Priority-Driven Task Processing in UAV-Assisted Software-Defined Edge Networks

Onur Kalinagac*§, Gürkan Gür *, Fatih Alagöz §
* Zurich University of Applied Sciences (ZHAW) InIT, Winterthur, Switzerland, name.surname@zhaw.ch
§Dept. of Computer Engineering, Bogazici University, Istanbul, Turkey, name.surname@boun.edu.tr

***Abstract*—For providing wireless connectivity and facilitating a capacity boost under transient high service load situations, a substitute or auxiliary fast-deployable network is instrumental. Unmanned Aerial Vehicle (UAV) networks are well suited for such needs owing to their high mobility and agility. This paper considers a software-defined edge network consisting of UAVs equipped with wireless access points, which serve mobile users with latency-sensitive workload in an edge-to-cloud continuum setting. It investigates the task offloading paradigm to provide prioritized services via this on-demand aerial network. Accordingly, a task processing optimization model is defined to minimize the overall penalty calculated based on priority-weighted delay values against a priori defined task deadlines. Since the defined assignment problem is NP-hard, tailored heuristic models are proposed and evaluated to study how the system performs under different operating conditions.**

I. Introduction

Unmanned Aerial Vehicles (UAVs) can facilitate a local network infrastructure when a quickly deployable solution is required. That may happen in situations such as short-term additional capacity requirements, post-disaster communications and dynamic data collection use cases [1]. In that regard, they have three critical characteristics which make them useful [2]. Firstly, located in the sky, they have a higher probability of connecting ground nodes and other UAVs via Line-of-Sight (LoS) links compared to terrestrial vehicles and infrastructure. Secondly, contrary to stationary ground infrastructure, UAVs can be dynamically deployed in response to emerging real-time requirements. Thirdly, a swarm of UAVs can construct scalable multi-UAV networks and provide ground users with pervasive connectivity in a flexible setting.

Applicable use cases in 5G and future networks for UAV networks also include novel intelligent mobility solutions such as self-driving cars as well as mission-critical ones relying on connected services, such as emergency communications or cloud robotics for search&rescue [3], [4]. In that case, UAVs can assist vehicular networks by providing cloud access, packet relaying or edge computing services. In this paper, we consider such a UAV integrated edge network and focus on the use case in which UAVs create a network for vehicles for emergency or ad hoc communications. Furthermore, network softwarization allows our system to cope with network management issues such as mobility-based frequent topology changes and dynamic system states [5]. A post-disaster operations management scenario is designed in which the network offers Vehicle-to-Vehicle (V2V) and Vehicle-to-Cloud (V2C) task offloading services to users, e.g., search&rescue team vehicles.

In such a time-sensitive and mission critical case, a key challenge is how to decrease delays while considering resource constraints. In this work, we focus on this research problem and develop a task offloading scheme entailing different heuristics in our UAV-assisted software-defined edge network. Under the conditions of limited network and resource availability, priority values are assigned to computational tasks to prioritize urgent and critical ones over others. Also, a processing deadline is introduced for each task as the key QoS criterion for performance evaluation.

As a key contribution, we propose a mathematical model to make decisions about centralized offloading of prioritized tasks in an agile environment. The model's objective function is to minimize average penalty scores caused by task completion time. Moreover, we implement three heuristic algorithms and a branch&bound quasi-optimal task offloading algorithm for the given objective function. We also investigate how such a system performs under different task load levels by emulating the system on the Mininet-Wifi [6]. As open-source contributions, we made some crucial improvements on Mininet-Wifi and its wireless medium simulation tool dependency `wmediumd` in order to support independent and simultaneous Wi-Fi packet transfers and contributed these developments to the open-source domain.

The rest of the paper is organized as follows. In Section II, we present an overview of related studies in UAV, task offloading and SDN. In Section III, we introduce the system-related models. Next, we formulate our problem in Section IV. Then, we explain each of the proposed task offloading algorithms in Section V. In Section VI, we introduce our simulation environment and its components. Later on, simulation results and performance measures are presented in the same section. We conclude the paper in Section VII.

II. Related Work

The current research works elaborate on different aspects of UAVs, task offloading and SDN. Similar to our study, Jia *et al.* [7] use UAVs as flying base stations to provide communications to rescue vehicles in disaster-affected areas. They look into the relationship between UAV altitude and vehicle connectivity for a single UAV scenario. In the case

 DOI: 10.1109/GIIS56506.2022.9936953

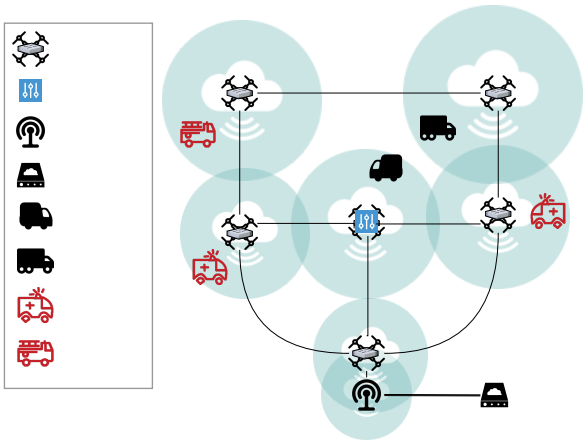

Fig. 1. The network model.

of many UAVs, they investigate the smallest number of UAVs required to achieve a certain level of vehicle communication. In [2], a case study on UAV-aided Vehicular Network (VN) architecture is presented, in which vehicles drive along a bi-directional two-lane straight highway while two mobile drones are flying over them to form a relay platform. They compared UAV-aided VN's throughput and latency performance to an 802.11p-based vehicular network, demonstrating its effectiveness. These studies focus on infrastructure level rather than possible applications such as task offloading on the constructed network.

Chen *et al.* [8] propose a post-disaster rescue computation task offloading scheme for cooperative UAVs. UAV computation tasks are offloaded to Unmanned Ground Vehicles (UGVs) that have idle computation resources. Both UAVs and UGVs seek their own maximum profits in its setting. Specifically, the stable matching algorithm has been proposed to transform the computation task offloading problem into a two-sided matching problem, taking into account that the algorithm iteratively solves the problem while maximizing the utility of UAVs. In a multi-UAV-assisted road traffic scenario, researchers [9] construct a three-player sequential game-based computational offloading methodology for processing UAV data. The computation delay, energy overhead, and communication&computation costs are all part of the utility function being studied. Contrarily, task owners in our model are mobile ground units and task offloading decisions are centralized while UAVs only provide the network.

In [4], an SDN-enabled UAV-assisted vehicular computation offloading cost optimization framework is defined. Vehicles can offload computationally intensive and time-sensitive tasks to reduce execution time and overall energy usage. Each vehicle receives network traffic data from the SDN controller. Vehicles make decentralized offloading decisions based on their own interests and global data. The stated problem is very similar to our work, but task offloading to other mobile ground vehicles and UAV mesh network is not possible in the given study contrary to ours.

TABLE I
SYSTEM PARAMETERS.

Symbol	Definition
U_i	The i^{th} UAV
V_i	The i^{th} vehicle
K_i	The i^{th} task
O_{i,V_j}	If V_j is the owner of K_i, 1; otherwise 0
α_i	The priority of K_i
D_i	The penalty point of K_i
C_i	If K_i is completed, 1; otherwise 0
p_{heavy}	The probability of task offloaded to the cloud if pool is heavily occupied
p_{medium}	The probability of task offloaded to the cloud if pool is moderately occupied
p_{light}	The probability of task offload to the cloud if pool is lightly occupied
Θ_{heavy}	The threshold value to classify the controller pool as heavily occupied
Θ_{light}	The threshold value to classify the controller pool as lightly occupied
Θ_{medium}	The threshold value to classify the controller pool as moderately occupied
S_{K_i}	The size of K_i
P_i	The i^{th} processor
β_i	The processing speed of P_i
β_{cloud}	The processing speed of cloud
S_{Q_i}	The queue size of P_i
Q_i	The queue of P_i
N_U	The number of UAVs
N_K	The number of tasks
N_P	The number of processors
z	The extra penalty point if the task deadline is missed
Time Related Parameters	
T	The number of time slots
$t_{total,i}$	The time elapsed to complete K_i
$t_{pool,i}$	The time elapsed to be assigned for K_i
$t_{tx,i}$	The time elapsed to transmit K_i
$t_{queue,i}$	The time elapsed on processor queue for K_i
$t_{process,i}$	The time elapsed to process K_i
$t_{deadline,i}$	The time budget for K_i to be processed without any QoS violation, i.e., penalty
b_{V_i}	The departure time slot of V_i
a_{K_i}	The arrival time slot of K_i
c_{K_i}	The completion time slot of K_i
$L_{K_i,P_j}(t)$	The link capacity between the owner of K_i and P_j at t
$L_{K_i,cloud}(t)$	The link capacity between the owner of K_i and the cloud server at t

III. SYSTEM MODEL

Our network model consists of N_U UAVs (U_i) which are mounted with wireless Access Points (APs) and have multiple Wi-Fi interfaces. For each UAV, one interface provides Wi-Fi service to terrestrial vehicles, while the others create mesh connections with nearby UAVs. One of the UAVs is connected to a ground base station (BS) which has a backhaul link to the Internet and relevant cloud services. One of the UAVs also has a computational unit mounted which runs an SDN controller on it and acts as the network controller which manages flow control of APs. This controller is operated by a service provider remotely. UAVs are not used to offload tasks; instead, they are primarily used for communication, which limits energy consumption. For the sake of simplicity, we do not consider energy constraints in our system. The network model and system parameters are shown in Fig. 1 and Table I, respectively.

An SDN controller application, namely offloading orchestrator, orchestrates all the task offloading decisions in a centralized way. There are two main types of terrestrial vehicles in the system. The first type (rescue&emergency) vehicles, which are task owners, signal their task information consisting of a deadline, priority factor, and size to the task pool of

the offloading orchestrator on the controller. The orchestrator application assigns these tasks to the second type (compute station) of vehicles, also called task processors, and informs both parties about the assignment. Only the identifier data for tasks are stored on the controller's task pool, not the task data itself meaning the data transfers are made from the task owner directly to the processor units.

The processor vehicles have their own limited-size queues to store unprocessed tasks. A processor may be used to download data of multiple tasks simultaneously according to the policy of the active task offloading algorithm. The processing speed and queue size are determined by the processor group (i.e., class-based processor capabilities). Similarly, the priority of a task is determined by the owner vehicle's group (i.e., class-based task priority). If a task is not completed by the deadline, the system receives a penalty proportional to the amount of time that has passed between the deadline and the task completion time. The mobility of vehicles is not linked to the task processing operation, i.e., a vehicle may leave the area, regardless of its tasks still being processed. The task is also sent back to the pool if the assigned processor vehicle leaves during packet transmission for that task. There are two cases for the situation that the task owner leaves the coverage area, depending on the deadline: The system receives an extra penalty score z if the deadline has already been missed; otherwise, it receives no penalty. To identify and determine the network location of vehicles, the orchestrator application listens for ARP packets. In addition, access points also send their associated stations' information periodically to the controller for vehicle departure or disconnection detection.

The average inter-arrival time for the tasks generated by the vehicles is exponentially distributed with rate λ. Upon a task arrival, if the task pool is not empty, the task can be offloaded/migrated to the cloud processing server with a probability (p_{heavy}, p_{medium}, p_{light}) depending on the occupancy of the task pool (Θ_{heavy}, Θ_{medium}, Θ_{light}). Essentially, we consider minimizing average prioritization-based weighted penalty points for task offloading management as a QoS Key Performance Indicator (KPI) and thus propose algorithms for that objective.

A. Channel Models

We chose two widely used path loss models [10]–[12] and applied them in our network simulations since there is no consensus on a proper path loss model for UAV-enabled networks [13]. In our simulation environment, calculated Received Signal Strength Indicator values (RSSI) are used to get probabilistic data rates from the RSSI-transmission mode matrix. The values in that matrix contain the probability of the Wi-Fi rate at the given RSSI value, i.e., 6, 9, 12, 18, 24, 36, 48 and 54 Mbps for the IEEE 802.11g standard.

1) Air-to-Ground (A2G) Communication Channel Model: This air-to-ground propagation model is based on the probability of line of sight and is used for our UAV-to-terrestrial-vehicle connections.

2) Air-to-Air (A2A) Communication Channel Model: For UAV-to-UAV communication, free-space channel model is utilized.

B. Mobility Model

UAVs are assumed to be anchored to fixed locations. However, they have small-scale mobility where these aerial systems are assumed to hover around these points in a limited space. Specifically, they are assumed to be forming a small-sized 8 figure by moving around two fixed points at the same altitude as the mobility model in [14].

IV. Problem Formulation

The problem that we address is the minimization of the average task penalty weighted by task priority in a software-defined edge network with integrated UAVs. Due to departures, some tasks may not be fulfilled even in the optimal scenario, which prevent us to use deadline as a constraint. As a result, we introduce penalty scoring system in which extra penalty score is added for unfilled tasks in order to encourage the completion of all tasks. The list of parameters that we utilize in the problem formulation is given in TABLE I.

A task's lifetime in our edge network is composed of various components due to pool wait, transmission time, processor queue wait and processing time. C_i variable represents if i^{th} task K_i is successfully processed. Parameter x_{ij} is equal to one if K_i is decided to be offloaded to P_j and similarly, y_i is equal to one if K_i is decided to be offloaded to the cloud. These decision variables are equal to zero otherwise. $t_{total,i}$ is the total time needed to complete K_i and defined as follows if and only if ($\leftrightarrow$) the task is completed which is the case C_i equals to 1:

$$C_i = \sum_{j=1}^{N_P} x_{ij} + y_i \qquad \forall i \tag{1}$$

$$t_{total,i} = t_{pool,i} + t_{tx,i} + t_{queue,i} + t_{process,i} \;\leftrightarrow\; C_i = 1 \tag{2}$$

$$t_{process,i} = S_{K_i}/(\sum_{j=1}^{N_P} \beta_i x_{ij} + \beta_{cloud} y_i) \;\leftrightarrow\; C_i = 1 \tag{3}$$

$$t_{tx,i} = S_{K_i}/(\sum_{j=1}^{N_P} L_{K_i,P_j}(t) x_{ij} + L_{K_i,cloud}(t) y_i) \;\leftrightarrow\; C_i = 1 \tag{4}$$

D_i is the penalty for K_i and calculated as:

$$D_i = \begin{cases} max(t_{total,i} - t_{deadline,i}, 0) & \text{if } C_i = 1 \\ 0 \quad \text{if } (C_i = 0) \wedge (O_{i,V_j} = 1) & \\ \quad \wedge (b_{V_j} < a_{K_i} + t_{deadline,i}) & \\ b_{V_j} - a_{K_i} - t_{deadline,i} + z & \\ \quad \text{if } (C_i = 0) \wedge (O_{i,V_j} = 1) & \\ \quad \wedge (b_{V_j} > a_{K_i} + t_{deadline,i}) & \end{cases} \tag{5}$$

 DOI: 10.1109/GIIS56506.2022.9936953

Now, let us present the optimization problem, which will be solved by the orchestrator application to decide on the task offloading at each time slot as follows:

$$w_i = \alpha_i \cdot D_i \tag{6}$$

$$\min_{x,y} \ \mathbb{D}(\lambda, \boldsymbol{R}) = \sum_{i=1}^{N_K} w_i / N_K \tag{7}$$

subject to:

$$\sum_{j=1}^{N_P} x_{ij} + y_i \leq 1 \qquad \forall i \tag{8}$$

$$x_{ij} \cdot c_{K_i} \leq b_{P_j} \qquad \forall i \tag{9}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, \forall j \tag{10}$$

$$y_i \in \{0, 1\} \qquad \forall i \tag{11}$$

Eq. 8 ensures that only one vehicle processor or cloud is assigned to a task. There is inequality because some tasks may not be completed due to the randomness of processor availability and task owner departure times. The task completion time slot, c_{K_i} in the left hand side of Eq. 9 is the sum of the task arrival time, a_{K_i} and $t_{total,i}$. Eq. 9 guarantees that the task should be completed before the processor departs. Because the departure times are not known during the run-time, only our quasi-optimal algorithm can use this constraint to guide decisions. For heuristics, a task is sent back to pool, if the constraint happens to be failed.

Task offloading decisions significantly affect each other's delay results, especially when related vehicles share the same APs or links for connectivity since available data rates for related vehicles drop. Thus each possible decision combination should be evaluated. This multi-user, multi-task problem is proved to be NP-hard [15] while the penalty scoring in our system model increases the complexity due to its nonlinear behavior upon task owner departures.

V. Task Offloading Algorithms

In our algorithm design, the system time is discretized into steps to perform offloading decisions on the generated time steps. Obviously, the exact time a vehicle will arrive/depart or a task arrives is unknown during the run-time. Performance comparisons are made using four different algorithms. The first three algorithms are greedy heuristics and are used during the simulation experiments. They make decisions according to the available data at each step, i.e., run in an online mode. They are namely *Aggressive-Wait*, *Aggressive Tx-Order*, and *Adaptive* offloading algorithms. The available data include vehicle-UAV associations, existing links' with their bandwidths and rough vehicle departure time. The final algorithm is named as *Quasi-Optimal* offloading algorithm and an oracle-type decision-maker since it operates by extracting better solutions from recorded past data for benchmarking. Since this algorithm cannot process all the possible solutions from all the data recorded due to combinatorial explosion, it is quasi-optimal by processing events in time windows.

a) Aggressive Wait Offloading Algorithm (AGG-1): The orchestrator application with the first greedy algorithm AGG-1 tries to offload tasks until all the queues on the available processors are full. New assignments to the same processor can be made before the current assigned task's data transfer is completed. The task assignment order is also applied on processing ordering in AGG-1 in order to complete the previously assigned task, probably, the one closer to its deadline, earlier. If the next assigned task is downloaded before the first one, the second task will be processed after the first is completed.

b) Aggressive Tx-Order Offloading Algorithm (AGG-2): The second greedy algorithm is similar to the first one, AGG-1. The only difference is that the task transmission completion order (Tx-Order), rather than the task assignment order as it was in AGG-1, determines the order in which tasks are processed; the earlier downloaded one is processed first. The rationale is to avoid wasting processing resources.

c) Adaptive Offloading Algorithm (ADP): In ADP, during an offloading decision, the processors with the tasks being transmitted are skipped, avoiding multiple transfers for the same processor. It aims to decrease the pressure on the limited network resources and improve prioritization since it leads to more tasks in the controller task pool TP and more processor-task combinations upon arrival of new processor nodes.

d) Quasi-optimal Offloading Algorithm (Q-OPT): The last approach is a branch&bound based algorithm that processes the whole simulation data to produce a sub-optimal yet close to optimal objective value for benchmarking our heuristics. It analyzes the past simulation data and functions as a oracle-type decision-maker. This processed data includes task details (e.g., size, deadline), vehicle positions, vehicle associated AP and RSSI values and task processor details.

In order to narrow down candidate solutions, first arrival and departure times of vehicles and their connection status per step are considered. For each task, possible processor assignments are determined as lists. Two more decisions, skipping, meaning failing on purpose, and offloading to the cloud are also added to these lists. Even the most under-loaded setup with meaningful results generates a huge candidate pool from the Cartesian product of decision possibilities, which cannot be handled due to combinatorial explosion. Thus, we have developed a time-windowed quasi-optimal processing approach. The solver processes and makes decisions on the tasks in the given time interval. The algorithm starts with the time window to check if the candidate solution pool can be processed. The algorithm shrinks the window end time (window length) until the number of decision combinations is less than a predefined threshold, $\Theta_{decision}$. The solver skips some of the combinations by bounding. After finding the optimal value for this interval, it saves the decision of the first task in this window and moves to the second time window, which is between the arrival time of the next task and the end of the simulation. It applies the same principle until all tasks are decided one by one.

 DOI: 10.1109/GIIS56506.2022.9936953

TABLE II
BASELINE SIMULATION PARAMETERS.

Parameter	Value
λ	0.1 $1/s$
N_U	9
Task size (randomly distributed)	30-40MB
$\overline{V}$	20 km/h
β_{cloud}	3MB/s
$t_{deadline,i}$ (randomly distributed)	50-120s
p_{heavy}	0.9
p_{medium}	0.7
p_{light}	0.3
Θ_{heavy}	10
Θ_{medium}	7
Θ_{light}	3
$\Theta_{decision}$	10M
P_{tx,U_i}, P_{tx,V_i}	23dBm
G_{tx,U_i}, G_{tx,V_i}, G_{rx,U_j}, G_{rx,V_j}	3dBm
σ^2	-91dBm
f	2.4GHz
ρ	2
η_{LoS}, η_{NLoS}	1dBm, 20dBm
z	60

TABLE III
VEHICLE PARAMETERS.

Role	Type	Proportion	Shape	Color	Priority	β_i	S_{Q_i}
Owner	A	0.1	Emergency	White	0.7	-	-
Owner	B	0.2	Firebrigade	Red	0.3	-	-
Processor	A	0.06	Trailer	Purple	-	2 MBps	100MB
Processor	B	0.1	Trailer	Cyan	-	1.5 MBps	70MB
-	-	0.54	Passenger	Pink	-	-	-

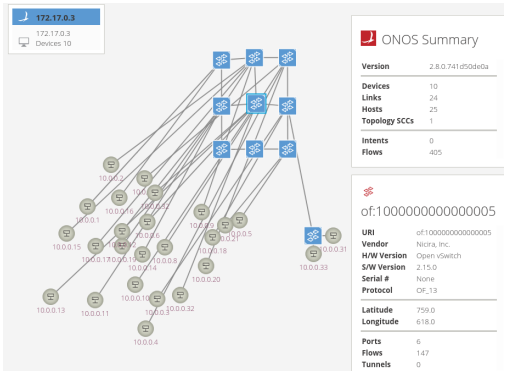

Fig. 2. A sample network topology in the simulations.

VI. PERFORMANCE EVALUATION

We use a machine with Intel Core i7-10875H (2.30GHz $\times$ 8), 16GB RAM, and 256GB SSD storage to run our tests. We use ONOS [16] v2.8 on Ubuntu 21.04 as SDN controller. Mininet-WiFi is used to create host devices, virtual stations and APs. For virtual switch emulation, Open vSwitchv2.15 is utilized [17]. For traffic generation from vehicles in our experiments, iPerf v3.10 is adopted [18]. The network control and packet delivery applications were developed in Python v3.9.5. For vehicle mobility simulation Eclipse SUMO v1.11.0 is used [19]. Table II lists the simulation parameters. Fig. 2 depicts a sample network topology in the simulations.

We have chosen an area of 1 square kilometer area in Istanbul to reflect an urban traffic and road infrastructure in our system. The selected region data is imported via the OSM Web Wizard, a utility application bundled with SUMO. On each run, we obtained location data step by step from the Traffic Control Interface (TraCI) of SUMO while tracking movements on the GUI interface. UAVs and the BS are inserted as the point of interest objects on the GUI and are updated on each step to visualize the topology better. Also, their ranges are highlighted with circles on the interface. Vehicle role and type are randomized according to the predefined distribution, which includes four types of vehicles with processor or task owner roles and one type for private vehicles that our network does not serve. Vehicle related simulation settings are listed in Table III.

In our experiments to evaluate our proposed framework, we have selected three different cases for task inter-arrival time to investigate the proposed system and the efficiency of our heuristic algorithms. For the evaluation of our algorithms, we added an Only-Cloud baseline option in which all task offloadings are done to the cloud. Each simulation run lasts for 15 minutes, including pre-population and cool-down periods. All the cases are run ten times, and the results are presented in the following section. On each run, the UAV connected to the BS is selected in round robin mode while the most centrally located UAV is selected as the one hosting SDN controller.

A. Scenario : Task Inter-arrival Time ($1/\lambda$)

The exponentially distributed expected time value between consecutive tasks is $1/\lambda$. Task Inter-arrival Time scenario is used to observe how the system reacts under varying loads (varying task generation intervals). The different values for the interval $1/\lambda$ are 5 sec. s (Case 1), 10 sec.s (Case 2), and 15 sec.s (Case 3).

B. Experimental Results

We present the numerical results of our experiments in this section and use them to investigate performance in our scenarios under three subcategories. First, we begin by comparing the objective function $\mathbb{D}$, the average of prioritization-based weighted penalty points, of each algorithm. Then, we consider the task failure and cloud offloading ratios along with controller pool time ($\bar{t}_{pool}$) to determine the algorithms' task processing characteristics. Lastly, we look at the averages of the task completion ($\bar{t}_{total}$), transmission ($\bar{t}_{tx}$), processor queue ($\bar{t}_{queue}$) and processing ($\bar{t}_{process}$) time values to understand the temporal results for task completion.

1) Impact of Task Inter-arrival Time: In our first scenario, the system was challenged with various network loads by changing the request interval with the same expected number of vehicles. Request intervals may seem small, but the test network uses 802.11g mode which has throughput up to 54 Mbps theoretically while it achieves much less due to overhead and path loss. In our emulation environment, the maximum achievable throughput was 36 Mbps and it is achievable for

 DOI: 10.1109/GIIS56506.2022.9936953

a vehicle-to-UAV link under 150 meters distance. However, many of the vehicles share the same interfaces which also affect their share of the bandwidth, i.e., transmission bitrates.

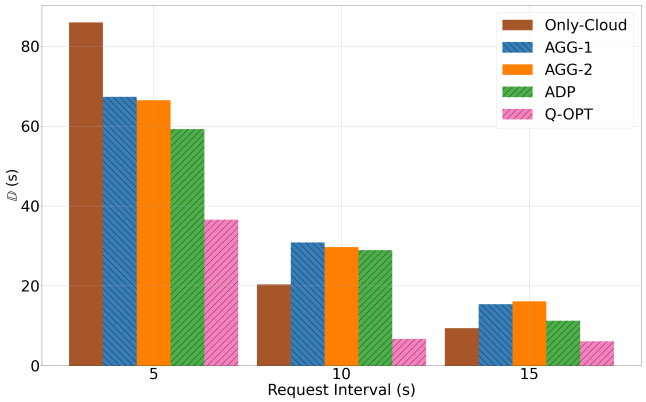

Fig. 3. Impact of task inter-arrival time on the objective function $\mathbb{D}$.

a) Penalty Results: These tests show how well the algorithms perform in terms of the main objective function, minimization of the average of prioritization-based weighted penalty points when network load changes. In Fig. 3, the impact of all three cases can be seen. Since the cloud side is stationary, a cloud connection is more stable than a connection between two moving vehicles. Also Only-Cloud has faster processing and no queue waiting time advantages. Thus, Only-Cloud option performs better under light load. On the other side, as the load increases, there is an exponential increase pattern of $\mathbb{D}$ for all the algorithms, but Only-Cloud option is the most impacted one because backhaul link to cloud becomes a bottleneck resulting slower packet transfers while our heuristic can distribute load to different processors.

The main logic of AGG-1 for preserving download starting order for task processing is to wait for tasks that are older or chosen earlier due to their priority-deadline values, it does not seem to be making a meaningful difference in terms of $\mathbb{D}$ compared to non-preserving version, AGG-2. Having oracle-type decisions and being theoretical output, Q-OPT seems to give superior results compared to the rest, especially as the load increases since it can make better decisions in advance while the rest makes with much less limited data. Among the online decision-makers, ADP outperforms the competition by a significant margin in all three cases.

b) Task Processing Characteristics: In the experimental results, eagerness of AGGs to fill their queues leads to having shorter assignment times, but it does not guarantee the overall processing will be quicker, too. It is observed that the opposite situation is true. Completing tasks one by one without getting the infrastructure congested provides better outcomes for ADP and Q-OPT than the AGGs; even the tasks wait longer in the controller queue. Another reason is that network load can be more easily managed in Q-OPT and ADP by delaying the decisions to get better assignment options from vehicles arriving in the area or from previously non-available vehicles.

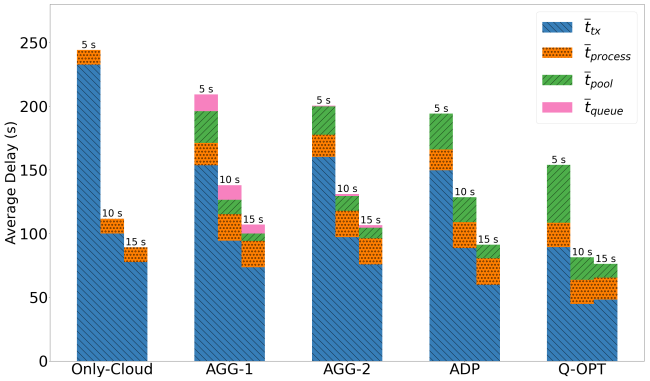

Fig. 4. Impact of task inter-arrival time on task delay composition.

c) Temporal Results for Task Completion: We can get insights about the system operation by looking at the life-cycle of tasks. In Fig. 4, stacked version of the component-based average packet delays are presented. The labels on the top of the bars refer to the inter-arrival time used for the given test group. The first point to make is that the majority of system delays are caused by transmission delays. Smarter network congestion management and routing protocols could be highly helpful for the performance of such a network. With the current parameters, there is no big impact of the processing time. Only on AGG-1, we see $5-10\%$ delay increase due to the queue wait procedure.

The average time the vehicle spends in the area is 320 seconds in the current scenario. By combining this data with the average system delay and task failure ratios, we may comment on the inter-arrival time required for heavy congestion levels. From this perspective, there is still room for higher loads which would make $\bar{t}_{total}$ over 300 seconds.

VII. Conclusion

In this paper, we elaborated on a critical use case for UAVs in post-disaster rescue scenarios with an emulation environment for a UAV-assisted software-defined edge network architecture. In that environment, we considered how task offloading can be flexibly used for efficient services and developed a task offloading scheme entailing different heuristics. A mathematical model has been proposed to make centralized offloading decisions for incoming task processing requests. For future work, the first direction is to extend our framework for smart routing. The biggest problem we have faced in our system is the limited capacity of the wireless links. Routing can help us to improve the potential of the system and mitigate this issue. It is also worth investigating how to develop smarter heuristic approaches for task offloading.

References

[1] O. Kalinagac, S. S. Kafiloglu, F. Alagoz, and G. Gur, "Caching and D2D sharing for content delivery in software-defined UAV networks," in *2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall)*, pp. 1–5, 2019.

[2] W. Shi, H. Zhou, J. Li, W. Xu, N. Zhang, and X. Shen, "Drone assisted vehicular networks: Architecture, challenges and opportunities," *IEEE Network*, vol. 32, pp. 130–137, 5 2018.

[3] X. Wang, L. Fu, Y. Zhang, X. Gan, and X. Wang, "VDNet: an infrastructure-less UAV-assisted sparse VANET system with vehicle location prediction," *Wireless Communications and Mobile Computing*, vol. 16, pp. 2991–3003, 12 2016.

[4] L. Zhao, K. Yang, Z. Tan, X. Li, S. Sharma, and Z. Liu, "A novel cost optimization strategy for SDN-enabled UAV-assisted vehicular computation offloading," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, pp. 3664–3674, 6 2021.

[5] H. Selvi, G. Gür, and F. Alagöz, "Cooperative load balancing for hierarchical SDN controllers," in *2016 IEEE 17th International Conference on High Performance Switching and Routing (HPSR)*, pp. 100–105, 2016.

[6] R. R. Fontes, S. Afzal, S. H. B. Brito, M. A. S. Santos, and C. E. Rothenberg, "Mininet-WiFi: Emulating software-defined wireless networks," in *Network and Service Management (CNSM), 2015 11th International Conference on*, pp. 384–389, Nov 2015.

[7] S. Jia and L. Zhang, "Modelling unmanned aerial vehicles base station in ground-to-air cooperative networks," *IET Communications*, vol. 11, pp. 1187–1194, 2017.

[8] W. Chen, Z. Su, Q. Xu, T. H. Luan, and R. Li, "VFC-based cooperative UAV computation task offloading for post-disaster rescue," in *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pp. 228–236, 2020.

[9] A. Alioua, H. eddine Djeghri, M. E. T. Cherif, S. M. Senouci, and H. Sedjelmaci, "UAVs for traffic monitoring: A sequential game-based computation offloading/sharing approach," *Computer Networks*, vol. 177, 2020.

[10] A. Al-Hourani, S. Kandeepan, and S. Lardner, "Optimal lap altitude for maximum coverage," *IEEE Wireless Communications Letters*, vol. 3, no. 6, pp. 569–572, 2014.

[11] S. Zhang, H. Zhang, B. Di, and L. Song, "Cellular UAV-To-X communications: Design and optimization for multi-UAV networks," *IEEE Trans. on Wireless Communications*, vol. 18, pp. 1346–1359, 2 2019.

[12] A. Al-Hourani, S. Kandeepan, and S. Lardner, "Optimal LAP altitude for maximum coverage," *IEEE Wireless Communications Letters*, vol. 3, no. 6, pp. 569–572, 2014.

[13] C. Liu, M. Ding, C. Ma, Q. Li, Z. Lin, and Y.-C. Liang, "Performance analysis for practical unmanned aerial vehicle networks with LoS/NLoS transmissions," *2018 IEEE International Conference on Communications Workshops, ICC Workshops 2018 - Proceedings*, pp. 1–6, 4 2018.

[14] O. Bouachir, A. Abrassart, F. Garcia, and N. Larrieu, "A mobility model for UAV ad hoc network," in *ICUAS 2014, International Conference on Unmanned Aircraft Systems*, (Orlando, United States), pp. pp 383–388, May 2014.

[15] W. Chen, D. Wang, and K. Li, "Multi-user multi-task computation offloading in green mobile edge cloud computing," *IEEE Transactions on Services Computing*, vol. 12, pp. 726–738, 9 2019.

[16] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, and G. Parulkar, "ONOS: Towards an open, distributed SDN OS," in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, (New York, NY, USA), p. 1–6, Association for Computing Machinery, 2014.

[17] B. Pfaff, J. Pettit, T. Koponen, E. J. Jackson, A. Zhou, J. Rajahalme, J. Gross, A. Wang, J. Stringer, P. Shelar, K. Amidon, and M. Casado, "The design and implementation of Open vSwitch," in *NSDI*, 2015.

[18] NLANR (National Laboratory for Applied Network Research)/DAST, "iPerf Tool." Available at https://iperf.fr/, 2019.

[19] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. Wießner, "Microscopic traffic simulation using SUMO," in *The 21st IEEE International Conference on Intelligent Transportation Systems*, IEEE, 2018.

 DOI: 10.1109/GIIS56506.2022.9936953