



Article Prioritization Based Task Offloading in UAV-Assisted Edge Networks [†]

Onur Kalinagac ^{1,2}, Gürkan Gür ^{2,*} and Fatih Alagöz ¹

- ¹ Department of Computer Engineering, Bogazici University, 34342 Istanbul, Turkey
- ² Institute of Applied Information Technology (InIT), Zurich University of Applied Sciences (ZHAW), 8401 Winterthur, Switzerland
- * Correspondence: gurkan.gur@zhaw.ch
- + This paper is an extended version of our paper published in the Global Information Infrastructure and Networking Symposium (GIIS) Argostoli, Greece, 26–28 September 2022, doi:10.1109/GIIS56506.2022.9936953.

Abstract: Under demanding operational conditions such as traffic surges, coverage issues, and low latency requirements, terrestrial networks may become inadequate to provide the expected service levels to users and applications. Moreover, when natural disasters or physical calamities occur, the existing network infrastructure may collapse, leading to formidable challenges for emergency communications in the area served. In order to provide wireless connectivity as well as facilitate a capacity boost under transient high service load situations, a substitute or auxiliary fast-deployable network is needed. Unmanned Aerial Vehicle (UAV) networks are well suited for such needs thanks to their high mobility and flexibility. In this work, we consider an edge network consisting of UAVs equipped with wireless access points. These software-defined network nodes serve a latency-sensitive workload of mobile users in an edge-to-cloud continuum setting. We investigate prioritization-based task offloading to support prioritized services in this on-demand aerial network. To serve this end, we construct an offloading management optimization model to minimize the overall penalty due to priority-weighted delay against task deadlines. Since the defined assignment problem is NP-hard, we also propose three heuristic algorithms as well as a branch and bound style quasi-optimal task offloading algorithm and investigate how the system performs under different operating conditions by conducting simulation-based experiments. Moreover, we made an open-source contribution to Mininet-WiFi to have independent Wi-Fi mediums, which were compulsory for simultaneous packet transfers on different Wi-Fi mediums.

Keywords: UAV networks; edge computing; task offloading; SDN; emergency computing; vehicular communications

1. Introduction

Future network infrastructure such as 6G networks is expected to provide much better QoS and user experience for a wide range of networked services anytime–anywhere in an edge-to-cloud continuum [1]. To this end, Unmanned Aerial Vehicles (UAVs) are instrumental to deploy a local network infrastructure dynamically or on an on-demand basis. These circumstances can occur during short-term demand surges for capacity, postdisaster communications, or dynamic data collection [2]. In that regard, UAV networks have three salient characteristics [3]. The first one is related to their location: since they are aerial systems, they enjoy a higher probability of line-of-sight (LoS) links for connecting ground nodes and other UAVs in comparison to terrestrial systems. Secondly, UAVs can be dynamically deployed in response to emerging capacity and connectivity requirements unlike the stationary ground infrastructure. Finally, a swarm of UAVs can construct scalable multi-UAV networks in a flexible configuration for pervasive and seamless services.



Citation: Kalinagac, O.; Gür, G.; Alagöz, F. Prioritization Based Task Offloading in UAV-Assisted Edge Networks. *Sensors* **2023**, *23*, 2375. https://doi.org/10.3390/s23052375

Academic Editors: Simone Genovesi, Manlio Bacco, Francesco Alessio Dicandia and Nelson Fonseca

Received: 19 December 2022 Revised: 12 February 2023 Accepted: 13 February 2023 Published: 21 February 2023



Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https:// creativecommons.org/licenses/by/ 4.0/). On the ground level, novel intelligent mobility solutions such as self-driving cars are already in operation, and vehicle communications are becoming more vital as user adoption increases. Applicable use cases in 5G and future networks may also be mission-critical, relying on connected services, such as emergency communications or cloud robotics for search and rescue. For these applications, UAVs can assist user devices and vehicles by providing cloud access, packet relaying or edge services for computation, storage and networking. In this work, we consider such a UAV integrated edge network and focus on the use case in which UAVs create an ad hoc network for connectivity including emergency communications. The investigated edge network is software-defined as envisaged in current 5G as well as future networks, which allows our system to cope with network management issues such as mobility-based frequent topology change and dynamic system states [4]. 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 and rescue team vehicles or related IoT devices.

For such use cases where we have time-sensitive and mission critical services, it is imperative to minimize delays via appropriate resource allocation while satisfying resource constraints. In this paper, we work on this problem and develop a task offloading scheme comprising different heuristics. We adopt a priority based approach for computation tasks in our edge UAV network with limited connectivity and resource availability. We also integrate a processing deadline into our scheme for each task as the key QoS criterion. Moreover, we investigate the performance of this system under different operating conditions via simulation-based experiments.

The contributions made in this paper can be listed as follows:

- Offloading management optimization model: We proposed a mathematical model to make centralized assignment/offloading decisions for the incoming task processing requests. Our model considers centralized assignment decisions for task offloading requests received from mobile ground user units. Offloading can be assigned to ground vehicles or a cloud processor service. The model's objective function is to minimize average penalty scores caused by task completion time;
- Open-source contribution to Mininet-WiFi: The wmediumd application, a Wi-Fi delay and loss simulator and used by a Mininet-WiFi emulator [5], did not support independent mediums simultaneously, which meant that all packets experience scheduling collisions with one another. It had prevented us from performing multiple data transfers at the same time without suffering a significant loss in throughput. We have completed the development of a source-code contribution for both Mininet-WiFi and wmediumd to isolate networks automatically with a custom configuration option;
- Heuristics and quasi-optimal algorithm for task offloading: Three heuristic algorithms and a branch and bound style quasi-optimal task offloading algorithm are implemented. The latter works on the previously simulated data to obtain better insight for the evaluation of our heuristic algorithms with respect to near-optimal operation.

The rest of the paper is organized as follows: In Section 2, we provide an overview of related studies in UAV networks, task offloading, and SDN. In Section 3, we present the system-related models. Next, we develop our problem formulation in Section 4. Then, we explain proposed task offloading algorithms in Section 5. In Section 6, we describe our simulation environment and its components. Later on, simulation results and performance metrics are presented in Section 7. Finally, we conclude the paper in Section 8.

2. Related Work

The related work is divided into the three main categories. The first category includes works on UAV applications in vehicular networks. The second category discusses task offloading algorithms and edge-cloud continuum perspective based on UAVs. Finally, SDN use cases for UAV and vehicular networks are reviewed in the final category.

2.1. UAV-Aided Vehicular Networks

Vehicles communicate with each other and with public networks through various communication systems, such as cellular networks and VANETs, to improve road efficiency and safety and access connected services [6]. UAVs mounted with communication equipment can improve the connectivity and efficiency of these networks in various ways [7]. UAVs can optimize the VANET routing process or they can be utilized as Store-Carry and Forward nodes to help ground vehicular networks improve connection and reliability in the presence of disconnected vehicles.

Similar to our study, Jia and Zhang [8] 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 of many UAVs, they investigate the smallest number of UAVs required to achieve a certain level of vehicle communication. In [3], 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 two studies focus on infrastructure level rather than possible applications such as task offloading on the constructed network.

Pourbaba et al. [9] proposed a sub-optimal method to increase coverage of full-duplex relay networks among ground vehicles by positioning UAVs. They have used a set of predefined UAV locations and have an estimated Signal-to-Interference-plus-Noise Ratio (SINR) between ground vehicles and UAVs. They have formulated an l_0 -norm non-combinatorial and NP-hard minimization problem and used an l_1 -norm approximation for it. With their approximation, a ten percent performance increase has been observed compared to a baseline scenario where UAVs have fixed locations.

In [10], data dissemination in UAV-aided VANETs is investigated by formulating a network throughput maximization problem to find delivery strategies and select the optimal paths for data delivery while considering the transmission rate of links and the delay constraint for data dissemination. In their analysis, the graph knapsack problem is reduced to the throughput maximization problem, and a polynomial-time approximation scheme is proposed to solve that graph knapsack problem. In another study on UAV-aided data dissemination, Zhang et al. [11] propose a protocol to reduce delay and improve system throughput by using UAVs as relayers with caching capabilities in VANETs. UAV trajectories are planned based on vehicle distributions and topology, which directly impacts the performance of UAV-assisted data dissemination. A centralized UAV trajectory algorithm (CTS-DP) is developed based on dynamic programming. Their results imply that CTS-DP outperforms the distributed trajectory scheduling algorithm (MHC) studied in their previous work. In the prior algorithm MHC, each UAV schedules its trajectory by maximizing vehicle coverage.

2.2. Task Offloading in UAV Networks

UAVs face challenges acting as edge computing nodes because of their computing power, weight, and battery limits. As a result, task offloading is one of the key approaches to facilitate the limited resources in the overall system as much as possible and was recently researched as part of the edge-cloud continuum paradigm.

Wang et al. [12] 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. They propose a stable matching algorithm 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 [13], a delay-optimal task offloading approach is proposed for a multi-tier edgecloud computing system in a multi-user environment. The main objective is the minimization of the total service time for UAVs applications. In this approach, UAV task execution is distributed across edge and cloud server computing nodes. Similarly, in [14], a dynamic heuristic algorithm is proposed to offload periodic tasks with the objectives of minimizing total time delay and energy consumption for the Software Defined Vehicular Network-supported services in the UAV-enabled MEC system.

Huang et al. [15] work on delay-sensitive task offloading to fog networks. They propose a heuristic particle swarm optimization algorithm based on a Lyapunov framework to complete tasks on time with the lowest energy consumption. In their stated scenario, computing units can be sped up to meet the time and stability constraints.

In a multi-UAV-assisted road traffic scenario, authors in [16] construct a three-player sequential game-based computational offloading methodology for processing UAV data. The computation delay, energy overhead, and communication and 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 [17], 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 the one we addressed in our work, but task offloading to other mobile ground vehicles and UAV mesh networks is not possible in the given study contrary to ours.

As an example of multistage optimization works, a UAV-assisted mobile offloading and the trajectory optimization problem is studied in [18]. The stated problem turns out to be non-convex, and the minimum user utility is maximized under the partial offloading schemes. Their three-stage optimization algorithm is based on successive convex approximation and nonlinear fractional programming. The power allocation, bandwidth allocation, offloading decision, and UAV trajectory are jointly optimized for the solution.

In [19], Kang et al. propose a hierarchical aerial computing system that would allow high altitude platforms (HAPs) and UAVs to work together to provide computing services for ground devices (GDs) with a variety of QoS requirements. In a hierarchical aerial computing system, they use the combined computing of UAVs and HAPs to meet the QoS requirements of GDs. They develop a partially observable Markov decision method to solve the very complex nonconvex optimization problem, which is constrained by multi-dimension resource management, energy constraints, and collision avoidance.

In [20], Sacco et al. propose a self-learning strategy that assists a UAV with deciding whether to delegate its tasks. The decisions are made based on the agent's projected behavior, which suggests whether or not edge cloud is advantageous to the incoming tasks. To make a prediction about future device load, they either use a model from the time-series class or a model from the class of ML regressors.

Zaman et al. [21] propose a task offloading framework with light user mobility prediction utilizing machine learning techniques. Based on past data, they predict the user's future location and may decide on the task offloading at an edge server which is close to the location instead of local task processing. They compare their task offloading algorithm with two other mobility based algorithms from the literature.

2.3. Software-Defined UAV Networks

SDN allows for network programmability by separating the control layer from dataforwarding network elements. SDN architecture can be used to reduce network configuration time and cost, which is critical for rapidly changing UAV networks [22]. It can also handle offloading requests and associated routing management based on network traffic [23].

Privacy and protection against external cyber threats are both possible by exploiting the capabilities of SDN architecture. In that regard, important application domains are analyzed from a security standpoint in [24]. The description and classification of performance assessment metrics in drone design are discussed for test-bed based approaches,

most widely used simulation platforms, and hybrid methods. In addition, potential vulnerabilities for drone-enabled SDN-based societies are presented.

An SDN routing framework for UAV networks is developed in [25]. A monitoring platform to obtain network statistics for SDN controllers and a load balancing algorithm based on that platform's analytical results are introduced. Routing decisions are controlled by considering the power limit of UAVs and the priority of flows.

For UAV-assisted infrastructure-less vehicular networks, Alioua et al. [26] propose a distributed SDN-based architecture. They also propose a road safety scenario as a use-case for their design. UAVs are used in this case to help emergency rescue vehicles in exploring and investigating inaccessible affected zones. Lastly, they formulate an offloading decision problem based on a two-player sequential game and prove the existence of Nash equilibrium.

Zhu et al. [27] consider an SDN-based cellular network with a UAV and BSs. UAV serves as a computational server in the system, calculating users' tasks, or as a relay node, forwarding users' jobs to MEC-enabled BSs, where they can be calculated in MEC servers. The objective function is formulated as minimization of the total weighted delay and energy consumption of the UAV and all users. They propose a joint mode selection and resource allocation optimization algorithm to solve this optimization problem. The proposed algorithm decomposes the optimization problem into two subproblems: task mode selection and resource allocation, which are performed in alternating iterations.

3. System Model

Our network model consists of N_U UAVs (U_i) and one ground base station (BS). UAVs are mounted with wireless Access Points (APs) having multiple Wi-Fi interfaces. One of the interfaces for each AP provides Wi-Fi service to terrestrial vehicles while the rest form mesh networks with nearby UAVs' APs. One of the UAVs is connected to the BS which has a backhaul link to the Internet and relevant cloud services. A computational unit that houses an SDN controller and serves as the network controller is also mounted on one of the UAVs. This controller is managed by a service provider remotely. UAVs are primarily used for communication, which reduces energy consumption, rather than to offload tasks. We do not take energy constraints into account for the sake of simplicity. The notations used in system modeling are listed in Table 1. The network model is shown in Figure 1.



Figure 1. Multi-UAV edge network model investigated in our work.

Symbol	Definition
U_i	The <i>ith</i> UAV
Vi	The <i>i</i> th vehicle
K _i	The <i>ith</i> task
O_{i,V_j}	If V_j is the owner of K_i , 1; otherwise, 0
α	The priority of K_i
D _i	The penalty point of K_i
w_i	The weighted penalty point of K_i
C_i	If K_i is completed, 1; otherwise, 0
p _{heavy}	The probability of the task offloaded to the cloud if the pool is heavily occupied
<i>p_{medium}</i>	The probability of the task offloaded to the cloud if the pool is moderately occupied
plight	The probability of the task offloaded to the cloud if the pool is lightly occupied
S_{K_i}	The size of K_i
P_i	The <i>i</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

Table 1. Notations in system modeling.

All the task offloading decisions are centrally made by a controller application, namely the offloading orchestrator. Two main types of terrestrial vehicles are present in the system. The first type corresponds to rescue and emergency vehicles, which are task owners and signal their task information including a deadline, priority factor, and size to the task pool. This pool is located at 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 this assignment. Only task identifier data are essentially stored in the task pool, not the task data itself. Therefore, the data transfers are carried out directly from the task owner to the processor units. In Figure 2, this basic task assignment flow is given. The processor vehicles have their own limited-size queues to store unprocessed tasks. A processor may simultaneously download the data of multiple tasks if the policy of the active task offloading algorithm allows that. The processing speed and queue size depend on the processor group (i.e., we assume 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 tasks are preserved for vehicle departures. In other words, a task is also sent back to the pool if the assigned processor vehicle leaves during packet transmission for that task. Depending on the deadline, two cases may occur for the situation in which a task owner leaves the coverage area: The system receives an extra penalty score z if the deadline is already missed; otherwise, it receives no penalty. To identify and monitor the network

location of vehicles, the orchestrator application listens for ARP packets. Additionally, access points notify the controller of vehicle departure or disconnection events at their associated stations periodically.



Figure 2. Task offloading sequence diagram.

The tasks created by the vehicles have an exponentially distributed inter-arrival time with rate λ . If the task pool is not empty when a new task arrives, it may be offload-ed/migrated to the cloud processing server with a probability based on the task pool's current occupancy at that time. Essentially, we consider minimizing average prioritization-based weighted penalty points for task offloading management as a QoS key performance indicator and thus propose algorithms to achieve that objective.

3.1. Channel Models

Since there is no consensus on a proper path loss model for UAV-enabled networks [28], we selected two popular path loss models [29,30] and applied them in our network simulations. In our simulation environment, the data rates of connections are probabilistic, and the probability values are obtained from the Received Signal Strength Indicator (RSSI) transmission mode matrix. The matrix values 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. The RSSI values for each connection are estimated based on the utilized channel models. The channel model equations used in our system are given in Appendix A.

3.2. Mobility Model

UAVs are deployed to particular locations to cover the area uniformly. 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 shape by moving around two fixed points at the same altitude as the mobility model in [31].

4. 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 prevents us from using a deadline as a constraint so we introduce a penalty scoring system in which an extra penalty score is added for unfilled tasks in order to encourage the completion of all tasks. The list of time-related notations that we utilize in the problem formulation is given in Table 2. In addition, we utilize indicator function 1_w that is assigned to one if w is true, zero, otherwise.

Symbol	Definition
Т	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
a_{P_i}	The arrival time slot of P_i
b_{P_i}	The departure time slot of P_i
a_{K_i}	The arrival time slot of K_i
CKi	The completion time slot of K_i

Table 2. Time-related notations in problem formulation.

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 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 when C_i equals 1:

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

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

$$N_p \qquad (2)$$

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

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

Equation (1) ensures that, in the best scenario, a task is processed only by a single processor. In Equation (2), the required time to complete K_i is calculated by summing all the related time components. The related components, processing time $t_{process,i}$ and data transmission time $t_{tx,i}$, are calculated in Equations (3) and (4).

 D_i is the penalty for K_i and calculated as:

$$D_{i} = \begin{cases} max(t_{total,i} - t_{deadline,i}, 0) & \text{if } C_{i} = 1 \\ 0 & \text{if } (C_{i} = 0) \land (O_{i,V_{j}} = 1) \land (b_{V_{j}} < a_{K_{i}} + t_{deadline,i}) \\ b_{V_{j}} - a_{K_{i}} - t_{deadline,i} + z \\ & \text{if } (C_{i} = 0) \land (O_{i,V_{j}} = 1) \land (b_{V_{j}} > a_{K_{i}} + t_{deadline,i}) \end{cases}$$
(5)

The weighted penalty point of K_i is calculated as:

$$w_i = \alpha_i \cdot D_i \qquad \forall i \tag{6}$$

where α_i is the task priority.

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:

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

subject to:

$$\sum_{i=1}^{N_p} x_{ij} + y_i \le 1 \qquad \forall i \tag{8}$$

$$x_{ij} \cdot c_{K_i} \le b_{P_i} \qquad \forall i \tag{9}$$

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

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

In Equation (7), we minimize the summation of the weighted penalty points of all tasks. Equation (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} on the left-hand side of Equation (9) is the sum of the task arrival time, a_{K_i} and $t_{total,i}$. Equation (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 the pool if the constraint happens to fail. Equations (10) and (11) represent the possible values of the decision variables.

Task offloading decisions significantly affect multiple delay results, especially when related vehicles share the same APs or links for connectivity since available data rates for those vehicles drop. Thus, each possible decision combination should be evaluated. This multi-user, multi-task problem is proved to be NP-hard [32], while the penalty scoring in our system model increases the complexity due to its nonlinear behavior upon task owner departures.

5. Task Offloading Algorithms in Our Model

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 times. The final algorithm is named the *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. The notations used in these algorithms are listed in Table 3.

SymbolDefinition w_i, P_j w_i if K_i is assigned to P_j $\Theta_{decision}$ The maximum number of candidate solutions that the algorithm will attempt to solve within a time-windowEREvent for system event receivedTPController task poolTATask to be assigned

Table 3. Notations in algorithms.

Penalty point list

Listn

5.1. 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. If the next assigned task is downloaded before the first one, the second task will be processed after the first is completed. The rationale is to complete the previously assigned task, probably being the one closer to its deadline, earlier. The AGG-1 algorithm is described in Algorithm 1, and it is run until the simulation time is up.

Algorithm 1: Aggressive-Wait Offloading Algorithm (AGG-1)
Wait for a system event <i>ER</i> ;
if ER is Event.TaskComplete then
Mark the completed K_i as completed and remove it from Q_j where $x_{ij} = 1$;
Increase P_j 's available S_{Q_j} by S_{K_i} ; if There is a task in Q_j and K_m , the one on the first line of the queue is already downloaded then Start to process K_m ; Add From Task Complete to clobel event resisters for the second start
Add Event. TaskComplete to global event registry for t _{process,m} later.
else if ER is Event.TxComplete then
Mark the transmitted K_i as waiting on Q_j ;
If K_i is the one on the first line of Q_j where $x_{ij} = 1$ then
Start to process K_i and mark it as processing; Add <i>Event TaskComplete</i> to global event registry for the start later
Add Loent. Tusk complete to global event registry for t _{process,i} fater.
else if ER is $Event.NewProcessor$ for P_j then
Add P_j to the available processors;
else if ER is Event. Processor Departure for P_j then
Remove P_j from available processors;
rion the tasks assigned to F_j , and the ones which are in the transmission
phase to <i>PP</i> ;
foreach K, helongs to the denarted V. do
if K: in TP then
Remove K_i from TP and mark K_i as failed;
else if K _i in transmission phase then
Remove K_i from the assigned Q_i and mark K_i as failed;
Increase available S_{Q_i} by S_{K_i} ;
else if EK is Event. New lask then
if It is not assigned to sloud then
Add it to TP:
while true do
Calculate w_{i,P_j} for each K_i from TP and the available P_j with remaining queue
size $\geq S_{K_i}$;
if There is no task–processor pair used for the calculation then
break;
X_{ij} = The task–processor pair with the lowest w_{i,P_j} ;
Assign K_i to P_j and mark K_i as being transmitted;
Remove K_i from TP ;
Decrease available S_{Q_j} by S_{K_i} ;
Go back to the first step.

5.2. 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 one downloaded earlier is processed first. This difference corresponds to the handling of *Event.TaskComplete* and *Event.TxComplete* events in Table 1, and the related processor does not check whether the downloaded task was the first one assigned before processing, contrary to AGG-1.

5.3. 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 to 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. Algorithm 2 shows the details of the algorithm.

Algorithm 2: Adaptive Offloading Algorithm (ADP)						
Wait for a system event <i>ER</i> ;						
The event handling part is identical to AGG-1.						
while true do						
Calculate w_{i,P_j} for each K_i from TP and the available P_j which is not assigned						
to any tasks that are marked as being transmitted;						
if There is no task–processor pair used for the calculation then						
break;						
X_{ij} = The task–processor pair with the lowest w_{i,P_i} ;						
Assign K_i to P_j and mark K_i as being transmitted;						
Remove K_i from TP ;						
Decrease available S_{Q_i} by S_{K_i} ;						
Go back to the first step.						

5.4. Quasi-Optimal Offloading Algorithm (Q-OPT)

The last approach is a branch and bound based algorithm that processes the whole simulation data to produce a sub-optimal yet close to optimal objective value for benchmarking our heuristic algorithms. It analyzes the data from past simulations and functions as an oracle-type decision-maker. This processed data includes task details (e.g., arrival time, size, deadline), vehicle positions, vehicle associated AP and RSSI values, and task processor details.

For each task, possible processor assignments are determined as lists. In order to narrow down the candidate solutions, arrival and departure times of vehicles and their connection status per step are considered to filter out the unfitting processors on each list. Two more decisions, skipping (meaning failing on purpose) and offloading to the cloud, are also added to these lists. Even the most underloaded 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. As illustrated in Figure 3, it 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.

Figure 3. Q-OPT algorithm-time window size detection on each step.

6. System Implementation

For our tests, we use an Ubuntu 21.04 operating system on a mobile workstation with an Intel Core i7-10875H (2.30 GHz \times 8), 16 GB RAM, and 256 GB SSD storage. The docker version of ONOS [33] v2.8 is utilized as an SDN controller. Mininet-WiFi [5] is used to create host devices, virtual stations, and APs. On Mininet-WiFi and its wireless medium simulation tool dependency wmediumd, some crucial improvements had to be made in order to support independent and simultaneous Wi-Fi packet transfers. They are merged into the main branch (https://github.com/intrig-unicamp/mininet-wifi/pull/419, accessed on 19 December 2022, https://github.com/ramonfontes/wmediumd/pull/9, accessed on 19 December 2022). Open vSwitchv2.15 [34] is used for virtual switch emulation. For traffic generation from vehicles in our experiments, iPerf v3.10 [35] is used. Eclipse SUMO v1.11.0 [36] is used for vehicle mobility simulation. We implemented our main simulation driver application in Python 3.9 and the offloading orchestrator in Java 11. The baseline simulation parameters are listed in Table 4.

Table 4. Baseline simulation parameters.	

Symbol	Value	Definition
λ	$0.1 \ { m s}^{-1}$	Exponentially distributed task generation rate
N _U	9	Number of UAVs
Task size	30–40 MB	Randomly distributed
\overline{V}	20 km/h	Average Vehicle Speed
β_{cloud}	3 MB/s	The processing speed of cloud
t _{deadline,i}	50–120 s	The time budget for K_i to be processed without any QoS violation (randomly distributed)
Pheavy	0.9	The probability of the task offloaded to the cloud if the pool is heavily occupied

Symbol	Value	Definition
<i>p_{medium}</i>	0.7	The probability of the task offloaded to the cloud if the pool is moderately occupied
plight	0.3	The probability of the task offloaded to the cloud if the pool is lightly occupied
Θ_{heavy}	10	The threshold value to classify the controller pool as heavily occupied
Θ_{medium}	7	The threshold value to classify the controller pool as moderately occupied
Θ_{light}	3	The threshold value to classify the controller pool as lightly occupied
$\Theta_{decision}$	10 M	The maximum number of candidate solutions that the algorithm will attempt to solve within a time- window
P_{tx,U_i}, P_{tx,V_i}	23 dBm	Transmission power of U_i , V_i
$G_{tx,U_i}, G_{tx,V_i}, G_{rx,U_j}, G_{rx,V_j}$	3 dBm	Antenna gains
σ^2	-91 dBm	Variance of White Gaussian Noise
f	2.4 GHz	The system carrier frequency
ρ	2	Air-to-air path loss exponent
η_{LoS}, η_{NLoS}	1 dBm, 20 dBm	The additional LoS,NLoS attenuation factors due to the LoS,NLoS connections
Z	60	The extra penalty point if the task deadline is missed

6.1. System Components

There are four main application modules for implementing and running simulations for our task offloading system in a UAV-assisted software-defined edge network environment. These are *Vehicle Mobility Generation, Network Infrastructure, Offloading Orchestrator,* and *Main Simulation Driver*. The overall system is depicted in Figure 4.

6.1.1. Vehicle Mobility Generation

We chose a one-square-kilometer area from Istanbul's Besiktas District (Figure 5) to reflect an urban traffic and road infrastructure in our system. The data from the selected region is imported by using the OSM Web Wizard, a utility application bundled with SUMO. On each run, we retrieved location data from the SUMO Traffic Control Interface (TraCI) iteratively while tracking the progress on the GUI interface. To visualize the topology better, UAVs and the BS are inserted as point-of-interest objects on the GUI interface and updated at each step. Additionally, circles are used to highlight APs' terrestrial ranges on the interface. In our simulations, we used five different vehicle classes: two of them with the task owner role, two of them with the task processor role, and one with no network role. The last one represents private vehicles that our network does not serve. Vehicle related simulation settings are listed in Table 5.



Figure 4. System components.



Figure 5. The visualization of topology elements in Besiktas network. The terrestrial coverage ranges of UAVs are indicated by the yellow circles around them.

Role	Class	Proportion	Shape	Color	Priority	β_i	S_{Q_i}
Owner	А	0.1	Emergency	White	0.7	-	-
Owner	В	0.2	Fire brigade	Red	0.3	-	-
Processor	А	0.06	Trailer	Purple	-	2 MBps	100 MB
Processor	В	0.1	Trailer	Cyan	-	1.5 MBps	70 MB
-	-	0.54	Passenger	Pink	-	-	-

Table 5. Vehicle parameters.

6.1.2. Network Infrastructure

The mac80211_hwsim software is a Linux kernel module, and it can virtualize Wi-Fi network cards. Mininet-WiFi, a fork of Mininet [37], initially loads this module with the desired number of virtual wireless interfaces necessary for our system. These interfaces can be configured just like the real ones, and Mininet-WiFi automates this process. mac80211_hwsim tracks the current channel of each card's radio and copies all transmitted frames to all the operating radio within the same channel. Mininet-WiFi also benefits from wmediumd, an application which allows simulating frame loss and delay by managing mac80211_hwsim from its netlink API. It has an event mechanism and adds artificial delays and failures to packet forwarding operations depending on the given channel model and radio locations. Dynamic configuration on these parameters can be performed via its socket API.

All Wi-Fi networks are contention-based and time-division duplexing systems [38]. In wmediumd, a queue mechanism is introduced to order packets to reflect delay caused by this concept. However, when multi-data transfers are performed simultaneously, it causes undesired low throughput levels even for the networks that are separated by distance or channel. There is a study on this issue that suggests a multi-threaded architecture from scratch [39]. To enable our experiments, we have developed a multi-medium mechanism on top of the existing repositories, making it possible to isolate queues of the nodes in different mediums. The new version of the application detects which nodes are talking to each other by sniffing AP-Station packet transmissions and grouping them. The code improvement also gives manual configuration ability to Mininet-WiFi end users to setup the system for advanced topologies. In our infrastructure, the average CPU usage by wmediumd during simulation was no more than 25% both before and after the multi-medium development, which indicates that our extension did not cause additional overhead and there was no CPU bottleneck in both cases.

A LoS probability-based channel model is also implemented on both Mininet-WiFi and wmediumd for AP-Station interfaces while keeping the existing log distance model for mesh interfaces. Thus, the applications are run in dual channel model mode. Unfortunately, a public contribution has not yet been made because it is currently a work in progress.

Nine APs representing UAVs and one AP representing the BS have been added to the network. UAVs are placed in a grid formation, and one of them is randomly selected for BS connection. A maximum of three separated mesh links per AP are allowed, and the links are selected by our routing algorithm, which tries to maximize bandwidth from BS to the rest of the APs after creating a minimum spanning tree with the highest link capacities. For AP-BS and AP-AP connections, Wi-Fi 802.11s mesh protocol is used. Different AP and mesh channels are assigned to the neighbor device interfaces. For the cloud server role, a Mininet host is linked to the BS AP. For the controller host role, another host is connected to the centrally positioned AP. A *localhost* machine is also linked to the system by using the Network Address Translation (NAT) feature of Mininet for debugging purposes. The iperf3.10 tool is used for its fresh 'zero copy' feature to lower the CPU burden caused by the app. For each data transmission, server and client apps are run on the related station and host interfaces.

6.1.3. Offloading Orchestrator

Two ONOS controller applications are developed, one for traffic monitoring and one for packet routing. The first one is used for testing purposes, which collects traffic data on device interfaces. On the other hand, the second one is one of the core developments required to facilitate the network. It works as an in-built reactive forwarding app, but there are crucial differences: When a new ARP packet is received, the in-built app floods the packet while our app installs all the flow rules required for end-to-end communication. Our application also installs ARP flow rules to lead ARP packets through the desired connection path, allowing mesh nodes to store host data. There are multiple separated mesh networks within the system. Mesh connections of APs do not pass data packets if they have not discovered the host beforehand in the current mesh protocol, IEEE 802.11s [40]. If a network location change is detected, the application invalidates all the previously installed flows related to that vehicle. A running sample of our topology is shown in Figure 6.



Figure 6. A sample network topology in the simulations.

6.1.4. Main Simulation Driver

A modular Python application handles the majority of the simulation logic. The simulation app performs its routines each second and then sleeps. Upon receiving vehicle locations from SUMO TraCI, it updates the location of the stations on Mininet-WiFi. It can send direct commands to a station and host to trigger task generation signaling events and V2V task data transfers. The controller is assumed to be mounted on the central UAV, so the application triggers Netcat client command from the station associated with the task owner vehicle.

7. Performance Evaluation

For the evaluation of the proposed system and the efficiency of our heuristic algorithms, we have selected two main scenarios and three different cases for each. As a baseline algorithm, we included the Only-Cloud option in which all tasks are offloaded to the cloud. Each simulation run lasts for 15 min, including pre-population and cool-down periods, and emulates the same amount of time. All the cases are run ten times for each case and algorithm combination. The results are presented in the following section.

7.1. Scenario 1—Task Inter-Arrival Time $(1/\lambda)$

The expected time value between two tasks is exponentially distributed and equals $1/\lambda$. This scenario shows how the system responds to different loads (varying task request intervals). The different values for the interval $1/\lambda$ are:

- *Case 1.1:* $1/\lambda = 5$ s;
- *Case* 1.2: $1/\lambda = 10$ s;
- *Case 1.3:* $1/\lambda = 15$ s.

7.2. Scenario 2—Average Vehicle Speed \overline{V}

Starting from the average walking speed of 5 km/h, we analyze the default case of 20 km/h and the relatively high-speed case. As the speed increases, vehicles become more likely to depart from the network coverage. Additionally, fast movement means more handover, resulting in more overhead:

- *Case 2.1:* $\overline{V} = 5 \text{ km/h}$;
- *Case 2.2:* $\overline{V} = 20 \text{ km/h}$;
- *Case 2.3:* $\overline{V} = 40 \text{ km/h}$.

7.3. Experimental Results

In this part, we give the numerical findings from our experiments and use them to analyze the performance under two different scenarios. First, we compare each algorithm's objective function \mathbb{D} , which represents the average of penalty points weighted by task priorities. Second, we determine their task processing characteristics by analyzing the task failure and cloud offloading ratios along with controller pool time (\bar{t}_{pool}). Lastly, we analyze their temporal results for task completion by considering 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.

7.3.1. Impact of Task Inter-Arrival Time

In Scenario 1, we applied three different task request inter-arrival times while maintaining the same expected number of vehicles. The APs deployed on the environment use 802.11 g mode, which has theoretical throughput of up to 54 Mbps but actually achieves much less due to overhead and path loss, so there was network congestion even at the *Case 1.3*, which has the highest task inter-arrival time. The maximum throughput in our simulation environment was 36 Mbps, which was achievable for a vehicle-to-UAV link under 150 m in length. Since many of the vehicles utilize the same interfaces, their share of the bandwidth becomes much less.

i Impact of Task Inter-Arrival Time on Penalty Results

Our first and main evaluation criterion is the performance analysis of the algorithms in terms of the main objective function \mathbb{D} , which aims to minimize the weighted average of the prioritization-based penalty points. In Figure 7, the impact of varying task request intervals on \mathbb{D} for each algorithm can be seen. A cloud connection is more stable than a connection between two moving vehicles because the cloud side is stationary. Cloud offloading also has the advantage of faster processing and no queue waiting time. As a result, the Only-Cloud option performs better compared to other heuristics under light load. On the other hand, as request interval decreases, all algorithms exhibit an exponential increase in \mathbb{D} . Among them, the Only-Cloud option is the most impacted one because the backhaul link to the cloud becomes a bottleneck, resulting in slower packet transfers, whereas our heuristics 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 the

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 make decisions with much less limited data. Among the online decision-makers, ADP outperforms the competition by a significant margin in all three cases.



Figure 7. Impact of task inter-arrival time on the objective function \mathbb{D} .

Along with \mathbb{D} , we can see the service delivery skills of the algorithms in Table 6. For Case 1.1, in which the system is in its most loaded state, users face a high ratio of task offloading failures, which is due to task owners leaving the area before their task is successfully transmitted at a high percentage. In addition, an increase in the cloud ratio means the number of tasks in the controller pools has been increased. ADP has the highest cloud offloading ratio, which is expected since the processors do not download multiple tasks simultaneously, leading the pool to be more crowded. Still, ADP has the best service scores compared to AGG-1 and AGG-2, considering \mathbb{D} and task failure ratios. After analyzing the experimental data records, we have discovered two main reasons. Firstly, the task download ratio per processor is always less than 1, which is not the case for the other two heuristic algorithms. Thus, it leads to a less congested network and current offloading tasks' transmission times to be shortened. Secondly, since the task pool gets filled more easily, utilization of the cloud commences earlier, while AGG algorithms start to offload to the cloud after all their queues are full.

Tabl	le 6.	Impact of	tas	k inter-	-arrival	l time o	n Ç	205	and	processing	characteristics.
------	-------	-----------	-----	----------	----------	----------	-----	-----	-----	------------	------------------

	Case ID	$1/\lambda$	\overline{V}	\mathbb{D}	Task Failure Ratio	Cloud Ratio	\overline{t}_{pool}
Only-Cloud	1.1	5	20	85.96	0.524	1	0
Only-Cloud	1.2	10	20	20.36	0.144	1	0
Only-Cloud	1.3	15	20	9.36	0.083	1	0
AGG-1	1.1	5	20	67.29	0.396	0.384	24.89
AGG-1	1.2	10	20	30.84	0.193	0.024	11.29
AGG-1	1.3	15	20	15.41	0.088	0.012	5.71
AGG-2	1.1	5	20	66.43	0.405	0.377	22.13
AGG-2	1.2	10	20	29.67	0.201	0.029	11.6
AGG-2	1.3	15	20	16.12	0.095	0.021	8.22
ADP	1.1	5	20	59.22	0.339	0.517	28.14
ADP	1.2	10	20	28.9	0.19	0.123	19.57
ADP	1.3	15	20	11.23	0.074	0.033	10.67
Q-OPT	1.1	5	20	36.56	0.237	0.227	45.46
Q-OPT	1.2	10	20	6.71	0.096	0.218	17.65
Q-OPT	1.3	15	20	6.1	0.093	0.388	10.72

ii Impact of Task Inter-Arrival Time on Task Processing Characteristics

From Table 6, we observe that AGGs achieve lower average pool time \bar{t}_{pool} resulting from their aggressiveness to fill processor queues. However, ADP and Q-OPT achieve better performance results in terms of the objective function since they complete tasks one by one without getting the infrastructure congested. Another reason is that Q-OPT and ADP allow for easier management of network load by delaying decisions in order to obtain better assignment options from vehicles that are arriving in the area or from vehicles that were not previously available.

iii Impact of Task Inter-Arrival Time on Temporal Results for Task Completion

In Figure 8, the component-based average packet delays are presented in a stacked form. The labels on the top of the bars indicate the task inter-arrival time used for the given test group. Numerical values are also represented in Table 7. By observing the task life cycle, we can obtain more insights about the system operation and the potential improvements. The first thing to notice is that transmission delays account for the majority of system delays. Increasing available bandwidth while improving network congestion management and routing protocols could be highly helpful for the performance of such a network. The processing time has little effect with the current parameters. Only on AGG-1 is a 5–10% delay increase observed due to the queue wait procedure.

	Case ID	$1/\lambda$	\overline{t}_{total}	\overline{t}_{tx}	t _{queue}	$\overline{t}_{process}$
Only-Cloud	2.1	5	47.31	0.199	1	0
Only-Cloud	2.2	20	63.66	0.282	1	0
Only-Cloud	2.3	40	71.04	0.358	1	0
AGG-1	2.1	5	209.27	153.71	13	17.46
AGG-1	2.2	10	138.02	94.43	12	20.8
AGG-1	2.3	15	107.07	73.59	7	20.76
AGG-2	2.1	5	200.54	160.24	1	17.45
AGG-2	2.2	10	131.07	97.19	2	20.77
AGG-2	2.3	15	106.64	75.82	2	20.56
ADP	2.1	5	194.17	149.76	0	16.27
ADP	2.2	10	128.57	88.57	0	20.4
ADP	2.3	15	91.34	59.75	0	20.87
Q-OPT	2.1	5	94.96	33.96	0	20.73
Q-OPT	2.2	10	73.05	35.7	0	20.35
Q-OPT	2.3	15	70.44	39.34	0	19.06

Table 7. Impact of task inter-arrival time on temporal results for task completion.

The vehicles stay in the area for an average of 320 s with the current settings. By combining this data with the average system delay and task failure ratios, we can state that it is still possible to have a lower task inter-arrival time than 5 s, which would increase \bar{t}_{total} to over 300 s, without fully congesting the network.

As a final analysis of this scenario, the system delay and task deadline of the tasks from simulations using the ADP algorithm are shown in a scatter graph in Figure 9a for Case 1.1, in Figure 9b for Case 1.2, and lastly in Figure 9c for Case 1.3. For each case, data from ten runs are reflected in the figures. It gives a good illustration of the strong effect of task generation rate on the system delay. In Case 1.1, high deviation around the average implies the system service instability at this level of load. Contrarily, in Case 1.3, our system runs at a steady state.



Figure 8. Impact of task inter-arrival time on task delay composition.



Figure 9. System delay and deadline distribution for ADP.

7.3.2. Impact of Average Vehicle Speed (V)

Vehicle mobility inherently has a significant effect on our framework since it changes the network topology. During handover, our offloading orchestrator reconfigures all userrelated flows, but Wi-Fi reconnection and flow installation cause observable disruptions in data transfer. In the baseline case from the previous scenario, vehicles were moving at 20 km/h so, for comparison, we had chosen human walking speed as 5 km/h for low mobility and 40 km/h for the double speed comparison. In order to deal with a moderately loaded network, we set the inter-arrival task interval $(1/\lambda)$ to 8 in this scenario.

i Impact of Average Vehicle Speed on Penalty Results

From Figure 10, we can see that mobility had a negative impact on performance. There are several main reasons according to the experiment data analysis. The first is that there has been more handover inference to data transfers, and the second is that offloading decision validities has lasted less; for example, task-assigned processors with good network connectivity have stayed in this mode for a much shorter duration. Only-Cloud baseline performance is the least affected by the vehicle speed changes due to its single-side stationarity. ADP seems to be the most efficient compared to the other heuristics and the Only-Cloud option.



Figure 10. Impact of average vehicle speed on the objective function \mathbb{D} .

ii Impact of Average Vehicle Speed on Task Processing Characteristics

In Table 8, we can see the service-related details. The task failure ratio for the Case 2.3 is high in general. Remembering that the task inter-arrival time is constant throughout the scenario, an increase in \bar{t}_{pool} and cloud ratio indicates that the offloading orchestrator has issues serving incoming requests.

iii Impact of Average Vehicle Speed on Temporal Results for Task Completion

The delay distributions are illustrated in Figure 11. As the vehicles become faster, handover overhead occurs due to the disconnections during network changes, increasing the transmission times. For all methods, the existing increase in failure ratio indicates that the offloading orchestrator has issues serving incoming requests as the speed increases.

	Case #	$1/\lambda$	\overline{V}	\mathbb{D}	Task Failure Ratio	Cloud Ratio	\overline{t}_{pool}
Only-Cloud	2.1	8	5	47.31	0.199	1	0
Only-Cloud	2.2	8	20	63.66	0.282	1	0
Only-Cloud	2.3	8	40	71.04	0.358	1	0
AGG-1	2.1	8	5	36.63	0.153	0.042	11.54
AGG-1	2.2	8	20	49.35	0.211	0.01	12.45
AGG-1	2.3	8	40	72.29	0.42	0.117	21.02
AGG-2	2.1	8	5	33.22	0.146	0.03	13.11
AGG-2	2.2	8	20	47.03	0.212	0.015	11.54
AGG-2	2.3	8	40	73.14	0.478	0.121	17.67
ADP	2.1	8	5	28.71	0.137	0.107	17.31
ADP	2.2	8	20	41.35	0.197	0.086	17.64
ADP	2.3	8	40	66.15	0.399	0.256	29.53
Q-OPT	2.1	8	5	5.51	0.055	0.146	26.27
Q-OPT	2.2	8	20	10.08	0.033	0.133	24.11
Q-OPT	2.3	8	40	30.69	0.143	0.188	35.27

Table 8. Impact of average vehicle speed on QoS and processing characteristics.



Figure 11. Impact of average vehicle speed on task delay distribution.

7.4. Algorithm Analysis

Our algorithms compare all possible task–processor pairings, which makes them exponential. Heuristic ones consider only the currently available processors, so the solution pool is small and the problem can be solved in real-time. In Q-OPT, we limit the size of the solution pool by adjusting the considered time interval on each step in order to solve it in a feasible amount of time.

7.5. Discussion

To summarize, we were able to describe the core fundamentals of our proposed UAV-assisted Software-Defined Edge Network and task offloading heuristics through a step-by-step examination of the test scenarios. These scenarios focused on two main aspects: task request inter-arrival time and the average speed of the vehicles.

The first scenario helped us to investigate the capacity of the designed network under three different traffic requirements. Unfortunately, many of the vehicles share the same interfaces, which impacts their available transmission bit rates and leads to low capacity levels. From the second scenario, we observed that vehicle mobility has a big impact on our system since it dynamically modifies the network topology. During the handover, all user-related flows should be modified. Our offloading orchestrator handles this job, but reconnection and topology discovery-related delay overheads occur during topology changes.

The Only-Cloud is sufficient when the request rate is low. However, we clearly observe that MEC-based solutions are necessary in high-load cases due to the bottleneck on the back-haul connection. After investigating the performance of our algorithms, it can be said that ADP outperformed the other heuristic algorithms. This result is related to the network's core problem: it requires effective congestion control methods to handle high levels of traffic load. If we compare AGG-1 and AGG-2 with each other, they have similar objective function results, but the average delay is lower in the more aggressive case, AGG-2, due to the extra queue time in AGG-1.

Q-OPT appears to outperform the others, especially as the service request rate increases because it can make better decisions in advance while the others make decisions with much fewer data in real-time. Q-OPT highly utilizes cloud server processing from the beginning and thus slows down when the network is congested. In addition, it fails to complete the processing of some tasks on purpose. It does not serve the task owners with their departure time earlier than their task's deadline even when the system is idle and the task can be handled. In the high-load cases, the algorithm also skips some tasks for the sake of a better overall score.

8. Conclusions

From the development side, several paper related artifacts have been developed in this work. Firstly, a core application orchestrating all the components of emulation has been created. An orchestrator application for ONOS has been developed for the reactive management of flows and hosts. Open-source contributions to Mininet-WiFi and wmediumd have been made on a crucial feature that allows users to operate in isolated Wi-Fi mediums. Three heuristic algorithms and a branch-and-bound style quasi-optimal task offloading algorithm were developed, and their performance analysis along with the Only-cloud baseline case has been conducted in terms of request rate and mobility.

When the demand in the system is low, the Only-Cloud approach is generally adequate. On the other hand, we note that MEC-based solutions are especially required in highload scenarios because of typical back-haul connection bottleneck and latency increase. Since it can make better decisions in advance while the others must make decisions with considerably less information in real-time, Q-OPT outperforms the others in terms of the objective function, particularly as the number of service requests rises. Among the heuristic algorithms, ADP achieves better performance results since the processors in ADP handle tasks one at a time without overloading the infrastructure.

The first direction for future work is to expand our framework for smart routing. The most serious issue we have encountered in our system tests is the limited capacity of wireless links. Smarter network congestion management and routing algorithms can help us to improve the capacity of the system and mitigate this issue. It could also be worth developing smarter heuristic approaches for task offloading. Lastly, better management of UAV mobility could be used to handle area-based congestion. We have observed that some of the APs are more likely to be active during our simulations. According to our analysis, this is because of the traffic model which leads vehicles, especially to the central areas in the simulated geographical area.

Author Contributions: Conceptualization, O.K., G.G. and F.A.; Methodology, O.K., G.G. and F.A.; Software, O.K.; Writing—original draft, O.K.; Writing—review & editing, G.G. and F.A.; Visualization, O.K., G.G.; Supervision, G.G. and F.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: https://github.com/onurklngc/drone-network-onos-and-mininet/tree/master.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Channel Models

The notations used in channel modeling are listed in Table A1.

Table A1. Notations used in Channel Modeling.

Symbol	Definition
η_{LoS}	The additional LoS attenuation factors due to the LoS connections
η_{NLoS}	The additional NLoS attenuation factors due to the NLoS connections
С	The speed of light traveling through a vacuum
f	The system carrier frequency
$d_{U_i,U_i}(t)$	The distance between U_i and U_j at t
$d_{U_i,V_j}(t)$	The distance between U_i and V_j at t
m,n	A2G channel parameters
$p_{LoS,U_i,V_i}(t)$	The probability of LoS connection between U_i and V_i at t
$p_{NLoS,U_i,V_i}(t)$	The probability of NLoS connection between U_i and V_i at t
G_{tx,U_i}	The transmission gain of U_i
G_{rx,U_i}	The receiver gain of U _j
G_{rx,V_i}	The receiver gain of V_j
$P_{U_i,U_i}(t)$	The average received power of U_j from U_i at t
$P_{U_i,V_i}(t)$	The average received power of V_j from U_i at t
$L_{K_i,P_j}(t)$	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
$L_{A,U_i}(t)$	The link capacity between $A \in \{U_i, V_i\}$ and U_j at time slot t
$\theta_i(t)$	The elevation angle at <i>t</i>
ρ	A2A path loss exponent

Appendix A.1. Air-to-Ground (A2G) Communication Channel Model

The utilized air-to-ground propagation model is based on the probability of line of sight and is used for UAV-to-terrestrial-vehicle communication.

In time slot *t*, the path losses for LoS and NLoS from i^{th} UAV U_i to j^{th} vehicle V_j are given by:

$$PL_{LoS,U_i,V_j}(t) = PL_{FS,U_i,V_j}(t) + \eta_{LoS}$$
(A1)

$$PL_{NLoS,U_i,V_i}(t) = PL_{FS,U_i,V_i}(t) + \eta_{NLoS}$$
(A2)

where $PL_{FS,U_i,V_i}(t)$ is free space path loss, which is calculated as:

$$PL_{FS,U_i,V_j}(t) = 20log(4\pi d_{U_i,V_j}(t)f/c)$$
 (A3)

In time slot *t*, a connection between i^{th} UAV U_i and j^{th} vehicle is LoS with a probability, $p_{LoS,U_i,V_i}(t)$ or NLos with a probability, $p_{NLoS,U_i,V_i}(t)$. They are calculated as:

$$p_{LoS,U_i,V_j}(t) = \frac{1}{1 + m \times exp(-n(\theta_i(t) - m))}$$
(A4)

$$p_{NLoS,U_i,V_i}(t) = 1 - p_{LoS,U_i,V_i}(t)$$
 (A5)

Then, the average received power of U_i from V_j is

$$P_{U_i,V_j}(t) = P_{tx,U_i} + G_{tx,U_i} + G_{rx,V_j} - \bar{PL}_{U_i,V_j}(t)$$
(A6)

where the average path loss is

$$\bar{PL}_{U_i,V_j}(t) = p_{LoS,U_i,V_j}(t)PL_{LoS,U_i,V_j}(t) + p_{NLoS,U_i,V_j}(t)PL_{NLoS,U_i,V_j}(t)$$
(A7)

Appendix A.2. Air-to-Air (A2A) Communication Channel Model

The free-space channel model is used for UAV-to-UAV communication. When U_i transmits signals to U_i , the received power at U_i is expressed as

$$P_{U_i,U_i}(t) = P_{tx,U_i} + G_{tx,U_i} + G_{rx,U_i} - PL_{FS,U_iU_i}(t)$$
(A8)

where

$$PL_{FS,U_{i}U_{j}}(t) = 10\log(4\pi d_{U_{i},U_{j}}(t)f/c)^{\rho}$$
(A9)

References

- 1. Porambage, P.; Gür, G.; Osorio, D.P.M.; Liyanage, M.; Gurtov, A.; Ylianttila, M. The Roadmap to 6G Security and Privacy. *IEEE Open J. Commun. Soc.* 2021, 2, 1094–1122. [CrossRef]
- Kalinagac, O.; Kafiloglu, S.S.; Alagoz, F.; Gur, G. Caching and D2D Sharing for Content Delivery in Software-Defined UAV Networks. In Proceedings of the 2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall), Honolulu, HI, USA, 22–25 September 2019; pp. 1–5. [CrossRef]
- 3. Shi, W.; Zhou, H.; Li, J.; Xu, W.; Zhang, N.; Shen, X. Drone Assisted Vehicular Networks: Architecture, Challenges and Opportunities. *IEEE Netw.* 2018, *32*, 130–137. [CrossRef]
- Selvi, H.; Gür, G.; Alagöz, F. Cooperative load balancing for hierarchical SDN controllers. In Proceedings of the 2016 IEEE 17th International Conference on High Performance Switching and Routing (HPSR), Yokohama, Japan, 14–17 June 2016; pp. 100–105. [CrossRef]
- Fontes, R.R.; Afzal, S.; Brito, S.H.B.; Santos, M.A.S.; Rothenberg, C.E. Mininet-WiFi: Emulating software-defined wireless networks. In Proceedings of the 2015 11th International Conference on Network and Service Management (CNSM), Barcelona, Spain, 9–13 November 2015; pp. 384–389. [CrossRef]
- 6. Zheng, K.; Zheng, Q.; Chatzimisios, P.; Xiang, W.; Zhou, Y. Heterogeneous Vehicular Networking: A Survey on Architecture, Challenges, and Solutions. *IEEE Commun. Surv. Tutor.* **2015**, *17*, 2377–2396. [CrossRef]
- Alzahrani, B.; Oubbati, O.S.; Barnawi, A.; Atiquzzaman, M.; Alghazzawi, D. UAV assistance paradigm: State-of-the-art in applications and challenges. J. Netw. Comput. Appl. 2020, 166, 102706. [CrossRef]
- Jia, S.; Zhang, L. Modelling unmanned aerial vehicles base station in ground-to-air cooperative networks. *IET Commun.* 2017, 11, 1187–1194. [CrossRef]
- Pourbaba, P.; Manosha, K.B.; Ali, S.; Rajatheva, N. Full-duplex UAV relay positioning for vehicular communications with underlay v2v links. In Proceedings of the 2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring), Kuala Lumpur, Malaysia, 28 April–1 May 2019. [CrossRef]
- Fan, X.; Huang, C.; Fu, B.; Wen, S.; Chen, X. UAV-Assisted Data Dissemination in Delay-Constrained VANETs. *Mob. Inf. Syst.* 2018, 2018, 8548301. [CrossRef]
- Zhang, R.; Zeng, F.; Cheng, X.; Yang, L. UAV-Aided Data Dissemination Protocol with Dynamic Trajectory Scheduling in VANETs. In Proceedings of the 2019 IEEE International Conference on Communications (ICC), Shanghai, China, 20–24 May 2019; pp. 1–6. [CrossRef]
- 12. Wang, Y.; Chen, W.; Luan, T.H.; Su, Z.; Xu, Q.; Li, R.; Chen, N. Task Offloading for Post-Disaster Rescue in Unmanned Aerial Vehicles Networks. *IEEE/ACM Trans. Netw.* **2022**, *30*, 1525–1539. [CrossRef]
- 13. Almutairi, J.; Aldossary, M.; Alharbi, H.A.; Yosuf, B.A.; Elmirghani, J.M.H. Delay-Optimal Task Offloading for UAV-Enabled Edge-Cloud Computing Systems. *IEEE Access* 2022, *10*, 51575–51586. [CrossRef]
- Wang, J.; Feng, D.; Zhu, J.; Huang, H. Vehicular Computation Offloading in UAV-enabled MEC Systems. In Proceedings of the 2022 IEEE 25th International Conference on Computer Supported Cooperative Work in Design (CSCWD), Hangzhou, China, 4–6 May 2022; pp. 1071–1076. [CrossRef]
- 15. Huang, C.; Wang, H.; Zeng, L.; Li, T. Resource Scheduling and Energy Consumption Optimization Based on Lyapunov Optimization in Fog Computing. *Sensors* **2022**, *22*, 3527. [CrossRef]
- 16. Alioua, A.; Djeghri, H.; Cherif, M.E.T.; Senouci, S.M.; Sedjelmaci, H. UAVs for traffic monitoring: A sequential game-based computation offloading/sharing approach. *Comput. Netw.* **2020**, *177*, 107273. [CrossRef]
- 17. Zhao, L.; Yang, K.; Tan, Z.; Li, X.; Sharma, S.; Liu, Z. A Novel Cost Optimization Strategy for SDN-Enabled UAV-Assisted Vehicular Computation Offloading. *IEEE Trans. Intell. Transp. Syst.* **2021**, *22*, 3664–3674. [CrossRef]
- Feng, G.; Wang, C.; Li, B.; Lv, H.; Zhuang, X.; Lv, H.; Wang, H.; Hu, X. UAV-assisted wireless relay networks for mobile offloading and trajectory optimization. *Peer -Peer Netw. Appl.* 2019, *12*, 1820–1834. [CrossRef]

- 19. Kang, H.; Chang, X.; Mišić, J.; Mišić, V.B.; Fan, J.; Liu, Y. Cooperative UAV Resource Allocation and Task Offloading in Hierarchical Aerial Computing Systems: A MAPPO Based Approach. *IEEE Internet Things J.* **2023**, 1. [CrossRef]
- Sacco, A.; Esposito, F.; Marchetto, G.; Montuschi, P. A Self-Learning Strategy for Task Offloading in UAV Networks. *IEEE Trans. Veh. Technol.* 2022, 71, 4301–4311. [CrossRef]
- Zaman, S.K.; Jehangiri, A.; Maqsood, T.; Shuja, J.; Ahmad, Z.; Umar, A. LiMPO: Lightweight mobility prediction and offloading framework using machine learning for mobile edge computing. *Clust. Comput.* 2022. [CrossRef]
- Secinti, G.; Darian, P.B.; Canberk, B.; Chowdhury, K.R. SDNs in the Sky: Robust End-to-End Connectivity for Aerial Vehicular Networks. *IEEE Commun. Mag.* 2018, 56, 16–21. [CrossRef]
- Misra, S.; Saha, N. Detour: Dynamic Task Offloading in Software-Defined Fog for IoT Applications. *IEEE J. Sel. Areas Commun.* 2019, 37, 1159–1166. [CrossRef]
- 24. Al-Turjman, F.; Abujubbeh, M.; Malekloo, A.; Mostarda, L. UAVs assessment in software-defined IoT networks: An overview. *Comput. Commun.* **2020**, *150*, 519–536. [CrossRef]
- Zhang, X.; Wang, H.; Zhao, H. An SDN framework for UAV backbone network towards knowledge centric networking. In Proceedings of the IEEE INFOCOM 2018—IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Honolulu, HI, USA, 15–19 April 2018; pp. 456–461. [CrossRef]
- Alioua, A.; Senouci, S.M.; Moussaoui, S.; Sedjelmaci, H.; Messous, M.A. Efficient Data Processing in Software-Defined UAV-Assisted Vehicular Networks: A Sequential Game Approach. Wirel. Pers. Commun. 2018, 101, 2255–2286. [CrossRef]
- Zhu, Y.; Wang, S.; Liu, X.; Tong, H.; Yin, C. Joint Task and Resource Allocation in SDN-based UAV-assisted Cellular Networks. In Proceedings of the 2020 IEEE/CIC International Conference on Communications in China, ICCC 2020, Chongqing, China, 9–11 August 2020; pp. 430–435. [CrossRef]
- Liu, C.; Ding, M.; Ma, C.; Li, Q.; Lin, Z.; Liang, Y.C. Performance Analysis for Practical Unmanned Aerial Vehicle Networks with LoS/NLoS Transmissions. In Proceedings of the 2018 IEEE International Conference on Communications Workshops, ICC Workshops 2018—Proceedings, Kansas City, MO, USA, 20–24 May 2018; pp. 1–6. [CrossRef]
- 29. Al-Hourani, A.; Kandeepan, S.; Lardner, S. Optimal LAP Altitude for Maximum Coverage. *IEEE Wirel. Commun. Lett.* 2014, *3*, 569–572. [CrossRef]
- Zhang, S.; Zhang, H.; Di, B.; Song, L. Cellular UAV-To-X Communications: Design and Optimization for Multi-UAV Networks. IEEE Trans. Wirel. Commun. 2019, 18, 1346–1359. [CrossRef]
- Bouachir, O.; Abrassart, A.; Garcia, F.; Larrieu, N. A Mobility Model For UAV Ad hoc Network. In Proceedings of the ICUAS 2014, International Conference on Unmanned Aircraft Systems, Orlando, FL, USA, 27–30 May 2014; pp. 383–388.
- 32. Chen, W.; Wang, D.; Li, K. Multi-User Multi-Task Computation Offloading in Green Mobile Edge Cloud Computing. *IEEE Trans.* Serv. Comput. 2019, 12, 726–738. [CrossRef]
- 33. Berde, P.; Gerola, M.; Hart, J.; Higuchi, Y.; Kobayashi, M.; Koide, T.; Lantz, B.; O'Connor, B.; Radoslavov, P.; Snow, W.; et al. ONOS: Towards an Open, Distributed SDN OS. In Proceedings of the Third Workshop on Hot Topics in Software Defined Networking, Chicago, IL, USA, 22 August 2014; Association for Computing Machinery: New York, NY, USA, 2014; pp. 1–6. [CrossRef]
- Pfaff, B.; Pettit, J.; Koponen, T.; Jackson, E.J.; Zhou, A.; Rajahalme, J.; Gross, J.; Wang, A.; Stringer, J.; Shelar, P.; et al. The Design and Implementation of Open vSwitch. In Proceedings of the NSDI, Oakland, CA, USA, 4–6 May 2015.
- 35. NLANR (National Laboratory for Applied Network Research)/DAST. iPerf Tool. 2019. Available online: https://iperf.fr/ (accessed on 19 December 2022).
- Lopez, P.A.; Behrisch, M.; Bieker-Walz, L.; Erdmann, J.; Flötteröd, Y.P.; Hilbrich, R.; Lücken, L.; Rummel, J.; Wagner, P.; Wießner, E. Microscopic Traffic Simulation using SUMO. In Proceedings of the 21st IEEE International Conference on Intelligent Transportation Systems, Maui, HI, USA, 4–7 November 2018.
- Lantz, B.; Heller, B.; McKeown, N. A Network in a Laptop: Rapid Prototyping for Software-Defined Networks. In Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks, Monterey, CA, USA, 20–21 October 2010; Association for Computing Machinery: New York, NY, USA, 2010. [CrossRef]
- Consortium, I.E. Broadband Wireless and WiMAX; Comprehensive Report; International Engineering Consortium: Chicago, IL, USA, 2005.
- Moreira, M.J.M. Yawmd: Multiple Medium Support and Performance Improvements for Wmediumd. Master's Thesis, University of Porto, Porto, Portugal, 2020.
- Carrano, R.C.; Magalhães, C.S.; Saade, D.C.M.; Albuquerque, C.V.N. IEEE 802.11s Multihop MAC: A Tutorial. *IEEE Commun. Surv. Tutor.* 2011, 13, 52–67. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.