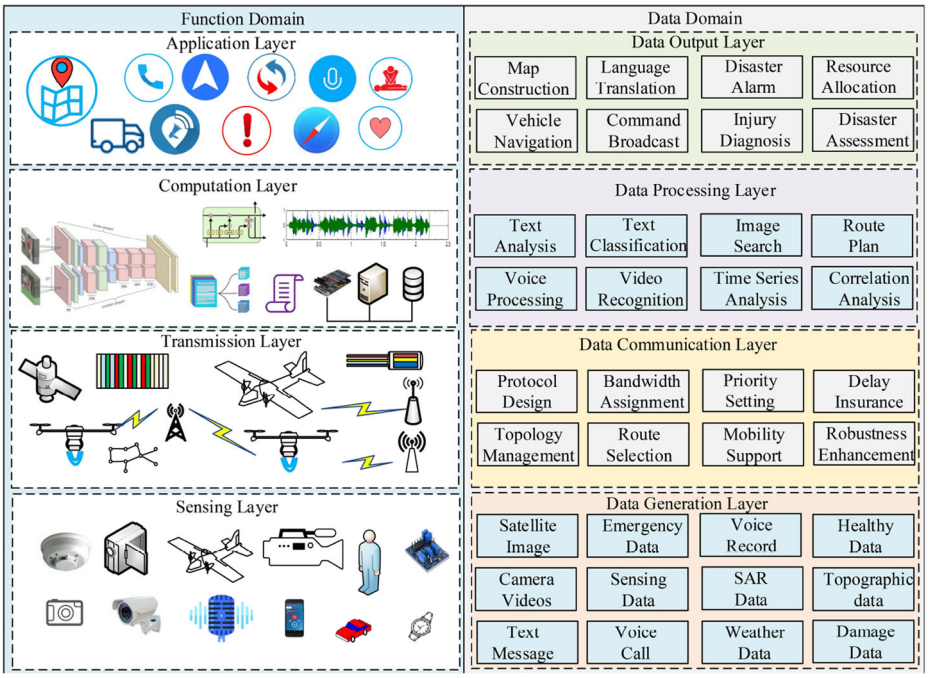


Figure 2 The system architecture from both function and data viewpoints

that can complete a task that consists of multiple SFs is called a composite service. In many cases, it is infeasible to deploy all the SFs on one single UAVFog node due to resource limitation. Therefore, we assume that it is reasonable to limit the number of SFs loaded at each UAVFog node within its resource budget. SFs that are equipped on multiple UAVFog nodes can compose a composite service.

The functional structure of a UAVFog node is depicted in Figure 3. The functionalities are divided into four units, i.e., the resources management unit (RMU), service management unit (SMU), user access unit (UAU), and collaborative management unit (CoMU). RMU is responsible for management of physical and virtual resources. Computing resources are the most important section, which offers support to other units. In the SMU, a core function module called decision making unit (DMU) is designed to perform service deployment, task offloading, task management and partition. Tasks offloaded from the mobile devices are processed first by the DMU. To be specific, DMU decides the partition and assignment of tasks. If the task can be completed in a single UAVFog node, DMU can make decision quickly by forwarding the task to the UAVFog node where the corresponding SF is deployed. If a task requires more than one SF, a composite service is needed. In such a case, an offloading decision should be made carefully by considering latency optimization and energy consumption. Meanwhile, it is also crucial for the UAVFog node itself to select suitable SFs to deploy such that the UAVFog system can be optimized based on given metrics. To explain the service deployment and task assignment, a use case will be discussed in details later.

CoMU plays an important role in service management. It receives tasks from other UAVFog nodes, and it may further forward tasks to other UAVFog nodes. It frequently communicates

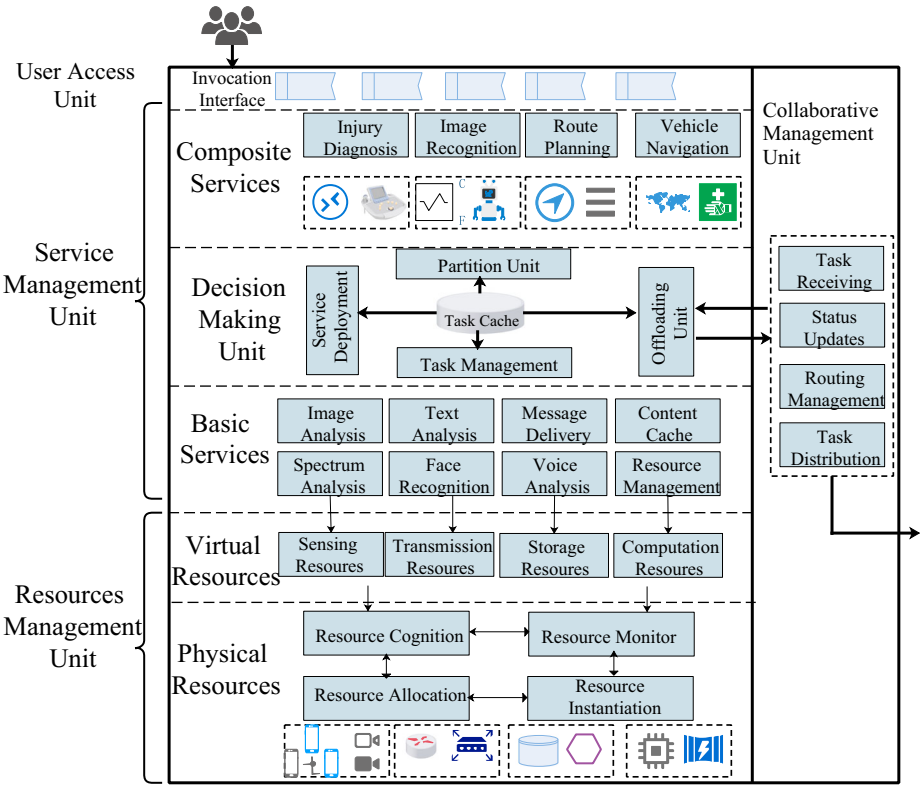


Figure 3 The structure of a UAVFog node

with DMU to be aware of the decision of task assignments. However, determining a subtask's execution node is non-trivial, and will be an essential part of our following analysis. Furthermore, CoMU is also responsible for exchanging status information with other UAVFog nodes. Thus, CoMU keeps the global SF deployment and task scheduling information on the entire FANET. UAU maintains an access interface for the service requests.

3.4 Service deployment and UAVFog collaboration

As introduced earlier, we intend to determine the service deployment on UAVFog nodes and the assignments of the tasks, such that results can be returned with the least response latency. We assume UAVFog nodes have different computing capabilities as well as the energy supplies. Computation-intensive tasks should be offloaded to the UAVFog nodes with sufficient computing resources. Due to limited resource at one single UAVFog node, a composite service may be dispatched on multiple nodes.

An example of service deployment at UAVFog nodes is shown in Figure 4, where two tasks (G_1 and G_2) are offloaded to five UAVFog nodes. We assume that the location of UAVFog nodes are stationary. G_1 and G_2 are offloaded to UAVFog nodes U_1 and U_4, respectively. The DMU analyzes a task when it arrives at a UAVFog node. The analysis includes the topology of the task, the number of sub-tasks it comprises, the demanded computational resources. At the same time, the DMU periodically interacts with CoMU to obtain

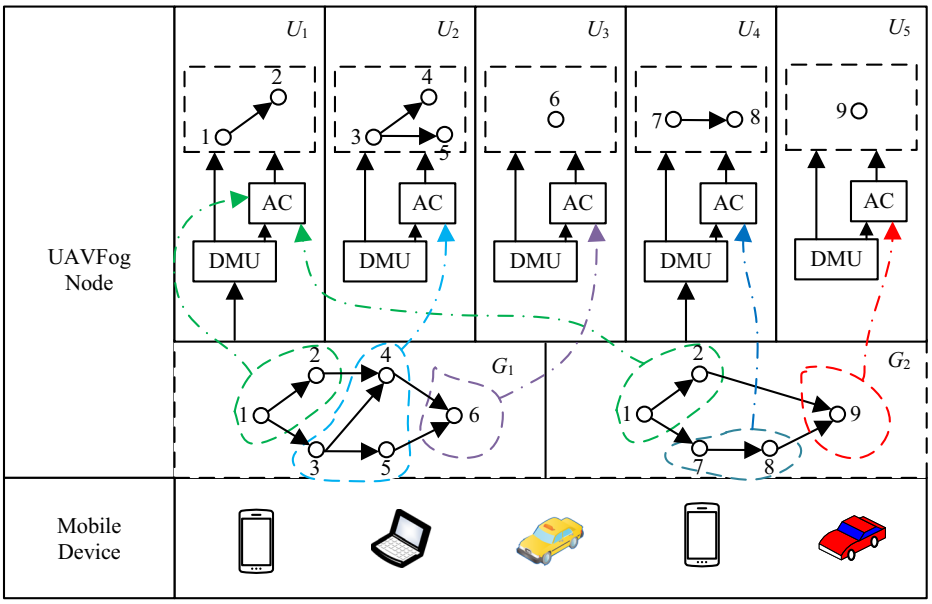


Figure 4 An example of service deployment at UAVFog nodes

the status of other UAVFog nodes. Based on these information, multiple UAVFog nodes collaboratively strive to achieve a globally optimal service deployment and task scheduling. Take G_1 and G_2 as an example, based on the service deployment decision, the arrangement can be sketched as follows. Two SFs (f_1 and f_2) are deployed on U_1 for sub-tasks 1 and 2, and three SFs are deployed on U_2 to accomplish sub-tasks 3, 4 and 5. Other service deployments follow the same pattern in Figure 4. As a result, the DMU of U_1 needs to distribute sub-tasks, i.e sub-task 3, 4, 5, and 6, to other UAVFog nodes, while sub-tasks 1 and 2 are retained in the task queue (managed by the Application Caching (AC) unit) for execution. We can observe that the key of this cooperation among UAVFog nodes is determining the service deployment and task scheduling policy that could jointly achieve the best latency performance.

3.5 Use case study

Figure 5 illustrates a use case, in which four UAVFog nodes work together to provide services. Each UAVFog node has limited serving area and SFs. For instance, U_1 leverages its sensing capabilities to capture the photos of the disaster-hit area, and searches for the injured people in step ①. Image processing methods, such as facial recognition, are performed on U_1 in step ②. U_1 distributes the data to U_2 which further helps rescue planning (step ③). Over its serving area, U_2 also incorporates raw data retrieved by its sensors into the task to improve the information accuracy. Rescuers take photos of the injured person, and offload the photos to U_2, where they can be analyzed in step ⑦. Combining the information and task from U_1, a diagnosis plan can be obtained. For example, if the wounded is in an emergency, a doctor may be sent for further treatments. To save the energy and balance the computing resources of UAVFog nodes, U_2 does not have the SFs which coordinates with the emergence center and plans rescue route. As a result, the tasks of route planning and

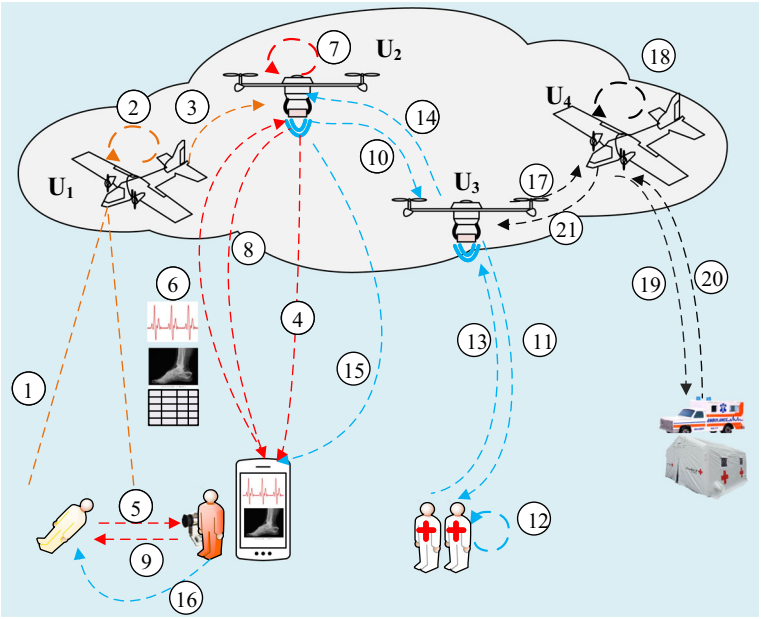


Figure 5 Use case study on the cooperation among UAVFog nodes

coordination with the emergence center need to be accomplished by other UAVFog nodes such as U_3 and U_4 in step (10) and step (17), respectively. For instance, U_3 is responsible for assisting diagnosis and communicating with doctors in step (11)-(13), and U_4 is responsible for the navigation of the ambulance in step (19)-(20). In this cooperation among UAVFog nodes, each UAVFog node only focuses on their own tasks, and could distribute a few tasks to other UAVFog nodes.

In this use case, the services on the UAVFog nodes were assumed to be deployed in advance. However, A fixed deployment of SFs may lead to low deployment efficiency due to two facts: 1) the unpredictable nature of the arrived tasks; 2) the dynamic nature of UAVFog nodes. Moreover, dependent task scheduling also plays a key role in promoting the disaster response efficiency. Therefore, the following sections concentrate on the joint optimization problem of SF deployment and task scheduling.

4 Service provision model and algorithm design

To explore the full potential of UAVFog-assisted disaster response architecture, we face a dilemma between the harsh task offloading requirements from mobile devices and limited energy supply and computation capacity of UAVFog nodes. The problem of limited energy supply at UAVFog nodes could be alleviated by adopting tethered drones that physically connected to a power station on the ground. However, the energy supply at the ground power station is also limited; in other words, saving energy at a UAVFog node is important although it is not treated as the most important objective in this paper. To tackle the computation shortage challenge, multiple UAVs need to collaboratively process computation-intensive or time-sensitive tasks. In this collaborative architecture, a limited number of SFs are deployed in each node; and an offloaded task may be processed by one

or multiple UAVFog nodes equipped with the desired services. However, predetermining the SFs on a UAVFog node is not a trivial task. Moreover, scheduling each arrived task is also challenging. This section formulates the joint optimization problem.

4.1 Service provision model

4.1.1 Disaster-hit area model

Suppose N mobile devices spread in an $L \times L$ area; M UAVFog nodes are available for serving or supporting the disaster response. The set of UAVFog nodes is denoted as $\mathcal{U} = \{u_1, u_2, \ldots, u_M\}$. A basic requirement is that all UAVFog nodes could cover this area. Moreover, all UAVFog nodes are deployed evenly, and each of them serves an area of roughly the same size, for example, a grid as shown in Figure 6. Note that in Figure 6, wireless links are established between adjacent UAVFog nodes to build a FANET. With the help of FANET, any two UAVFog nodes could communicate with each other directly or through their neighbors.

In the scheduling period, the locations of mobile devices do not change, and the location of the i-th device is denoted as $(x_i^G, y_i^G, 0)$, $1 \le i \le N$. Assume the deployment position of the j-th UAVFog node is (x_j^U, y_j^U, H), where H is the height of the UAVFog node and is assumed to be a constant, $1 \le j \le M$. Then, the distance between mobile device i and UAVFog node j is:

$$d_{ij}^{GU} = \sqrt{H^2 + (x_j^U - x_i^G)^2 + (y_j^U - y_i^G)^2} \tag{1}$$

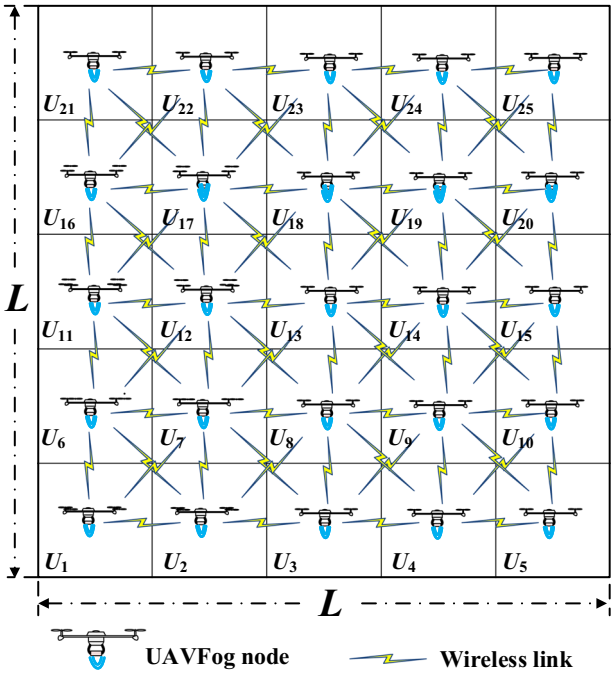


Figure 6 A typical deployment scenario of multiple UAVFog nodes

4.1.2 Transmission model

It is relatively easier for a UAVFog node to establish a Line of Sight (LoS) wireless link with the mobile devices on the ground. The probability that there exists a LoS link between the i-th mobile device and the j-th UAVFog node is [22]:

$$P_{ij}^{LoS} = \frac{1}{1 + \psi \cdot \exp(-\beta(\theta_{ij} - \psi))} \tag{2}$$

where ψ and β depend on the communication frequency and the propagation environment, θ_{ij} refers to the elevation angle:

$$\theta_{ij} = \frac{180}{\pi} \times \sin^{-1}\left(\frac{h}{d_{ij}^{GU}}\right)$$

The transmission path between the i-th device and the j-th UAVFog node is:

$$L_{ij} = \begin{cases} \eta_1 \left(\frac{4\pi f d_{ij}^{GU}}{c}\right)^{\epsilon}, \text{LoS link} \\ \eta_2 \left(\frac{4\pi f d_{ij}^{GU}}{c}\right)^{\epsilon}, \text{Non-LoS link} \end{cases}$$

Here, f is the carrier frequency; ϵ is the path loss exponent; η_1 and η_2 are the excessive path loss coefficients in LoS and NLoS cases, and c is the speed of light. Therefore, the average path loss between device i and UAVFog node j is [17]:

$$\bar{L}_{ij} = \left(P_{LoS}^{ij}\eta_1 + \left(1 - P_{LoS}^{ij}\right)\eta_2\right)\left(\frac{4\pi f}{c} d_{ij}^{GU}\right)^{\epsilon}$$

The received signal-to-interference-plus-noise-ratio (SINR) at the i-th device from the j-th UAVFog node is:

$$\gamma_{ji}^{UG} = \frac{P_j^{UG}}{(I_{ij} + \sigma^2)\bar{L}_{ij}} \tag{3}$$

where P_j^{UG} is the transmission power of the j-th UAVFog node, I_{ij} represents the interference from other devices using the same electromagnetic spectrum, and σ^2 refers to the ambient environment noise. Note that mobile device i may receive signals from multiple UAVFog nodes. In this circumstance, the UAVFog node with the highest SINR at the i-th device will be chosen. In the following analysis, the chosen UAVFog node is called the i-th device's associated UAVFog node. The transmission rate from the j-th UAVFog node to device i is:

$$r_{ji}^{UG} = W_1 \log_2\left(1 + \gamma_{ji}^{UG}\right) \tag{4}$$

where W_1 is the bandwidth of the UAV-to-ground wireless channel. In this way, both the received SINR and transmission rate from device i to the j-th UAVFog node, denoted by γ_{ij}^{GU} and r_{ij}^{GU}, respectively, could be calculated by:

$$\gamma_{ij}^{GU} = \frac{P_i^G}{(I_{ij} + \sigma^2)\bar{L}_{ij}} \tag{5}$$

$$r_{ij}^{GU} = W_1 \log_2(1 + \gamma_{ij}^{GU}) \tag{6}$$

where P_i^G is the transmission power of the i-th device.

A link between two UAVFog nodes is modeled by the free-space path loss model since there is no signal obstruction or reflection in a clear air space. The distance between the j-th and the k-th UAVFog nodes is:

$$d_{jk}^{UU} = \sqrt{(x_j^U - x_k^U)^2 + (y_j^U - y_k^U)^2}$$

In the free-space path loss model, the channel power is expressed as $\beta_0 \left(d_{jk}^{UU}\right)^{-2}$, where β_0 is the channel power at the reference distance $d_0 = 1m$. The transmission rate from UAVFog node j to node k is:

$$r_{jk}^{UU} = W_2 \log_2 \left(1 + \frac{P_j^{UU} \beta_0}{(d_{jk}^{UU})^{-2} \sigma^2}\right) \tag{7}$$

where W_2 is the bandwidth of the UAV-to-UAV wireless channel, and $\frac{P_j^{UU} \beta_0}{(d_{jk}^{UU})^{-2} \sigma^2}$ is the SINR at UAVFog node k. A wireless link could be established from UAVFog node j to k when $\frac{P_j^{UU} \beta_0}{(d_{jk}^{UU})^{-2} \sigma^2} \geq \gamma_0$, where γ_0 is the SINR threshold.

4.1.3 Service function/sub-tasks model

A UAVFog node has limited computation and transmission resources. This constrains the SFs that could be deployed on it. For instance, the computation module that a DJI drone usually carries is the Manifold 2-G with 128GB storage space, which is less powerful than a desktop PC. Therefore, the functions deployed on a UAVFog node should be carefully designed. The set of all the SFs, i.e. sub-tasks, that could be deployed on UAVFog nodes is denoted as $\mathcal{F} = \{ f_1, f_2, \ldots, f_K \}$, and K is the number of SFs.

Each task could be described as a directed acyclic graph (DAG) $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of sub-tasks and $\mathcal{E}$ refers to the set of connections between sub-tasks. There is a directed edge from sub-task v_i to v_j if the execution of v_j depends on the computation results of v_i (here, $v_i, v_j \in \mathcal{V}$). In other words, an edge $(v_i, v_j) \in \mathcal{E}$ on the graph G means that sub-task v_j cannot start until sub-task v_i finishes. To model the data dependency between v_i and v_j, a weight w_{ij} is assigned on edge (v_i, v_j) to represent the amount of data transmitted from v_i to v_j.

Recall that a sub-task is mapped to an SF deployed on the UAVFog nodes. Moreover, the set of sub-tasks, i.e. $\mathcal{V}$, in task G belongs to the SF set $\mathcal{F}$, i.e. $v_i, v_j \in \mathcal{V} \subset \mathcal{F}$.

Two DAGs are shown in Figure 4. Task G_1 contains 6 sub-tasks, i.e. $\{v_1, v_2, \ldots, v_6\}$. Task G_2 consists of 5 sub-tasks, i.e. $\{v_1, v_2, v_7, v_8, v_9\}$. Tasks G_1 and G_2 are offloaded from devices to UAVFog nodes U_1 and U_4 respectively. Upon a task is partitioned into sub-tasks or functions by the DMU according to its DAG, and is distributed to one or multiple UAVFog nodes with the required SFs.

4.1.4 Service function deployment model

Due to the limited payload, computation, and storage capacity of a small-size UAV, it is impractical to deploy all the requested SFs on a single UAVFog node. Here, we focus on the computation resource on a UAVFog node and determine the set of SFs that is deployed

on it. Assume that the amount of computation resource on the j-th UAVFog node as C_j. Denote the set of SFs deployed on the j-th UAVFog node as $\mathcal{F}_j$. Then,

$$\mathcal{F}_j \subset \mathcal{F}, 1 \leq j \leq M \tag{8}$$

$$\sum_{f \in \mathcal{F}_j} C(f) \leq C_j \tag{9}$$

where $C(f)$ refers to the computation resource occupied by SF f. Inequality (9) means that the amount of resources required by all deployed SFs at a UAVFog node cannot exceed its capacity.

As the requested SFs or sub-tasks are unknown before deploying UAVFog nodes. To ensure that a requested SF could be provided by at least one UAVFog node. It yields

$$\mathcal{F}_1 \cup \mathcal{F}_2 \cup \ldots \cup \mathcal{F}_M = \mathcal{F} \tag{10}$$

Combining (9) and (10), we require that the total computation resource of all UAVFog nodes could carry out the SF set. Assume that each UAVFog node has enough storage capacity to store all the SF images since the hard drives are usually much cheaper than the computation units. However, only a few SFs could be instantiated at a time due to limited computation resources.

A UAVFog node needs time to instantiate or load a requested SF. For the j-th UAVFog node, the loading time of the SF f_k is assumed to be I_j^k. I_j^k depends on the resources allocated to the container for instantiating the SF. Generally speaking, the more the allocated resources, the shorter I_j^k will be.

4.1.5 Task execution model

The execution time of a task G refers to the time interval from G is offloaded to its completion time, which is determined by the time when its last sub-task finishes. For a specific sub-task $v \in \mathcal{V}$ in task G, the execution time is:

$$t(v) = \frac{b(v)c(v)}{f(A(v))} \tag{11}$$

where $b(v)$ is the number of input bits of task v, and each input bit requires $c(v)$ CPU cycles for processing. $A(v)$ refers to the UAVFog node that executes task v, and $f(A(v))$ is the CPU cycles assigned by $A(v)$ for processing task v. The execution time of task G, denoted as T_{exe}^G, is defined as:

$$t_{exe}^G = \sum_{v \in \mathcal{V}} t(v) \tag{12}$$

Moreover, the data transmission time between sub-tasks, denoted by t_{trans}^G is determined by the weights on all edges of $\mathcal{E}$ in task G. Suppose the waiting time of the sub-tasks on different UAVFog nodes, denoted as t_{wait}^G, cannot be neglected. In total, the response latency of task G is defined as:

$$t_{resp}^G = t_{exe}^G + t_{trans}^G + t_{wait}^G \tag{13}$$

4.2 Problem formulation

Timely task execution is desired in the UAVFog-based disaster-response architecture. Therefore, we minimize the average response latency of all the offloaded tasks during a

period $\mathcal{T}$. Denote the set of tasks arrived in the period $\mathcal{T}$ as $\mathcal{G}$, then the objective of the SF deployment and scheduling model is defined as:

$$\min_{A(v), f(U_j)} \frac{\sum_{G \subset \mathcal{G}} t_{resp}^{G}}{|\mathcal{G}|} \tag{14}$$

$$\text{subject to} \quad v \in \mathcal{V}, \tag{15}$$

$$1 \leq j \leq M, \tag{16}$$

$$A(v) \in \{U_1, \ldots, U_M\}, \tag{17}$$

$$(8), (9), (10) \tag{18}$$

Note that, the energy consumption is not included in our objective function due to the adoption of the tethered UAVFog node powered by a ground power station.

5 SF deployment and scheduling algorithm

5.1 Basic idea

From (14), we know that the average task response latency depends on two sets of variables, i.e. the SFs deployed on each UAVFog node, and the sub-tasks executed on each UAVFog node. They are integer variables and independent. Therefore, (14) is NP-hard and can not be solved using polynomial optimization methods. To obtain a solution in an acceptable time, we develop three heuristic algorithms. Specifically, we divide the whole deployment and scheduling algorithm into three stages: (i) dependency and topology-aware SF deployment; (ii) context-aware greedy task scheduling; and (iii) congestion-aware SF reallocation.

Before the deployment of the UAVFog nodes, one has to decide the deployment of the SFs on each UAVFog node. Based on the constraints of (18), every SF should have at least one instance in the UAVFog network, and the deployed services on a UAVFog node can not exceed its capacity. In the dependency and topology-aware SF deployment algorithm, the SFs are assigned to the UAVFog nodes based on their usage frequency. Then, the context-aware greedy task scheduling algorithm schedules the arrived tasks at a UAVFog node to achieve the shortest expected completion time. Finally, the congestion-aware SF reallocation algorithm is designed to re-deploy the SFs in case of congestion or idleness.

5.2 Dependency and topology-aware SF deployment

Although the task offloading behaviors of mobile devices in the disaster-hit area is unknown in advance. However, the frequently-requested tasks and their DAGs are known in advance and can be implemented on the UAVFog nodes before deployment. The popularity-based method has been widely adopted in machine learning and edge computing area for caching contents or predicting Web page accessing [12, 18, 21]. Based on the DAGs of the frequent-requested tasks, one can predict and obtain the popularity of the SFs. This assumption can be analogized to the observation that people usually use very limited number (for instance, less than 100) of APPs on their smartphone although millions of APPs are available in the APP store. To be specific, we assume that a small subset of the tasks, i.e. $\mathcal{D} \subset \mathcal{G}$ are known. In the example DAGs shown in Figure 4, sub-task 1 (f_1) and sub-task 2 (f_2) will be executed in a higher probability than sub-task 3-9 since they are included in both G_1 and G_2.

Then, one can assign the often-used SFs to the UAVFog nodes that could be accessed with low latency. From the FANET perspective, the most popular SFs should be placed at the "topology center" of the network, whose average distance to all other UAVFog nodes is the shortest. Moreover, the higher an SF or sub-task's popularity value, the closer it will be placed to the "topology center". In the example network shown in Figure 6, f_1 and f_2 are placed on U_{13}, i.e. the "topology center" of the network. The centrality of a UAVFog node is measured by its average distance to all other UAVFog nodes in the FANET. Based on this definition, U_{13} in Figure 6 has the highest centrality value, while U_1, U_5, U_{21}, and U_{25} have the lowest centrality value.

After deciding the deployment location of each SF, the dependencies between SFs are considered because data needs to be transmitted between dependent SFs. This process incurs transmission latency that depends on the bandwidth/distance between their host UAVFog nodes. For example, f_3 and f_5 should be placed on UAVFog nodes close to each other to reduce the data transmission latency, since there is data transmission between f_3 and f_5 in G_1 as shown in Figure 4. In this way, the transmission latency is reduced through considering the dependency between SFs in our initial deployment. Algorithm 1 illustrates our Dependency and topology-aware SF deployment (DeToSFD) method.

In Algorithm 1, Step 1-12 calculates the number of occurences of each SF in the given set of frequent-requested DAGs. Afterwards, Step 13 ranks the SFs based on their request frequency in a descending order. Then, the centrality values of all UAVFog nodes are calculated based on the given topology of the FANET. Step 18-27 deploy the most frequently-requested SFs on the UAVFog nodes with higher centrality values. The even placement of SFs on UAVFog nodes can achieve load balancing between UAVFog nodes in the initial deployment. Step 29 returns the mapping results of the SFs to UAVFog nodes.

5.3 Context-aware greedy task scheduling

Given the deployed SFs on all UAVFog nodes at a time, for an arrived task G, our next objective is to dispatch its sub-tasks to UAVFog nodes that have the corresponding SFs. When there is only one instance for each SF, it is straightforward to allocate a sub-task f in G to the UAVFog node that has the corresponding SF instance. When there are multiple SF instances for the sub-task f, one has to select the best SF instance from all available ones. Conducting an exhaustive search from all feasible scheduling decisions is time-consuming and even infeasible. In the worst case, M^n searches are needed for a task with n sub-tasks.

A heuristic algorithm is proposed to schedule the arrived tasks. For each sub-task f of an arrived task G, all of the UAVFog nodes that have the corresponding SF are identified first. Then, the expected completion time of f on the candidates is calculated. Finally, the UAVFog node with the smallest expected completion time is chosen as the execution node. A random choice is made if multiple candidates have the same completion time.

Figure 7 shows an example of the context-aware greedy task scheduling method. In this figure, three UAVFog nodes, i.e. U_1, U_2, and U_3, are considered. The squares in each UAVFog node represent its SF instances, and the queue length of each SF. U_1 has f_1, f_2, f_5; and the queue lengths for those instances are 3, 2, and 1 respectively. On node U_2, f_1, f_3, and f_4 are deployed, and their queue lengths are 1, 2, and 1. U_3 hosts f_5, f_6, and f_8, and their respective queue lengths are 2, 2, and 1. When G_1 arrives at U_1, it first has to decides the execution node for f_1 based on the DAG of G_1 shown in Figure 4. At this moment, there are two candidates, U_1 and U_2. Therefore, the expected completion time is calculated based on transmission time, waiting-for-transmission time, execution time, and waiting-for-execution-time in the sub-task queue. Here, U_2 is chosen since it has lower expected

completion time for f_1. Afterwards, a greedy scheduling is conducted for sub-task f_2, and U_1 is selected since no other candidate is available. In this way, the target UAVFog nodes for all sub-tasks are determined for offloading. These information is synchronized with other UAVFog nodes to enable the same understanding of the running status of the system.

Algorithm 1 Dependency and Topology-aware SF Deployment (DeToSFD).

Input: The DAG of frequent-requested tasks, i.e $\mathcal{D}$
The set of SFs, i.e. $\mathcal{F} = \{ f_1, f_2, \ldots, f_K \}$
The topology of the FANET
The set of UAVFog nodes $\mathcal{U} = \{u_1, u_2, \ldots, u_M\}$
Output: The SFs deployed on each UAVFog node

foreach *SF* f *in* $\mathcal{F}$ **do**
  F[f] = 0
end foreach
foreach *Task* G *in* $\mathcal{D}$ **do**
  foreach *SF* f *in* $\mathcal{F}$ **do**
    if G *contains* f **then**
      F[f] = F[f] + 1
    else
      continue
    end if
  end foreach
end foreach
Rank the number of occurrences of all SFs in vector F[1 ... K], i.e. popularity, in descending order
Let the set of SFs be $\mathcal{F}'$ after ranking operations
Calculate all UAVFog nodes' centrality values based on the given topology
Rank all UAVFog node based on their centrality values in descending order
Let the set of UAVFog nodes be $\mathcal{U}'$ after ranking operations
while $\mathcal{F}'$ *is not null* **do**
  if $|\mathcal{F}'| \geq \lceil \frac{K}{M} \rceil$ **then**
    $\sigma = \lceil \frac{K}{M} \rceil$
  else
    $\sigma = |\mathcal{F}'|$
  end if
  Place the most σ SFs with the biggest occurrences on the UAVFog node with the highest centrality value in $\mathcal{U}'$
  Place the selected σ SFs in $\mathcal{F}$ on the selected UAVFog nodes in $\mathcal{U}$
  Remove the most σ SFs from $\mathcal{F}'$
  Remove the UAVFog node with the highest centrality value in $\mathcal{U}'$
end while
Return the set of SFs placed on each UAVFog node in $\mathcal{U}$

Algorithm 2 illustrates the pseudo code of the context-aware greedy task scheduling. The inputs include the set of UAVFog nodes, the running SFs on each UAVFog node, and the queue lengths on each UAVFog node, and the queue length of the wireless links between any two UAVFog nodes. For each sub-task f in an arrived task G, Step 2-10 find all the UAVFog nodes that have the required SF instance of f. In Step 11, the shortest transmission

and execution time of f are calculated based on the completion time of all its parents. For each candidate of f, Step 12-18 derive the expected completion time of f through adding its waiting-for-transmission time, transmission time, waiting-for-execution time, and execution time. The candidate that has the shortest expected completion time is chosen as the scheduled UAVFog node of f in Step 19. Step 20 records the expected completion time of f. Step 21-22 update the queue information on the involved UAVFog nodes. In Step 24, the scheduler notifies other UAVFog nodes about its decision. Step 25 returns the scheduling results.

Algorithm 2 Context-aware Greedy Task Scheduling (CoGTS).

Input: The DAG of an arrived task G
The set of UAVFog nodes $\mathcal{U} = \{u_1, u_2, \ldots, u_M\}$
The set of SF instances on each UAVFog node and each SF's queue length
The queue length of the wireless link between any two given UAVFog nodes
Output: The scheduled UAVFog node for each sub-task in G

1 **foreach** *sub-task f in G's DAG* **do**
2 The set of f's candidate UAVFog nodes is set to be O
3 O = []
4 **foreach** U_i *in* $\mathcal{U}$ **do**
5 **if** U_i *has f's corresponding SF instance* **then**
6 Add U_i as one candidate into $\mathcal{O}$
7 **else**
8 **continue**
9 **end if**
10 **end foreach**
11 Determine f's eariliest transmission and execution times based on all its parents' completion time
12 **foreach** *Candidate O_i in $\mathcal{O}$* **do**
13 Calculate the expected waiting-for-transmission time using the queue lengths from f's all parent SFs' execution UAVFog nodes to O_i
14 Calculate the expected transmission time based on the bandwidth from f's all parent SFs' execution UAVFog nodes to O_i and the weights on G's DAG
15 Calculate the expected waiting-for-execution time using the queue length of f's desired SF instance on O_i
16 Calculate the expected execution time based on f's required resources and the allocated resources to f's desired SF instance on O_i
17 Calculate f's expected completion time, i.e. t_i, for executing it on O_i
18 **end foreach**
19 Choose the UAVFog node with the least expected completion time, for instance O_f, as the scheduled execution UAVFog node for f from $\mathcal{O}$
20 Record f's completion time for later usage to determine its children's earliest start time
21 Update the transmission queue length on the link from f's all parent SFs' execution UAVFog nodes to O_f
22 Update the execution queue length of f's desired SF instance on O_f
23 **end foreach**
24 Notify the scheduled results to all other UAVFog nodes
25 **Return** all G's sub-tasks' scheduled UAVFog nodes

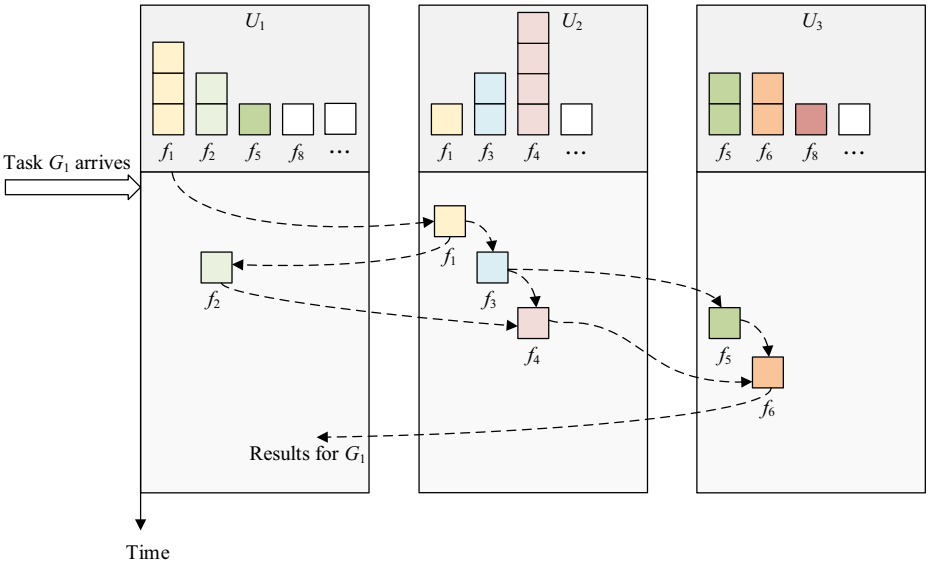


Figure 7 An example of the task scheduling

5.4 Congestion-aware SF reallocation

In the initial deployment of SFs on the UAVFog nodes, at least one instance of each SF is ensured. However, with the arrival of tasks, the load on an SF instance may be heavy and a sub-task queue needs to be maintained. On the other hand, the idle resources on the UAVFog nodes can not be utilized. Putting these two factors together, the adjustment of the placement of SFs is necessary. A UAVFog node has three possible operations:

1) Add a new SF instance: an SF image is loaded from the storage space to run as a new SF instance, and its needed resource is allocated to the instance.
2) Delete an existing SF instance: an existing SF instance is removed and its occupied resource is released for future usage.
3) Replace an existing SF instance: an existing SF instance is removed and a new SF instance is started.

To determine the time for adding or deleting an SF instance, several events need to be defined for triggering the re-allocation behaviors.

1) An SF instance is overloaded: when the queue length of an SF instance exceeds a threshold δ, the queuing latency would be long and may continue to increase.
2) An SF instance is idle for a long time: an SF instance has not been scheduled for a long time that exceeds δ_t.

In the case of former event, i.e. an SF execution encounters congestion, more SF instances are needed to relieve the congested instance. In this case, the adjustment may involve adding a new instance or replacing an existing instance. For example, the length of the sub-task

queue at the UAVFog node U_2 is 4 in Figure 7, and a new SF instance needs to be loaded on U_2 or another UAVFog node if the threshold $\delta = 3$. In the case of latter event, the idle SF instance could be safely removed as long as there is at least one active instance with the same function in the FANET. For instance, in Figure 7, both U_1 and U_3 have an f_8 instance, and no sub-task is running on the f_8 instance on U_1. Therefore, one could safely remove the f_8 instance on U_1 if it has been idle for a long time.

Algorithm 3 illustrates the congestion-aware SF reallocation. For a specific UAVFog node U_i and each SF instance f on U_i, Step 3 checks if the queue length of this instance exceeds δ. If the outcome is true, Step 4-15 examine all its neighbor UAVFog node for idle resource to run another f's instance. If a UAVFog node with idle resource is found, Step 7-9 start a new instance on the neighbor UAVFog node, and reduce the queue length on U_i by half by migrating the sub-tasks in the queue on U_i to the new instance. If no one-hop neighbor with idle resource is found, two-hop neighbors will be examined and so forth in Step 17. Step 20-35 check idle instances on U_i. If an idle instance is found, Step 22-28 check the one-hop neighbors of U_i for searching another instance of f. The idle instance on U_i will be removed if another instance is found in the neighbors. Otherwise, Step 31 examines two-hop neighbors. Step 37 synchronizes the SF adjustments on U_i to other UAVFog nodes.

6 Experiments and results

6.1 Experimental settings

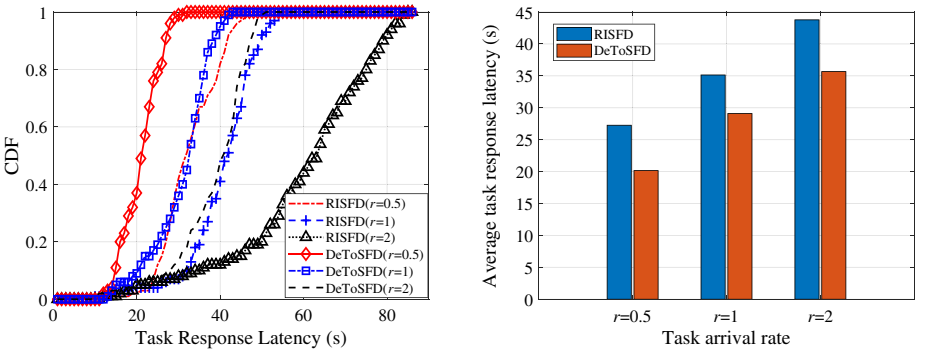
An event-driven simulator is developed to evaluate the performance of three algorithms presented in Section 5. 25 UAVFog nodes are deployed in a 2000m $\times$ 2000m area, and the topology is shown in Figure 6. Each UAVFog node serves a grid area with a height of 50m, and locates at the center of the grid. Each UAVFog node could host up to 5 SFs, and each of them occupies the same amount of resources. The resource of a UAVFog node is abstracted to one dimension, and varies from 25 to 50. A wireless link is built between two adjacent UAVFog nodes, and the transmission parameters are chosen as those in reference [22]. The derived bandwidth between two neighbors in horizontal (for example, U_1 and U_2) or vertical directions (for example, U_1 and U_6) is around 2.8Mbps; in contrast, the bandwidth of the links between two neighbors in diagonal or anti-diagonal directions is 2Mbps. The DAG of each arrived task contains 5, 6, or 7 sub-tasks, and there are 50 different SFs in total. The resource demand by a sub-task or SF varies from 3 to 10. The weights on the edge between two sub-tasks fall between 0.6Mbps to 2Mbps. The arrival rate of the tasks varies from 0.5 to 2 tasks per second. 50 repeated experiments are conducted for each group of parameter settings, and the average value of all experiments are adopted as the final results.

A random initial SF deployment (RISFD) algorithm is chosen as the benchmark for evaluating DeToSFD algorithm. In RISFD, SFs are randomly deployed onto UAVFog nodes. CoGTS algorithm is compared with a random task scheduling (RTS) algorithm. In RTS, sub-tasks are randomly scheduled to their candidates. Finally, the performance of the UAVFog system is evaluated with and without deploying CoSFR algorithm. The response latency and average response latency of the arrived tasks defined in (13) are chosen as the comparison metrics.

Algorithm 3 Congestion-aware SF Reallocation (CoSFR).

Input: The set of UAVFog nodes $\mathcal{U} = \{u_1, u_2, \ldots, u_M\}$
The set of SF instances on each UAVFog node and each SF's queue length
The reallocation threshold for the queue length δ
The reallocation threshold for the idle time δ_t
Output: The updated SF instances running on each UAVFog node

foreach *UAVFog node* U_i *in* $\mathcal{U}$ **do**
  foreach *SF instance* f *on* U_i **do**
    if *The queue length of* f*'s instance on* $U_i > \delta$ **then**
      foreach U_i*'s one-hop neighbor* U_j *UAVFog node* **do**
        Add_a_new_instance = False
        if *There is idle resource for running a instance for* f **then**
          Start a intance for f on U_j
          Reduce the queue length of f's instance on U_i by half
          Migrate the sub-tasks in f's instance on U_i to the new instance on U_j
          Add_a_new_instance = True
          Break
        else
          Continue
        end if
      end foreach
      if *Add_a_new_instance* == *False* **then**
        Check U_i's two-hop neighbor UAVFog nodes in the same way and so forth
      end if
    else
      if f*'s instance's queue length is 0* **then**
        if f*'s instance has not been called for longer than* δ_t **then**
          foreach U_i*'s neighbor* U_j *UAVFog node* **do**
            Remove_an_instance = False
            if *There is a* f*'s instance on* U_j **then**
              Remove the f's instance on U_i
              Remove_an_instance = True
              Break
            end if
          end foreach
          if *Remove_an_instance* == *False* **then**
            Check U_i's two-hop neighbor UAVFog nodes in the same way and so forth
          end if
        end if
      end if
    end if
  end foreach
  Update all the SF adjustment to all other UAVFog nodes
end foreach
Return all $\mathcal{U}$'s updated SF instances



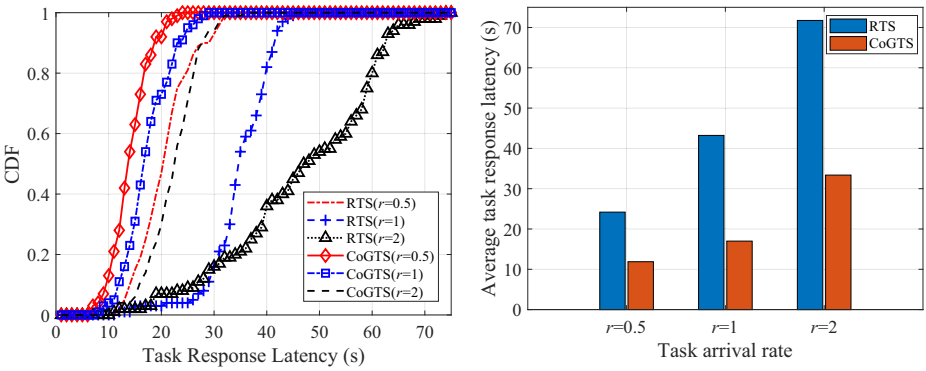
(a) CDF of algorithms with different r. (b) Average response latency of algorithms.

Figure 8 The task response latency when adopting DeToSFD and RISFD algorithms

6.2 Experimental results and analysis

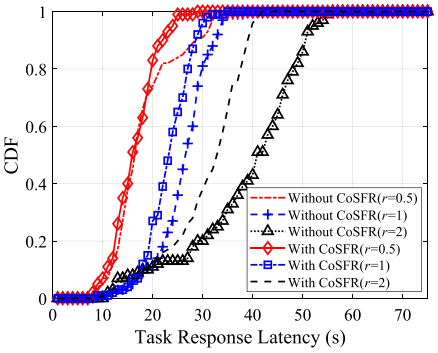
According to DeToSFD algorithm, two SFs are deployed on each UAVFog node since there are 50 SFs and 25 UAVFog nodes. Figure 8 illustrates the experimental results when RISFD and DeToSFD algorithms are adopted in the initial SF deployment stage. Different task arrival rate, r=0.5, r=1, and r=2, are adopted to investigate different system loads. From Figure 8b, we can know that DeToSFD algorithm's average task response latency is remarkably lower than that of RISFD algorithm. This is due to the fact that DeToSFD considers the relationship between the "popularity" of an SF and the centrality of its deployed UAVFog. Placing popular SFs at the center of the network greatly reduces the latency incurred by data transmission among UAVFog nodes. Moreover, Figure 8b illustrates that heavier loads (i.e. larger r) result larger average task response latency. This increment is due to the increased queue length at the SFs because of the large number of arrived sub-tasks.

To evaluate the performance of CoGTS algorithm, based on the deployment results of DeToSFD algorithm, another three random chosen SFs are loaded on each UAVFog node to give the scheduling algorithm multiple candidates for each given SF. Figure 9 illustrates

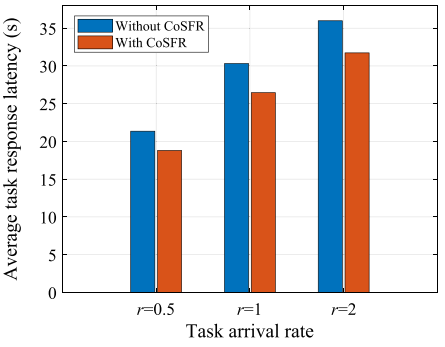


(a) CDF of algorithms with different r. (b) Average response latency of algorithms.

Figure 9 The task response latency when adopting CoGTS and RTS algorithms



(a) CDF of algorithms with different r.



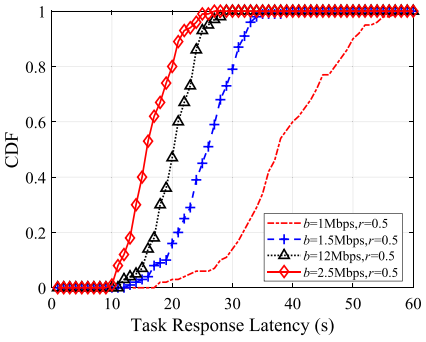
(b) Average response latency of algorithms.

Figure 10 The task response latency with and without CoSFR algorithms when the total number of arrived tasks is 100

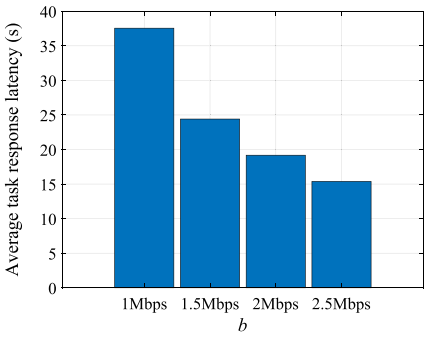
the experimental results when RTS and CoGTS algorithms are adopted. CoGTS algorithm could greatly reduce the average task response latency with different task arrival rate. Moreover, the average task response latency of CoGTS algorithm is less than half of that of RTS algorithm. Similar observations can be obtained from Figure 9a because CoGTS algorithm can effectively find the near-optimal scheduling in a greedy way. In contrast, the bad performance of RTS algorithm is due to its ignorance of the task running status on UAVFog nodes.

To evaluate the performance of the CoSFR algorithm, the number of arrived tasks increases from 100 to 400. This will lead to long queue length at many SF instances. Figure 10 shows the results of the UAVFog system with and without CoSFR algorithm. From Figure 10 one can see that deploying CoSFR algorithm could reduce the average response latency by roughly 5 seconds. The reduction of the response latency comes from the reduced queuing delay at heavy-loaded UAVFog nodes. By reallocating often-used or congested SFs, CoSFR algorithm achieves better load balance among UAVFog nodes.

To investigate the impacts of the transmission bandwidth between UAVFog nodes, the bandwidth of the diagonal or anti-diagonal link, i.e. b, is set to be 1Mbps, 1.5Mbps, 2Mbps,

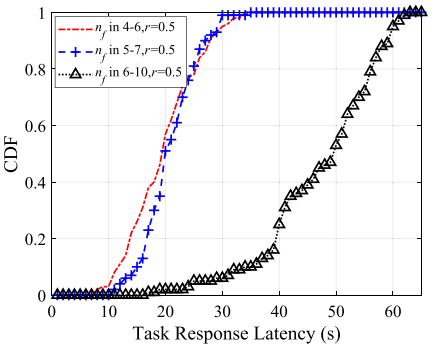


(a) CDF of algorithms' task response latency with different r.

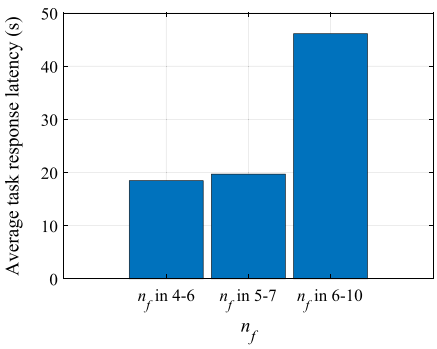


(b) Average response latency.

Figure 11 The task response latency with different b when the total number of arrived tasks is 100



(a) CDF of algorithms' task response latency with different n_f.



(b) Average response latency.

Figure 12 The task response latency with different n_f when the total number of arrived tasks is 100

and 2.5Mbps respectively; correspondingly, the bandwidth of the horizontal and vertical link, roughly $\sqrt{2}b$ is set to be 1.4Mbps, 2.1Mbps, 2.8Mbps, and 3.5Mbps, respectively. Figure 11a shows the response latency of 100 arrived tasks. Figure 11b illustrates the average latency with different b. From Figure 11, it can be seen that higher b implies smaller latency because higher b could reduce both t^G_{trans} and t^G_{wait} in (13). This reduction is related to the data volume transmitted among dependent sub-tasks. Generally speaking, the more data transmitted, the greater the latency is reduced.

To show the impacts of the average number of sub-tasks, denoted as n_f, in the DAG of each task, a series of experiments are conducted. Figure 12 illustrates the results when n_f varies from 4 to 6, 5 to 7, and 6 to 10, respectively. One can see that with the increase of n_f, the response latency grows because more data transmission and processing are needed. A sharp increase is observed when n_f is in the range of 6-10. This is due to the congestion of the transmission link and the SF instances.

7 Conclusion and future work

To support efficient disaster-response operations, three algorithms are presented in this paper for UAVFog-assisted disaster-response architecture. After introducing the architecture, the structure and the collaboration mechanism between UAVFog nodes are detailed. Afterwards, the service functions (SFs) deployment and task scheduling problem is formulated. Then, an algorithm, called Dependency and topology-aware SF deployment (DeToSFD), for determining the initial deployment location of each SF is designed. To schedule an arrived task, the context-aware greedy task scheduling (CoGTS), is put forward. The last algorithm, called congestion-aware SF reallocation (CoSFR), is proposed to reallocate SFs in case of congestion at a SF instance. A series of experiments have been conducted to investigate the performance of three algorithms. Experimental results have shown that these algorithms can greatly reduce the task response latency in the UAVFog systems.

In the future, the energy consumption of the UAVFog nodes will be considered, and a prototype will be built based on DJI drones to investigate the design of UAVFog nodes. Moreover, the reliability of the UAVFog nodes will be incorporated into the algorithm design.

References

1. Al-Rakhami, M., Gumaei, A., Alsahli, M., Hassan, M.M., Alamri, A., Guerrieri, A., Fortino, G.: LW-Coedge: A lightweight virtualization model and collaboration process for edge computing. World Wide Web **23**, 1341–1360 (2020)
2. Al-Turjman, F., Abujubbeh, M., Malekloo, A., Mostarda, L.: UAVs assessment in software-defined iot networks: An overview. Comput. Commun. **150**, 519–536 (2020)
3. Alves, M.P., Delicato, F.C., Santos, I.L., Pires, P.F.: LW-Coedge: a lightweight virtualization model and collaboration process for edge computing. World Wide Web **23**, 1127–1175 (2020)
4. Arbia, D.B., Alam, M.M., Kadri, A., Hamida, E.B., Attia, R.: Enhanced IoT-based end-to-end emergency and disaster relief system. J. Sens. Actuator Netw. 6(3) (2017)
5. Cheng, N., Xu, W., Shi, W., Zhou, Y., Lu, N., Zhou, H., Shen, X.: Air-ground integrated mobile edge networks: Architecture, challenges, and opportunities. IEEE Commun. Mag. **56**(8), 26–32 (2018)
6. Deruyck, M., Wyckmans, J., Joseph, W., Martens, L.: Designing UAV-aided emergency networks for large-scale disaster scenarios. EURASIP J. Wirel. Commun. Netw. **2018**(1), 79 (2018)
7. Erdelj, M., Natalizio, E., Chowdhury, K.R., Akyildiz, I.F.: Help from the sky: Leveraging UAVs for disaster management. IEEE Pervasive Comput. **16**(1), 24–32 (2017)
8. Ge, Y.-F., Yu, W.-J., Cao, J., Wang, H., Zhan, Z.-H., Zhang, Y., Zhang, J.: Distributed memetic algorithm for outsourced database fragmentation. IEEE Trans Cybern 1–14 (2020)
9. Hu, Q., Cai, Y., Yu, G., Qin, Z., Zhao, M., Li, G.Y.: Joint offloading and trajectory design for UAV-enabled mobile edge computing systems. IEEE Int Things J **6**(2), 1879–1892 (2018)
10. Hu, J., Jiang, M., Zhang, Q., Li, Q., Qin, J.: Joint optimization of UAV position, time slot allocation, and computation task partition in multiuser aerial mobile-edge computing systems. IEEE Trans. Veh. Technol. **68**(7), 7231–7235 (2019)
11. Hua, M., Wang, Y., Li, C., Huang, Y., Yang, L.: UAV-Aided mobile edge computing systems with one by one access scheme. IEEE Trans. Green Commun. Netw. **3**(3), 664–678 (2019)
12. Huang, J., Peng, M., Wang, H., Cao, J., Gao, W., Zhang, X.: A probabilistic method for emerging topic tracking in microblog stream. World Wide Web **20**, 325–350 (2017)
13. Król, M., Natalizio, E., Zema, N.R.: Tag-Based Data Exchange in Disaster Relief Scenarios. In: 2017 international conference on computing, networking and communications (ICNC), pp. 1068–1072 (2017)
14. Li, J.-Y., Zhan, Z.-H., Wang, H., Zhang, J.: Data-driven evolutionary algorithm with perturbation-based ensemble surrogates. IEEE Trans Cybern 1–13 (2020)
15. Liu, B., Zhu, Q., Zhu, H.: Trajectory optimization and resource allocation for UAV-assisted relaying communications. Wirel. Netw. **26**(1), 739–749 (2020)
16. Lu, Z., Cao, G., La Porta, T.: Teamphone: Networking smartphones for disaster recovery. IEEE Trans. Mob. Comput. **16**(12), 3554–3567 (2017)
17. Mozaffari, M., Saad, W., Bennis, M., Debbah, M.: Mobile unmanned aerial vehicles (uavs) for energy-efficient internet of things communications. IEEE Trans. Wirel. Commun. **16**(11), 7574–7589 (2017)
18. Peng, M., Zeng, G., Sun, Z., Huang, J., Wang, H., Tian, G.: Personalized app recommendation based on app permissions. World Wide Web **21**, 89–104 (2018)
19. Tang, Q., Chang, L., Yang, K., Wang, K., Wang, J., Sharma, P.K.: Task number maximization offloading strategy seamlessly adapted to UAV scenario. Comput. Commun. **151**, 19–30 (2020)
20. Wei, X., Li, L., Tang, C., Subramaniam, S.: Uavfog-assisted data-driven disaster response: Architecture, Use Case, and Challenges. In: 21St international conference on web information systems engineering (WISE2020), pp. 591–606 (2020)
21. Wei, X., Liu, J., Wang, Y., Tang, C., Hu, Y.: Wireless edge caching based on content similarity in dynamic environments. J. Syst. Archit. **115**, 102000 (2021). https://doi.org/10.1016/j.sysarc.2021.102000
22. Wei, X., Tang, C., Fan, J., Subramaniam, S.: Joint optimization of energy consumption and delay in cloud-to-thing continuum. IEEE Int. Things J. **6**(2), 2325–2337 (2019)
23. Wu, Q., Zeng, Y., Zhang, R.: Joint trajectory and communication design for multi-UAV enabled wireless networks. IEEE Trans. Wirel. Commun. **17**(3), 2109–2121 (2018)
24. Yang, Z., Pan, C., Wang, K., Shikh-Bahaei, M.: Energy efficient resource allocation in UAV-enabled mobile edge computing networks. IEEE Trans. Wirel. Commun. **18**(9), 4576–4589 (2019)
25. Zeng, Y., Xu, J., Zhang, R.: Energy minimization for wireless communication with rotary-wing UAV. IEEE Trans. Wirel. Commun. **18**(4), 2329–2345 (2019)

Affiliations

Xianglin Wei[1] · Li Li[2] · Lingfeng Cai[1] · Chaogang Tang[3] · Suresh Subramaniam[2]

Suresh Subramaniam
suresh@gwu.edu

Xianglin Wei
wei_xianglin@163.com

Li Li
lili1986@gwu.edu

1 The 63rd Research Institute, National University of Defense Technology, Nanjing 210007, China

2 Department of Electrical and Computer Engineering, George Washington University, Washington, DC 20052, USA

3 School of Computer Science and Technology, China University of Mining and Technology, Xuzhou 221116, China