PRIORITIZATION DRIVEN TASK OFFLOADING IN UAV-AIDED SOFTWARE-DEFINED NETWORKS

by

Onur Kalınağaç

B.S., Electrical & Electronics Engineering, Boğaziçi University, 2015

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering

Boğaziçi University

2022

ACKNOWLEDGEMENTS

I cannot begin to express my thanks to my instructor and thesis supervisor, Prof. Fatih Alagöz, who gave me the necessary background and as many opportunities as one could from the beginning. I have been able to finish my M.S. thesis thanks to his supervision and profound belief in my work. I shall be forever obliged for his invaluable insights.

I would also like to extend my deepest gratitude to Dr. Gürkan Gür for unwavering guidance during my M.S. studies. He has spent his precious time mentoring me with constructive advice and inspired me to pursue scientific research. Therefore, I have been able to complete this research.

Special thanks go to fellow researcher Dr. Sinem Kafiloglu for co-authoring my first scientific paper and sharing her in-depth knowledge.

Many thanks to my SATLAB colleagues for their dedicated contribution and feedback. They continuously provided insightful suggestions and were always enthusiastic about mentoring and helping with difficulties in this process.

I am deeply indebted to my family for their everlasting support and love. This thesis would not have been materialized without their relentless support and encouragement even in the most difficult times.

ABSTRACT

PRIORITIZATION DRIVEN TASK OFFLOADING IN UAV-AIDED SOFTWARE-DEFINED NETWORKS

Under demanding operational conditions such as traffic surges, coverage issues, very-low latency requirements, or security concerns, terrestrial cellular 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 the first response and emergency communications in the served area. In order to provide low-latency services for first-responders and baseline connectivity for the public 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. This thesis considers a software-defined network consisting of Unmanned Aerial Vehicles (UAVs) mounted with Wi-Fi access points, which serve mobile users with the 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. In addition to baseline connectivity for users in the deployed area, this computing-communication infrastructure essentially provides Vehicle-to-Vehicle (V2V) and Vehicle-to-Cloud (V2C) task offloading services to terrestrial units. In our work, task processing, and offloading management, task deadlines are taken into account as the key criterion for meeting service quality requirements. Accordingly, an offloading management optimization model is defined to minimize the overall penalty due to delay weighted with task priorities. 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.

ÖZET

İHA DESTEKLİ YAZILIM TANIMLI AĞLARDA ÖNCELİK ODAKLI GÖREV DAĞITIMI

Trafik dalgalanmaları, kapsama sorunları, çok düşük gecikme gereksinimleri veya güvenlik endişeleri gibi zorlu operasyonel koşullar altında, karasal hücresel ağlar, kullanıcılara ve uygulamalara beklenen hizmet düzeylerini sağlamada yetersiz kalabilir. Ayrıca, doğal afetler veya fiziksel felaketler sırasında, mevcut ağ altyapısı çökebilir ve hizmet verilen alanda ilk müdahale ve acil durum iletişimi için ciddi zorluğa sebep olabilir. Saha müdahale ekipleri için düşük gecikmeli hizmetler ve herkese açık temel internet bağlantısı sağlamak ve ayrıca geçici yüksek hizmet yükü durumlarında kapasite artırmayı kolaylaştırmak için yedek veya yardımcı olma görevinde, hızlıca kurulabilir bir ağ gereklidir. İnsansız Hava Aracı (İHA) ağları, hızlı ve esnek hareket kabiliyetleri sayesinde bu tür ihtiyaçlar için oldukça uygundur. Bu tez, uçtan buluta süreklilik ortamında gecikmeye duyarlı iş yüküne sahip olan mobil kullanıcılara hizmet veren, Wi-Fi erişim noktası monte edilmiş İHAlardan oluşan yazılım tanımlı bir ağı ele almaktadır. İsteğe bağlı hava ağı aracılığıyla öncelikli hizmetler sağlamak için görev atama paradigmasını araştırır. Hizmet verilen alandaki kullanıcılar için temel bağlantıya ek olarak, bu bilgi işlem-iletişim altyapısı esas olarak karasal birimlere araçtan araca (V2V) ve araçtan buluta (V2C) görev atama hizmetleri sağlar. Çalışmamızda, görev işleme ve görev atama yönetimi, hizmet kalitesi gereksinimlerinin karşılanmasının değerlendirilmesinde ana kriter olarak görev süreleri dikkate alınmaktadır. Buna göre, görev önceliğine göre ağırlıklandırılmış gecikme sürelerinin toplamını en aza indirmek için bir atama yönetimi optimizasyon modeli tanımlanır. Tanımlanan atama problemi NP-hard olduğundan, sistemin farklı çalışma koşullarında nasıl performans gösterdiğini incelemek için uyarlanmış buluşsal modeller önerilmiş ve değerlendirilmiştir.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS . . . iii
ABSTRACT . . . iv
ÖZET . . . v
LIST OF FIGURES . . . viii
LIST OF TABLES . . . x
LIST OF SYMBOLS . . . xii
LIST OF ACRONYMS/ABBREVIATIONS . . . xv
1. INTRODUCTION . . . 1
1.1. Key Contributions . . . 2
1.2. Thesis Outline . . . 3
2. RELATED WORK . . . 5
2.1. UAV-Aided Vehicular Networks . . . 5
2.2. Software-Defined UAV Networks . . . 6
2.3. Task Offloading in UAV Networks . . . 7
3. PROPOSED MODEL: TASK OFFLOADING IN UAV-AIDED SOFTWARE DEFINED NETWORK (UAV-Aided SDNET) . . . 9
3.1. System Model . . . 9
3.2. Channel Models . . . 11
3.2.1. Air-to-Ground (A2G) Communication Channel Model . . . 13
3.2.2. Air-to-Air (A2A) Communication Channel Model . . . 14
3.3. Problem Formulation . . . 14
3.4. Task Offloading Algorithms . . . 17
3.4.1. Aggressive-Wait Offloading Algorithm (AGG-1) . . . 19
3.4.2. Aggressive Tx-Order Offloading Algorithm (AGG-2) . . . 19
3.4.3. Adaptive Offloading Algorithm (ADP) . . . 19
3.4.4. Quasi-optimal Offloading Algorithm (Q-OPT) . . . 23
4. PERFORMANCE ANALYSIS . . . 25
4.1. Experimental Environment and Parameters . . . 25
4.2. System Components . . . 25

4.2.1. Vehicle Mobility Generation 27
4.2.2. Network Infrastructure 28
4.2.3. Controller Application 31
4.2.4. Main Simulation Driver 32
4.3. Experimental Scenarios and Performance Metrics 33
4.3.1. Scenario 1 - Task Inter-arrival Time ($1/\lambda$) 33
4.3.2. Scenario 2 - Average Vehicle Speed $\overline{V}$ 33
4.3.3. Scenario 3 - Processing Speed 34
4.4. Results and Performance Evaluation 34
4.4.1. Impact of Task Inter-arrival Time 34
4.4.1.1. Penalty Results 35
4.4.1.2. Task Processing Characteristics 37
4.4.1.3. Temporal Results for Task Completion 37
4.4.2. Impact of Average Vehicle Speed $\overline{V}$ 41
4.4.2.1. Penalty Results 41
4.4.2.2. Task Processing Characteristics 42
4.4.2.3. Temporal Results for Task Completion 43
4.4.3. Impact of Processing Speed 45
4.4.3.1. Penalty Results 45
4.4.3.2. Task Processing Characteristics 46
4.4.3.3. Temporal Results for Task Completion 46
4.5. Discussions 46
5. CONCLUSION AND FUTURE DIRECTIONS 50
REFERENCES 52

LIST OF FIGURES

Figure 3.1. UAV-Aided Software Defined Network (UAV-Aided SDNET) model. 10

Figure 3.2. Aggressive-Wait Offloading Algorithm (AGG-1) 20

Figure 3.3. Aggressive Tx-Order Offloading Algorithm (AGG-2) 21

Figure 3.4. Adaptive Offloading Algorithm (ADP) 22

Figure 3.5. Quasi-optimal Offloading Algorithm (Q-OPT) 24

Figure 4.1. System components. 27

Figure 4.2. SUMO mobility model source: Besiktas, Istanbul. 28

Figure 4.3. Vehicle-station pairing (Map: Beşiktaş, İstanbul). 30

Figure 4.4. Network topology. 32

Figure 4.5. Impact of task inter-arrival time on the objective function $\mathbb{D}$. . . . 35

Figure 4.6. Impact of task inter-arrival time on task delay composition. 38

Figure 4.7. System delay and deadline distribution for the Case 1.1 using ADP. 39

Figure 4.8. System delay and deadline distribution for the Case 1.2 using ADP. 40

Figure 4.9. System delay and deadline distribution for the Case 1.3 using ADP. 40

Figure 4.10. Impact of average vehicle speed on the objective function $\mathbb{D}$. . . . 41

Figure 4.11. Impact of average vehicle speed on task delay distribution. 44

Figure 4.12. Impact of processing speed on the objective function $\mathbb{D}$. 45

Figure 4.13. Impact of processing speed on task delay distribution. 49

LIST OF TABLES

Table 3.1. Channel Model Parameters 12

Table 3.2. Time and Duration Related Parameters 15

Table 3.3. System Parameters 16

Table 3.4. Algorithm Parameters 18

Table 4.1. Baseline Simulation Parameters 26

Table 4.2. Vehicle Parameters 29

Table 4.3. Impact of task inter-arrival time on QoS 36

Table 4.4. Impact of task inter-arrival time on task processing characteristics 37

Table 4.5. Impact of task inter-arrival time on temporal results for task completion 39

Table 4.6. Impact of average vehicle speed on QoS 42

Table 4.7. Impact of average vehicle speed on task processing characteristics . 43

Table 4.8. Impact of average vehicle speed on temporal results for task completion 44

Table 4.9. Impact of average processing speed on QoS 46

Table 4.10. Impact of average processing speed on task processing characteristics 47

Table 4.11. Impact of average processing speed on temporal results for task completion 48

LIST OF SYMBOLS

a_{K_i}	The arrival time slot of K_i
a_{P_i}	The arrival time slot of P_i
b_{P_i}	The departure time slot of P_i
b_{V_i}	The departure time slot of V_i
C_i	Indicator for K_i completion by system
c	The speed of light traveling through a vacuum
c_{K_i}	The completion time slot of K_i
D_i	The penalty score of K_i
$d_{U_i,U_j}(t)$	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
ER	Event received
f	The system carrier frequency
G_{tx,U_i}	The transmission gain of U_i
G_{rx,U_j}	The receiver gain of U_i
G_{tx,V_i}	The transmission gain of V_i
G_{rx,V_j}	The receiver gain of V_i
INFINITY	The positive infinity value which is greater than any other number
K_i	The i^{th} task
$L_{A,U_j}(t)$	The link capacity between $A \in \{U_i, K_i, P_i\}$ and U_j at t
$L_{K_i,P_j}(t)$	The link capacity between K_i and P_j at t
$L_{K_i,cloud}(t)$	The link capacity between K_i and the cloud server at t
$List_D$	Penalty score list
m	A2G channel parameter
N_K	The number of tasks
N_P	The number of processors
N_U	Number of UAVs
n	A2G channel parameter
O_{i,V_j}	If V_j is owner of K_i 1; otherwise 0

P_i	The i^{th} processor
$P_{U_i,U_j}(t)$	The average received power of UAV U_j from U_i at t
$P_{U_i,V_j}(t)$	The average received power of vehicle V_j from U_i at t
P_{tx,U_i}	Transmission power of U_i
P_{tx,V_i}	Transmission power of V_i
$PL_{LoS,U_i,V_j}(t)$	The path loss for LoS from U_i to V_j at t
$PL_{NLoS,U_i,V_j}(t)$	The path loss for NLoS from U_i to V_j at t
$PL_{FS,U_iU_j}(t)$	The free space path loss between U_i and U_j at t
$PL_{FS,U_i,V_j}(t)$	The free space path loss between U_i and V_j at t
$p_{LoS,U_i,V_j}(t)$	The probability of LoS connection between U_i and V_i at t
$p_{NLoS,U_i,V_j}(t)$	The probability of NLoS connection between U_i and V_i at t
p_{heavy}	The probability of task offload to the cloud if pool is heavily occupied
p_{light}	The probability of task offload to the cloud if pool is lightly occupied
p_{medium}	The probability of task offload to the cloud if pool is moderately occupied
Q_i	The queue of P_i
S_{K_i}	The size of K_i
S_{Q_i}	The queue size of P_i
T	Simulation time, the number of time slots
t	Current time step
$t_{deadline,i}$	The time budget for K_i to be processed before
$t_{pool,i}$	The time elapsed to be assign K_i
$\bar{t}_{pool}$	The average time spent on controller queue
$t_{process,i}$	The time elapsed to process K_i
$\bar{t}_{process}$	The average time spent during processing
$t_{queue,i}$	The time elapsed on processor queue for K_i
$\bar{t}_{queue}$	The average time spent on processor queue
$t_{total,i}$	The time elapsed to complete K_i
$\bar{t}_{pool}$	The average task completion time

$t_{tx,i}$	The time elapsed to transmit K_i
$\bar{t}_{tx}$	The average time spent during transmission
TP	Controller task pool
U_i	The i^{th} UAV
V_i	The i^{th} vehicle
w_i	The task priority times the task penalty score of K_i
w_{i,P_j}	w_i if K_i is assigned to P_j
x_{ij}	Decision variable for offloading K_i to P_j
y_i	Decision variable for offloading K_i to the cloud
z	The extra penalty oint if the task deadline is missed
α_i	The priority of K_i
β_{cloud}	The processing speed of cloud
β_i	The processing speed of P_i
$\mathbb{D}$	The average of prioritization-based weighted penalty points
η_{LoS}	The additional LoS attenuation factors due to the LoS connections
η_{NLoS}	The additional NLoS attenuation factors due to the NLoS connections
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
$\Theta_{decision}$	The maximum number of candidate solutions that the algorithm will attempt to solve within a time-window
$\theta_i(t)$	The elevation angle
λ	Exponentially distributed task generation rate
ρ	Air-to-air path loss exponent
σ^2	Variance of White Gaussian Noise

LIST OF ACRONYMS/ABBREVIATIONS

A2A	Air-to-Air
A2G	Air-to-Ground
ADP	Adaptive Offloading Algorithm
AGG-1	Aggressive-Wait Offloading Algorithm
AGG-2	Aggressive Tx-Order Offloading Algorithm
AGGs	AGG-1 and AGG-2
AP	Access Point
BS	Base Station
IAT	Inter-arrival Time
MB	Megabyte
KB	Kilobyte
LoS	Line of Sight
MBps	Megabyte per second
Mbps	Megabit per second
MEC	Mobile Edge Computing
NLoS	Non Line of Sight
Q-OPT	Quasi-optimal Offloading Algorithm
OF	OpenFlow
ONOS	Open Network Operating System
QoS	Quality of Service
RAM	Random Access Memory
RSSI	Received Signal Strength Indicator
SDN	Software Defined Networks
SSD	Solid-state Drive
SSID	Service Set Identifier
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
V2C	Vehicle-to-Cloud

V2I	Vehicle-to-Infrastructure
V2V	Vehicle-to-Vehicle
VANET	Vehicular Ad Hoc Network
VE	Vehicle Equipment
VN	Vehicular Network

1. INTRODUCTION

Unmanned Aerial Vehicles (UAVs) can be used to deploy a local network infrastructure when a quickly deployable solution is required, such as during demand surges, short-term network capacity requirements, or post-disaster communications and data collection use cases [1]. In that regard, they have three critical characteristics which make them promising [2]. First, located in the sky, they have a higher probability of connecting ground nodes and other UAVs via line-of-sight (LoS) links compared to vehicles and terrestrial infrastructures. Second, contrary to stationary ground infrastructure, UAVs can be dynamically deployed in response to dynamic 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.

On the ground level, novel intelligent mobility solutions such as self-driving cars are already in operation, and vehicle communications are getting more vital as user adoption increases. The security of connected vehicles depends on the quality of perception of their surroundings and networking as capabilities for information exchange and access play a crucial role for possible cooperation scenarios [3]. Among various applications, especially legacy vehicles, may not be able to handle all of the new applications and services due to their limited hardware [4]. In that case, UAVs can assist vehicular networks on that subject by providing cloud access, packet relaying or edge computing services. In this thesis, we have considered such a UAV-aided network and focused our efforts on the use case in which UAVs create a network for vehicles in times of emergency or ad hoc communications. 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. Under the conditions of limited network and resource availability, priority values are assigned to 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.

In order to cope with the mobility-based frequent topology change and dynamic system states, network softwarization is adopted. Therefore, network intelligence is logically centralized in software-based controllers while the control and data planes are separated in the SDN architecture. SDN controllers give a programmatic interface to network management, allowing any level network administration applications to be developed on top of it [5]. Our task orchestrator application on the SDN controller is also used for packet routing and vehicle network position tracking. It has allowed us in resolving several difficulties for packet routing and transmission due to the Wi-Fi mesh protocol.

1.1. Key Contributions

In this thesis, we elaborate on an important use case for UAVs in post-disaster rescue scenarios and create an emulation environment for our UAV-aided software-defined network architecture. Furthermore, we evaluate the performance of our heuristic task offloading algorithms in order to gain a better understanding of how such a system performs under various conditions. For comparison, a quasi-optimal solver utilizing a branch&bound algorithm has been developed. We also discuss potential challenges to be addressed if such a system is implemented in real life.

The contributions made in this thesis 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 done to other ground vehicles or a cloud processor service. The model's objective function is to minimize average penalty scores caused by task completion time.
- *Orchestrator application:* A reactive routing application is written in Java for ONOS to install and delete path flows by sniffing packets and listening to host events in the test environment. It creates an end-to-end flow path for both ARP and Ethernet packets, unlike the built-in reactive forwarding app, to avoid

multiple mesh-related compatibility issues.

- *Open-source contribution to Mininet-Wifi:* The `wmediumd` application, a WiFi delay and loss simulator written in C++ and used by the well-known Mininet-Wifi emulator, 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 and have shared it publicly.
- *Heuristics and quasi-optimal algorithm for task offloading:* Three heuristic algorithms and a branch&bound style quasi-optimal task offloading algorithm are implemented in Python. The latter works on the previously simulated data to get better insight for the evaluation of our heuristic algorithms with respect to near-optimal operation.
- *Implemention of A2G Channel Model:* The LoS probability based channel model is implemented on top of Mininet-Wifi and `wmediumd`.
- *Simulation management application:* Core Python application orchestrates all the components of the test environment step by step.

1.2. Thesis Outline

In this thesis, Chapter 2 provides an overview of current studies in our research area by discussing the related works in the literature. The shared literature works are grouped into UAV-Aided Vehicular Networks, Task Offloading, and Software Defined UAV Networks categories.

In Chapter 3, we explain the system-related models, formulate our problem and explain each of the proposed task offloading algorithms.

Next, in Chapter 4, we start with describing our test environment. The main components and their development process, as well as the problems encountered, are then presented. The chapter continues with an explanation of our test scenarios. The

results for each of predefined test scenarios are then presented and discussed one by one.

Chapter 5 is the conclusion of our thesis. We present a summary of the findings as well as the list of open questions which may form a basis for future research.

2. RELATED WORK

This chapter summarizes the existing research on the three main topics studied in our thesis. The first section includes works on UAV applications in vehicular networks. The second section 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 section.

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 *et al.* [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 [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.

Pourbaba *et al.* [9] proposed a sub-optimal method to increase coverage of full-duplex relay network among ground vehicles by positioning UAVs. They have used a set of predefined UAV locations and have 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. On 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. Software-Defined UAV Networks

SDN allows for network programmability by separating the control layer from data-forwarding network elements. SDN architecture can be used to reduce network configuration time and cost, which is critical for rapidly changing UAV networks [12]. It can also handle offloading requests and associated routing management based on network traffic [13].

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 [14]. 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 [15]. 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.* [16] 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.* [17] 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.

2.3. 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 recently researched as part of edge-cloud continuum paradigm.

Chen et al. [18] 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 [19] 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. Compared to single-action base approaches, the approach performs 43 percent to 97 percent better.

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.

As an example of multistage optimization works, a UAV-assisted mobile offloading and the trajectory optimization problem is studied in [20]. 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.

3. PROPOSED MODEL: TASK OFFLOADING IN UAV-AIDED SOFTWARE DEFINED NETWORK (UAV-Aided SDNET)

This chapter discusses the proposed UAV-aided SDN-enabled network and its system components. First, the task assignment procedure, QoS evaluation methodology, and the communication among devices are explained. Second, Air-to-Air (A2A) and Air-to-Ground (A2G) channel models are defined. Third, the logic behind the problem we are dealing with is explained and the optimization problem is formulated. Lastly, proposed task offloading algorithms are introduced and explained separately.

3.1. System Model

Our network model consists of N_U UAVs (U_i), one of which is connected to a ground base station (BS) which has a backhaul link to the Internet and relevant cloud services. UAVs are mounted with wireless Access Points (APs) controlled by an SDN controller with OpenFlow protocol and having multiple Wi-Fi interfaces. One UAV has also an integrated computational unit (e.g., a Single Board Computer (SBC)), which runs that SDN controller and functions as the network controller. For each UAV, one interface provides Wi-Fi service to terrestrial vehicles, while the others create Mesh connections with nearby UAVs. The system is shown in Figure 3.1.

A 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, informs both parties about the assignment. Only the identifier data for tasks are stored on the controller's task pool, not the task data's itself meaning the

data transfers are made from the task owner directly to the processor units.

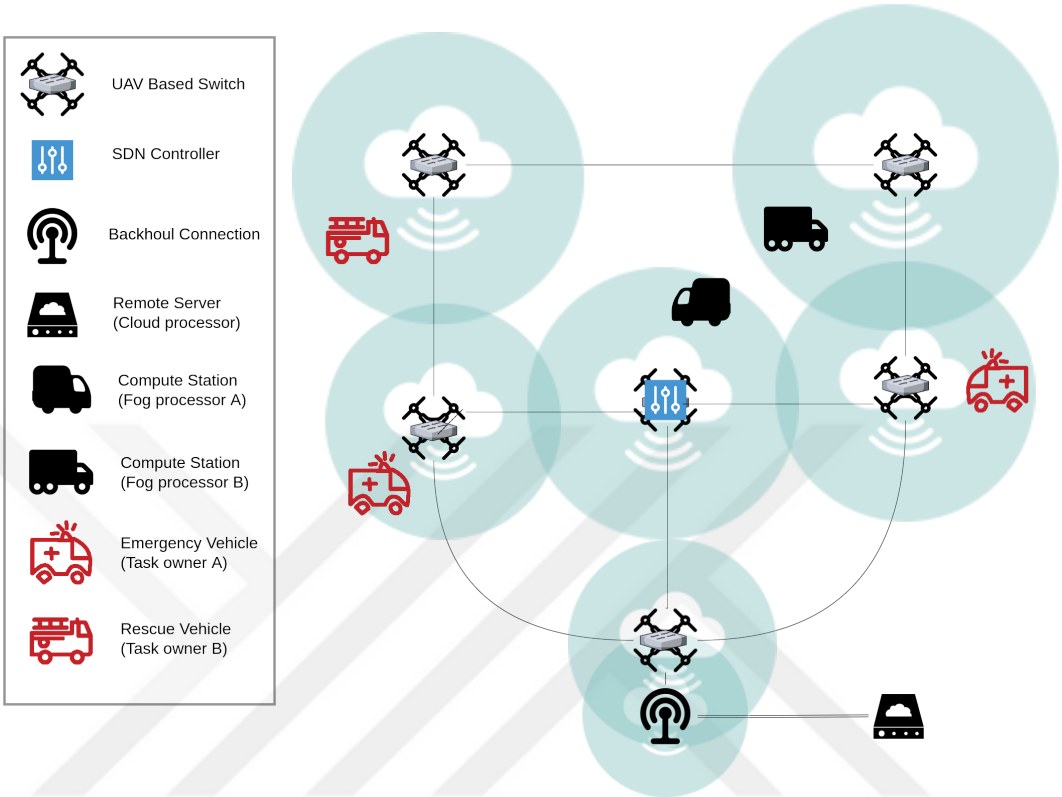


Figure 3.1. UAV-Aided Software Defined Network (UAV-Aided SDNET) model.

The processor vehicles have their own limited-sized 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 the area 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 point z if the deadline has already been missed; otherwise, it receives no penalty.

The average inter-arrival time for the tasks generated by the vehicles is exponentially distributed with rate represented as λ. Upon a task arrival, if the task pool is not empty, the task can be offloaded or migrated to the cloud processing server with a probability depending on the occupancy of the task pool. Essentially, we consider minimizing average prioritization-based weighted penalty points for task offloading management as a QoS requirement and thus propose algorithms for that objective.

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 detections.

The 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 [21].

3.2. Channel Models

We chose two widely used path loss models [22–24] and applied them in our network simulations since there is no consensus on a proper path loss model for UAV-enabled networks [25].

In our simulation environment, calculated Received Signal Strength Indicator (RSSI) values are used to get probabilistic data rates from the RSSI-transmission rate matrix. Matrix values contain the probability of the Wi-Fi rate at the given RSSI value. These data rates are 6, 9, 12, 18, 24, 36, 48 and 54 Mbps for the IEEE 802.11g standard, which we used. The definitions of parameters used in the channel models are given in Table 3.1.

Table 3.1. Channel Model Parameters.

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
c	The speed of light traveling through a vacuum
f	The system carrier frequency
$d_{U_i,U_j}(t)$	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
$p_{LoS,U_i,V_j}(t)$	The probability of LoS connection between U_i and V_i at t
$p_{NLoS,U_i,V_j}(t)$	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_j}	The receiver gain of U_j
G_{rx,V_j}	The receiver gain of V_j
$P_{U_i,U_j}(t)$	The average received power of U_j from U_i at t
$P_{U_i,V_j}(t)$	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
$L_{A,U_j}(t)$	The link capacity between $A \in \{U_i, V_i\}$ and U_j at time slot t
$\theta_i(t)$	The elevation angle at t
ρ	A2A path loss exponent

3.2.1. Air-to-Ground (A2G) Communication Channel Model

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

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

$$PL_{LoS,U_i,V_j}(t) = PL_{FS,U_i,V_j}(t) + \eta_{LoS} \tag{3.1}$$

$$PL_{NLoS,U_i,V_j}(t) = PL_{FS,U_i,V_j}(t) + \eta_{NLoS} \tag{3.2}$$

where the free space path loss $PL_{FS,U_i,V_j}(t)$ is calculated as

$$PL_{FS,U_i,V_j}(t) = 20log(4\pi d_{U_i,V_j}(t)f/c). \tag{3.3}$$

The average path loss is

$$\bar{PL}_{U_i,V_j}(t) = p_{LoS,U_i,V_j}(t)PL_{LoS,U_i,V_j}(t) + p_{NLoS,U_i,V_j}(t)PL_{NLoS,U_i,V_j}(t) \tag{3.4}$$

where $p_{LoS,U_i,V_j}(t)$ and $p_{NLoS,U_i,V_j}(t)$ are the probabilities of LoS and NLoS and they are computed as

$$p_{LoS,U_i,V_j}(t) = \frac{1}{1 + m \times exp(-n(\theta_i(t) - m))} \tag{3.5}$$

$$p_{NLoS,U_i,V_j}(t) = 1 - p_{LoS,U_i,V_j}(t). \tag{3.6}$$

Then, the average received power of U_i from V_j is

$$P_{U_i,V_j}(t) = P_{tx,U_i} + G_{tx,U_i} + G_{rx,V_j} - \bar{PL}_{U_i,V_j}(t). \tag{3.7}$$

3.2.2. Air-to-Air (A2A) Communication Channel Model

For UAV-to-UAV communication, free-space channel model is utilized. When U_i transmits signals to U_j, the received power at U_j is expressed as

$$P_{U_i,U_j}(t) = P_{tx,U_i} + G_{tx,U_i} + G_{rx,U_j} - PL_{FS,U_iU_j}(t) \tag{3.8}$$

where

$$PL_{FS,U_iU_j}(t) = 10log(4\pi d_{U_i,U_j}(t)f/c)^{\rho}. \tag{3.9}$$

3.3. Problem Formulation

A task's lifetime in our UAV-Aided SD-Net 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 assigned to one if K_i is decided to be offloaded to P_j and similarly, y_i is assigned to one if K_i is decided to be offloaded to the cloud. These decision variables are equal to zero if not one and their relation can be presented as

$$C_i = \sum_{j=1}^{N_P} x_{ij} + y_i \qquad \forall i. \tag{3.10}$$

The time-related task parameters are given in Table 3.2. $t_{total,i}$ is the total time needed to complete K_i and it is defined as

$$t_{total,i} = t_{pool,i} + t_{tx,i} + t_{queue,i} + t_{process,i} \qquad \leftrightarrow \qquad C_i = 1 \tag{3.11}$$

where

$$t_{process,i} = S_{K_i}/(\sum_{j=1}^{N_P} \beta_i x_{ij} + \beta_{cloud} y_i) \qquad \leftrightarrow \qquad C_i = 1 \tag{3.12}$$

$$t_{tx,i} = S_{K_i}/(\sum_{j=1}^{N_P} L_{K_i,P_j}(t) x_{ij} + L_{K_i,cloud}(t) y_i) \qquad \leftrightarrow \qquad C_i = 1. \tag{3.13}$$

Table 3.2. Time and Duration Parameters.

Symbol	Definition
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
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
c_{K_i}	The completion time slot of K_i

D_i is the penalty score of K_i, it is calculated as

$$D_i = \begin{cases} max(t_{total,i} - t_{deadline,i}, 0) & \text{if } C_i = 1 \\ 0 & \text{if } (C_i = 0) \wedge (O_{i,V_j} = 1) \wedge (b_{V_j} < t_{deadline,i}) \\ b_{V_j} - t_{deadline,i} + z & \text{if } (C_i = 0) \wedge (O_{i,V_j} = 1) \wedge (b_{V_j} > t_{deadline,i}). \end{cases} \tag{3.14}$$

In an emergency case, some of the service requests can be much more vital, and lives may depend on their deadlines. In such a system, meeting these deadlines would matter the most while constructing the problem. Instead of minimizing the average delay, we use deadlines for task offloading order arrangement to see how the system performs under pressure.

The problem that we address is the minimization of the average task penalty weighted by task priority in a UAV-aided Software-Defined Network. The list of parameters that we utilize in the problem formulation is given in Table 3.3. Besides, we

utilize indicator function 1_w that is assigned to one if w is true, zero otherwise. The task penalty weighted by task priority for K_i is calculated as

$$w_i = \alpha_i \cdot D_i. \tag{3.15}$$

Table 3.3. 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
S_{K_i}	The size 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
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

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. The optimization problem is formulated as

$$\min_{x,y} \quad \mathbb{D}(\lambda, \boldsymbol{R}) = \sum_{i=1}^{N_K} w_i / N_K \tag{3.16}$$

subject to:

$$\sum_{j=1}^{N_P} x_{ij} + y_i \leq 1 \qquad \forall i \tag{3.17}$$

$$x_{ij} \cdot c_{K_i} + t_{process,i} + t_{tx,i} \leq b_{P_j} \qquad \forall i \tag{3.18}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, \forall j \tag{3.19}$$

$$y_i \ \in \{0, 1\} \qquad \forall i. \tag{3.20}$$

Equation (3.17) 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. Equation (3.18) guarantees that the task should be completed before the processor departs.

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 proved to be NP-hard [26] while the penalty scoring in our system model increases the complexity due to its nonlinear behavior upon task owner departures.

3.4. Task Offloading Algorithms

The system time is discretized into steps to perform offloading decisions on the generated time steps. The exact time a vehicle arrives/departs 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 final algorithm is named as *Quasi-Optimal* offloading algorithm and oracle-type decision maker since it operates by extracting better solutions from recorded past data. Since this algorithm cannot process all the possible solutions from all the data recorded due to combinatorial explosion, it is quasi-optimal. The algorithm parameters are listed in Table 3.4.

Table 3.4. Algorithm Parameters.

Symbol	Definition
ER	Event for system event received
INFINITY	The positive infinity value which is greater than any other number
$List_D$	Penalty score list
TP	Controller task pool
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-window
$Event.TaskComplete$	Event for task completion
$Event.TxComplete$	Event for task transmission complete
$Event.NewProcessor$	Event for arrival of a task processor
$Event.ProcessorDeparture$	Event for departure of a task processor
$Event.TaskOwnerDeparture$	Event for departure of a task owner
$Event.NewTask$	Event for arrival of a new task to *TP*

3.4.1. Aggressive-Wait Offloading Algorithm (AGG-1)

The orchestrator application in 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-2 in order to complete the previously assigned task, probably the one closer to its deadline, earlier. The rationale is not to waste completed processing work and satisfy more deadlines. If the next assigned task is downloaded before the first one, the second task will be processed after the first is completed. AGG-1 algorithm described in Figure 3.2 runs until the simulation time ends.

3.4.2. 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) determines the order in which tasks are processed; the earlier downloaded one is processed first. During the handling of *Event.TaskComplete* and *Event.TxComplete*, the related processor does not check whether the downloaded task was the first assigned one, contrary to AGG-1. Details are provided in Figure 3.3.

3.4.3. Adaptive Offloading Algorithm (ADP)

The third scheme is named as Adaptive Offloading Algorithm. 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 on *TP* and more processor-task combinations upon arrival of new processor nodes. Figure 3.4 shows the details of the algorithm.

Wait for a system event *ER*;
if *ER is Event.TaskComplete* **then**
    Mark the task as completed and remove it from the processor queue;
    Increase the associated processor's available queue size by the size of the completed task;
    if *There is a task in the processor queue and K_i, the one on the first line of the queue is already downloaded* **then**
        Start to process K_i and mark it as processing;
        Calculate $t_{process,i}$ and add *Event.TaskComplete* to global event registry for $t_{process,i}$ slots later.
    end
else if *ER is Event.TxComplete* **then**
    Mark the transmitted task K_i as waiting on queue;
    if *K_i is the one on the first line of the same processor's queue* **then**
        Start to process K_i and mark it as processing;
        Calculate $t_{process,i}$ and add *Event.TaskComplete* to global event registry for $t_{process,i}$ later.
    end
else if *ER is Event.NewProcessor* **then**
    Add new vehicle to the available processors;
else if *ER is Event.ProcessorDeparture* **then**
    Remove the vehicle from available processors;
    From the tasks assigned to the related processor, add the ones which are in the transmission phase to *TP*;
else if *ER is Event.TaskOwnerDeparture* **then**
    foreach *K_i belongs to the departed task owner* **do**
        if *K_i 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 processor's queue and mark K_i as failed;
            Increase available queue size of the related processor by S_{K_i};
        end
    end
else if *ER is Event.NewTask* **then**
    Offload to cloud with a probability depending on the current *TP* size;
    if *It is not assigned to cloud* **then**
        Add it to *TP*;
    end
end
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;
    end
    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 queue size of P_j by S_{K_i};
end

Figure 3.2. Aggressive-Wait Offloading Algorithm (AGG-1).

Wait for a system event *ER*;
if *ER is Event.TaskComplete* **then**
    Mark the task as completed and remove it from the processor queue;
    Increase the associated processor's available queue size by the size of the completed task;
    if *There is already downloaded task in the same processor's queue* **then**
        Start to process downloaded first task K_i from the queue considering queue order and mark it as processing;
        Calculate $t_{process,i}$ and add *Event.TaskComplete* to global event registry for $t_{process,i}$ later.
    end
else if *ER is Event.TxComplete* **then**
    Mark the transmitted task K_i as waiting on queue;
    if *There is no task processed currently on the same processor* **then**
        Start to process K_i and mark it as processing;
        Calculate $t_{process,i}$ and add *Event.TaskComplete* to global event registry for $t_{process,i}$ later.
    end
else if *ER is Event.NewProcessor* **then**
    Add new vehicle to the available processors;
else if *ER is Event.ProcessorDeparture* **then**
    Remove the vehicle from available processors;
    From the tasks assigned to the related processor, add the ones which are in the transmission phase to *TP*;
else if *ER is Event.TaskOwnerDeparture* **then**
    foreach K_i *belongs to the departed task owner* **do**
        if K_i *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 processor's queue and mark K_i as failed;
            Increase available queue size of the related processor by S_{K_i};
        end
    end
else if *ER is Event.NewTask* **then**
    Offload to cloud with a probability depending on the current *TP* size;
    if *It is not assigned to cloud* **then**
        Add it to *TP*;
    end
end
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;
    end
    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 queue size of P_j by S_{K_i};
end

Figure 3.3. Aggressive Tx-Order Offloading Algorithm (AGG-2).

Wait for a system event *ER*;
if *ER is Event.TaskComplete* **then**
    Mark the task as completed and remove it from the processor queue;
    Increase the associated processor's available queue size by the size of the completed task;
    if *There is already downloaded task in the same processor's queue* **then**
        Start to process downloaded first task K_i from the queue considering queue order and mark it as processing;
        Calculate $t_{process,i}$ and add *Event.TaskComplete* to global event registry for $t_{process,i}$ later.
    end
else if *ER is Event.TxComplete* **then**
    Mark the transmitted task K_i as waiting on queue;
    if *There is no task processed currently on the same processor* **then**
        Start to process K_i and mark it as processing;
        Calculate $t_{process,i}$ and add *Event.TaskComplete* to global event registry for $t_{process,i}$ later.
    end
else if *ER is Event.NewProcessor* **then**
    Add new vehicle to the available processors;
else if *ER is Event.ProcessorDeparture* **then**
    Remove the vehicle from available processors;
    From the tasks assigned to the related processor, add the ones which are in the transmission phase to *TP*;
else if *ER is Event.TaskOwnerDeparture* **then**
    foreach K_i *belongs to the departed task owner* **do**
        if K_i *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 processor's queue and mark K_i as failed;
            Increase available queue size of the related processor by S_{K_i};
        end
    end
else if *ER is Event.NewTask* **then**
    Offload to cloud with a probability depending on the current *TP* size;
    if *It is not assigned to cloud* **then**
        Add it to *TP*;
    end
end
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;
    end
    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 queue size of P_j by S_{K_i};
end

Figure 3.4. Adaptive Offloading Algorithm (ADP).

3.4.4. 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. This processed data includes:

- tasks' details such as arrival time, size, deadline
- vehicle positions
- vehicles' associated AP and RSSI values
- processor details

In order to narrow down candidate solutions, first arrival&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 fail on purpose, and offloading to the cloud are also added to these lists. Even the most under-loaded set up 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. 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. Q-OPT algorithm pseudocode is given in Figure 3.5.

Filter the candidate solutions by applying Equation (3.18);
$Solution_List = []$;
for $i = 1$, $i{+}{+}$, *while* $i < N_K$ **do**
    $window_{end} = T + 1$;
    do
        $window_{end} -= 1$;
        $Time_Window = (a_{K_i}, window_{end})$;
        $Tasks_In_Window$ is the tasks generated in $Time_Window$;
        $num_{combination}$ = row number of the Cartesian product of decisions of tasks in $Tasks_In_Window$;
    while $num_{combination} > \Theta_{decision}$;
    $combinations$ = the Cartesian product of decisions of tasks in $Tasks_In_Window$;
    num_{tasks} = The number of the tasks in $Tasks_In_Window$;
    $best_result_{window}$ = INFINITY;
    foreach *Row combinationincombinations* **do**
        $result_{window} = 0$;
        $k = 0$;
        while $k < num_{tasks}$ **do**
            $result_{decision}$ = The penalty score for applying action, $combination[k]$;
            $result_{window}$ += $result_{decision}$;
            if $result_{window} > best_result_{window}$ **then**
                break;
            end
            $k{+}{+}$;
        end
        if $k \neq N_K$ **then**
            Bounding Action: Skip the next combinations starting with the decisions as in combination[0:k];
        end
        else if $best_result_{window} > result_{window}$ **then**
            $best_result_{window} = result_{window}$;
            $Solution_List[i] = combination[0]$;
        end
    end
end
return Solution_List;

Figure 3.5. Quasi-optimal Offloading Algorithm (Q-OPT).

4. PERFORMANCE ANALYSIS

This chapter discusses the characteristics of UAV-aided SD Network and the performance evaluation of the task offloading techniques. First, the simulation environment and its components are introduced, followed by how the development progressed. Then, three main scenarios for simulation runs are given. Finally, simulation results and performance measures are presented, and each scenario is analyzed in detail.

4.1. Experimental Environment and Parameters

We use a machine with Intel Core i7-10875H (2.30GHz × 8), 16GB RAM, and 256GB SSD storage to run our tests. We use ONOS [27] v2.8 on Ubuntu 21.04 as SDN controller. Our custom forks [28, 29] from Mininet-WiFi [30] v2.4.3 and `wmediumd` are used to create host devices, virtual stations and APs. For virtual switch emulation, Open vSwitchv2.15 is utilized [31]. For traffic generation from vehicles in our experiments, iPerf v3.10 is adopted [32]. For developing the network control and packet delivery applications, Python v3.9.5 is used. For vehicle mobility simulation Eclipse SUMO v1.11.0 is used [33]. Table 4.1 contains the baseline simulation parameters.

4.2. System Components

There are four main application modules for implementing and running simulations for our task offloading system in a UAV-aided SD Network environment. These are:

- Vehicle Mobility Generation
- Network Infrastructure
- Controller Application
- Main Simulation Driver

Table 4.1. 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

The system components and their interactions are shown in Figure 4.1 and detailed in the following parts.

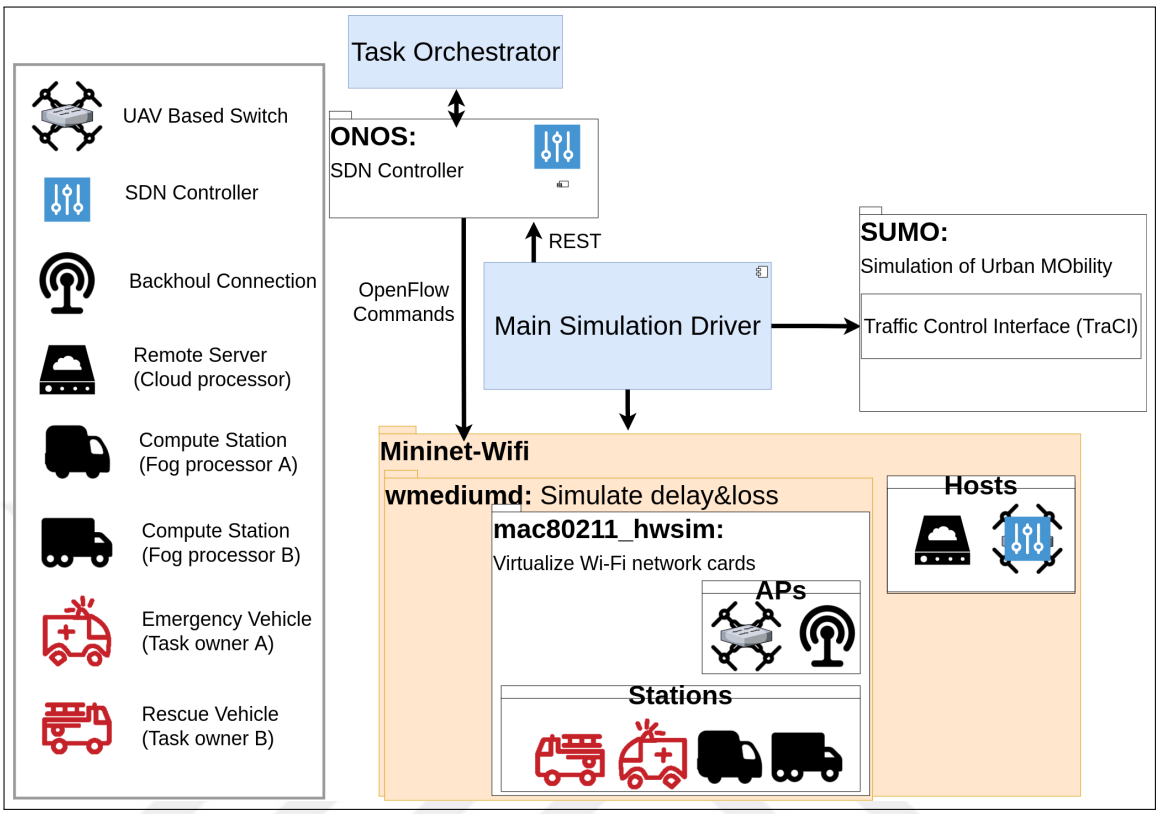


Figure 4.1. System components.

4.2.1. Vehicle Mobility Generation

We have chosen an area of 1 square kilometer from Beşiktaş District (Figure 4.2) in Istanbul to reflect its 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 4.2.

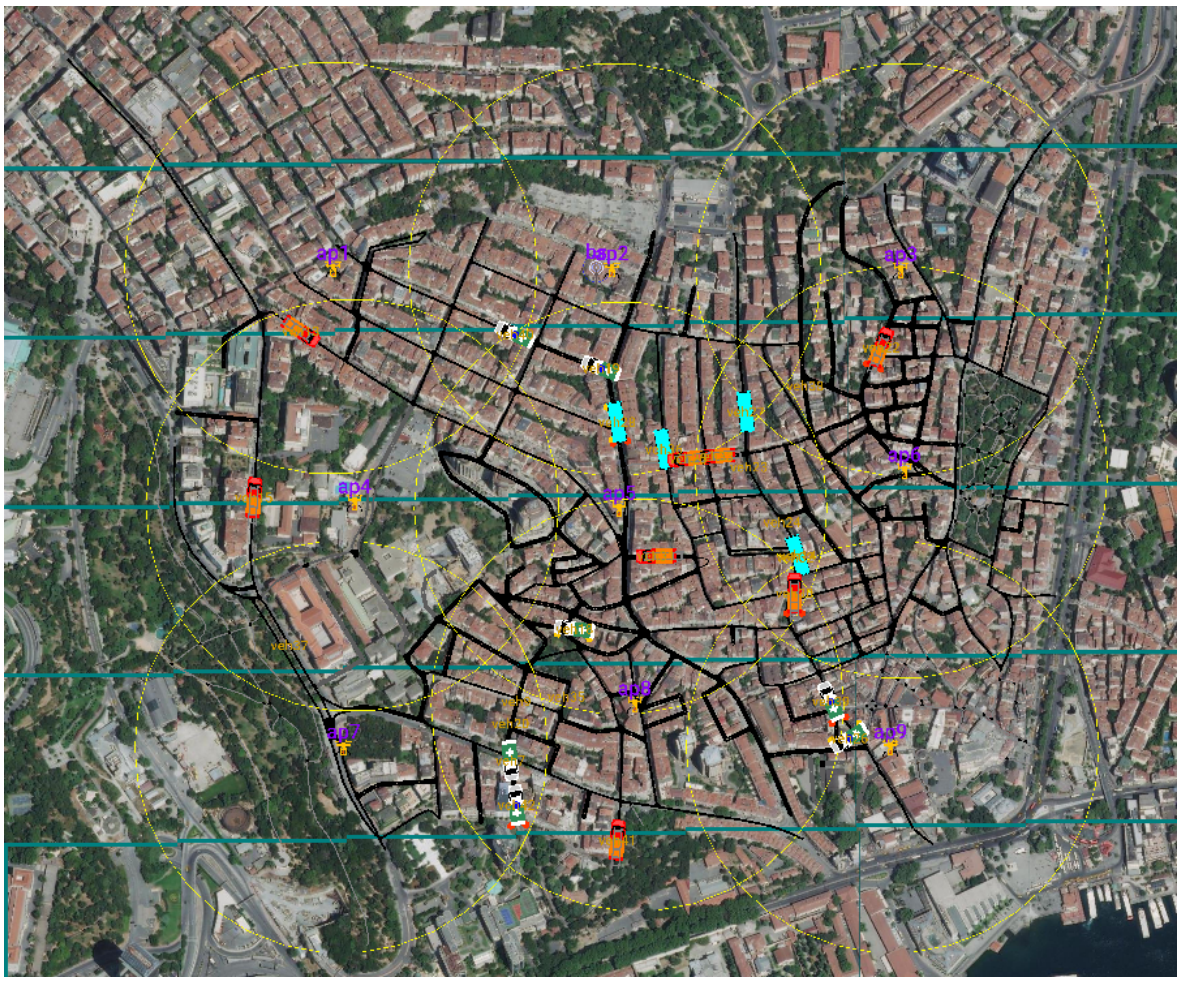


Figure 4.2. SUMO mobility model source: Besiktas, Istanbul.

4.2.2. Network Infrastructure

`mac80211_hwsim` is a Linux operating system kernel module and it can virtualize WiFi network cards. Mininet-WiFi, a fork of Mininet [34], 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 radios within the same channel. Mininet-WiFi also benefits from `wmediumd`, an application. `wmediumd` 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.

Table 4.2. 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	-	-	-

All Wi-Fi networks are contention-based and time-division duplexing systems [35]. In `wmediumd`, a queue mechanism is introduced to order packets to reflect delay caused by this concept. However, when the 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 the scratch [36]. 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 talking to each other by sniffing AP-Station packet transmissions and grouping them. Average CPU usage by `wmediumd` was at 25% level during simulation. The code improvement also gives manual configuration ability to Mininet-WiFi end user to setup the system for advanced topologies. All the developments were shared on [28, 29] and pull requests to the project repos were made as well.

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 bit spaghetti code.

Nine APs representing UAVs and one AP representing the BS have been added to the network in order to cover the selected 1 square kilometer area where each UAV covers around 350 meters. The numbers of generated APs and stations highly affect the initialization computations, clean-up operations and RAM usage so we have limited the area to this area size. UAVs are placed in a grid formation, and one of them is randomly selected for BS connection. Maximum 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, AP-AP connections Wi-Fi 802.11s mesh protocol is used. Different AP and mesh channels are assigned to the neighbor device interfaces.

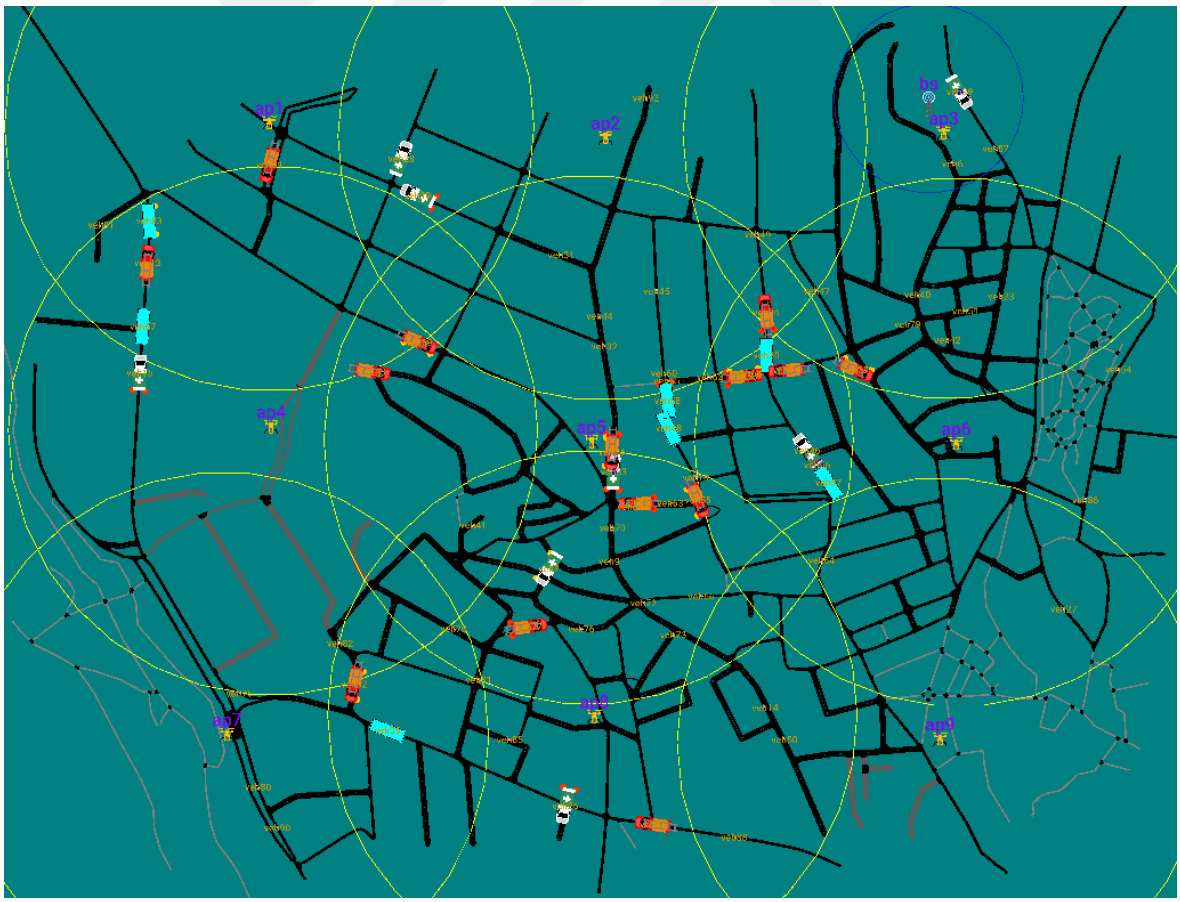


Figure 4.3. Vehicle-station pairing (Map: Beşiktaş, İstanbul).

Mininet-WiFi station objects are initially configured but kept at a distant location from the network, keeping them at hot stand-by to integrate into the simulation when needed. In that regard, a station is assigned to a arriving vehicle and its location is updated. For instance, each vehicle on Figure 4.3 has been assigned to its own station. Upon the vehicle's departure, the station is sent back to the remote to be used later on. 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. *Localhost* machine is also linked to the system by using the Network Address Translation (NAT) feature of Mininet for debug purposes.

`iperf3.10` 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 shells.

4.2.3. Controller Application

Two ONOS 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 reactive forwarding app floods the packet while our app installs all the flow rules required for an end-to-end communication. The 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 [37]. 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 can be seen in Figure 4.4.

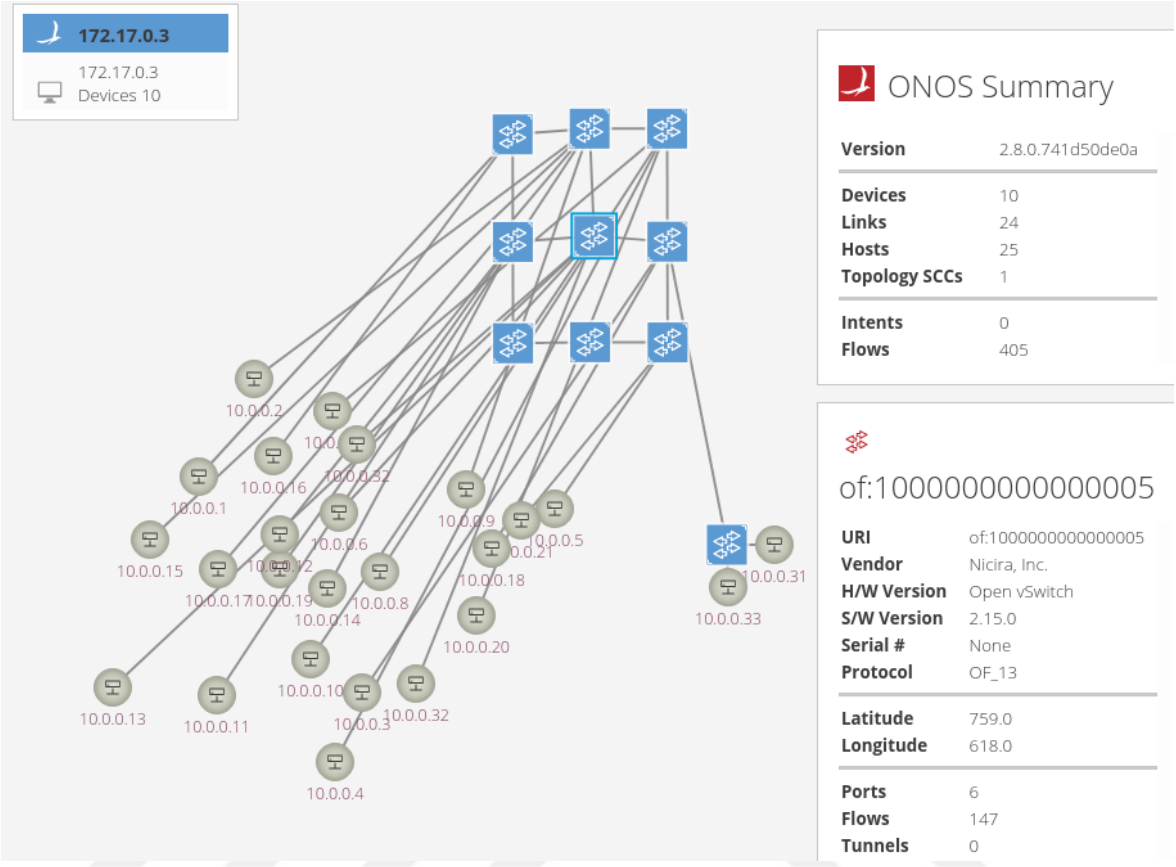


Figure 4.4. Network topology.

4.2.4. 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 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. As a side note, we have observed that Netcat client command like `echo TEST | nc IP PORT -w 1` runs much faster than a basic Python socket programming method.

4.3. Experimental Scenarios and Performance Metrics

In our experiments to evaluate our proposed framework, we have selected three main scenarios and selected three different cases for each to investigate the proposed system and the efficiency of our heuristic algorithms. 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 next section.

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

The exponentially distributed expected time value between consecutive tasks is $1/\lambda$. This 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:

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

4.3.2. Scenario 2 - Average Vehicle Speed $\overline{V}$

Starting from the average walking speed of 5 km/h, we analyze the default case 20 km/h and 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:* $\overline{V} = 5$ km/h
- *Case 2.2:* $\overline{V} = 20$ km/h
- *Case 2.3:* $\overline{V} = 40$ km/h

4.3.3. Scenario 3 - Processing Speed

In this scenario, we have investigated the effect of the ratio of the processing time to the transmission time by changing the processing speed of different task processor types. For convenience, labels such as slow, medium, and fast will be used during discussions.

- *Case 3.1:* (Slow) β_i = 0.7MBps (Type A), 0.5MBps (Type B)
- *Case 3.2:* (Medium) β_i = 2MBps (Type A), 1.5MBps (Type B)
- *Case 3.3:* (Fast) β_i = 5MBps (Type A), 4MBps (Type B)

4.4. Results and Performance Evaluation

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.

4.4.1. 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 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.

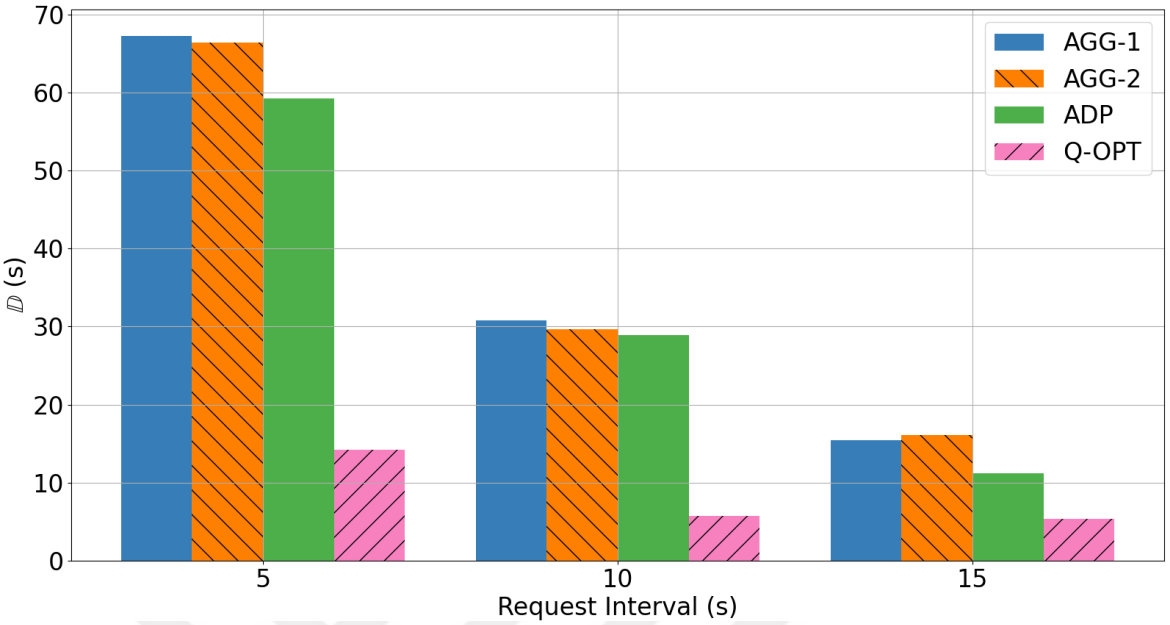


Figure 4.5. Impact of task inter-arrival time on the objective function $\mathbb{D}$.

4.4.1.1. Penalty Results. This part of the tests shows how well the algorithms performed in terms of the main objective function, minimization of the average of prioritization-based weighted penalty points when the network load changes. In Figure 4.5, the impact of all three cases can be seen. As the load increases, there is an exponential increase pattern of $\mathbb{D}$ for all the algorithms. 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.

Table 4.3. Impact of task inter-arrival time on QoS.

	Case #	$1/\lambda$	$\mathbb{D}$	Task Failure Ratio	Cloud Ratio
AGG-1	*1.1*	5	67.29	0.396	0.384
AGG-1	*1.2*	10	30.84	0.193	0.024
AGG-1	*1.3*	15	15.41	0.088	0.012
AGG-2	*1.1*	5	66.43	0.405	0.377
AGG-2	*1.2*	10	29.67	0.201	0.029
AGG-2	*1.3*	15	16.12	0.095	0.021
ADP	*1.1*	5	59.22	0.339	0.517
ADP	*1.2*	10	28.9	0.19	0.123
ADP	*1.3*	15	11.23	0.074	0.033
Q-OPT	*1.1*	5	14.19	0.128	0.051
Q-OPT	*1.2*	10	5.78	0.114	0.128
Q-OPT	*1.3*	15	5.36	0.107	0.224

Along with $\mathbb{D}$, we can see the service delivery skills of the algorithms in Table 4.3. 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 in a high percentage. Also, 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 the two main reasons behind this success. Firstly, the task download ratio per processor is always less than 1, which is not the case for the other two heuristic algorithms. Therefore, it leads to a less congested network and currently offloading tasks' transmission times to be shortened. Secondly, since the task pool gets filled more easily, utilization of the cloud starts earlier, while the AGG algorithms start to offload to the cloud after all their queues are full.

4.4.1.2. Task Processing Characteristics. From Table 4.4, we can say that 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.

Table 4.4. Impact of task inter-arrival time on task processing characteristics.

	Case #	$1/\lambda$	Task Failure Ratio	Cloud Ratio	$\bar{t}_{pool}$
AGG-1	*1.1*	5	0.396	0.384	24.89
AGG-1	*1.2*	10	0.193	0.024	11.29
AGG-1	*1.3*	15	0.088	0.012	5.71
AGG-2	*1.1*	5	0.405	0.377	22.13
AGG-2	*1.2*	10	0.201	0.029	11.6
AGG-2	*1.3*	15	0.095	0.021	8.22
ADP	*1.1*	5	0.339	0.517	28.14
ADP	*1.2*	10	0.19	0.123	19.57
ADP	*1.3*	15	0.074	0.033	10.67
Q-OPT	*1.1*	5	0.128	0.051	40.27
Q-OPT	*1.2*	10	0.114	0.128	17
Q-OPT	*1.3*	15	0.107	0.224	12.05

4.4.1.3. Temporal Results for Task Completion. We can get insights about the system operation by looking at the life-cycle of tasks. In Figure 4.6, 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. Numerical values are also represented in Table 4.5. 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 some room for higher loads which would make $\bar{t}_{total}$ over 300 seconds.

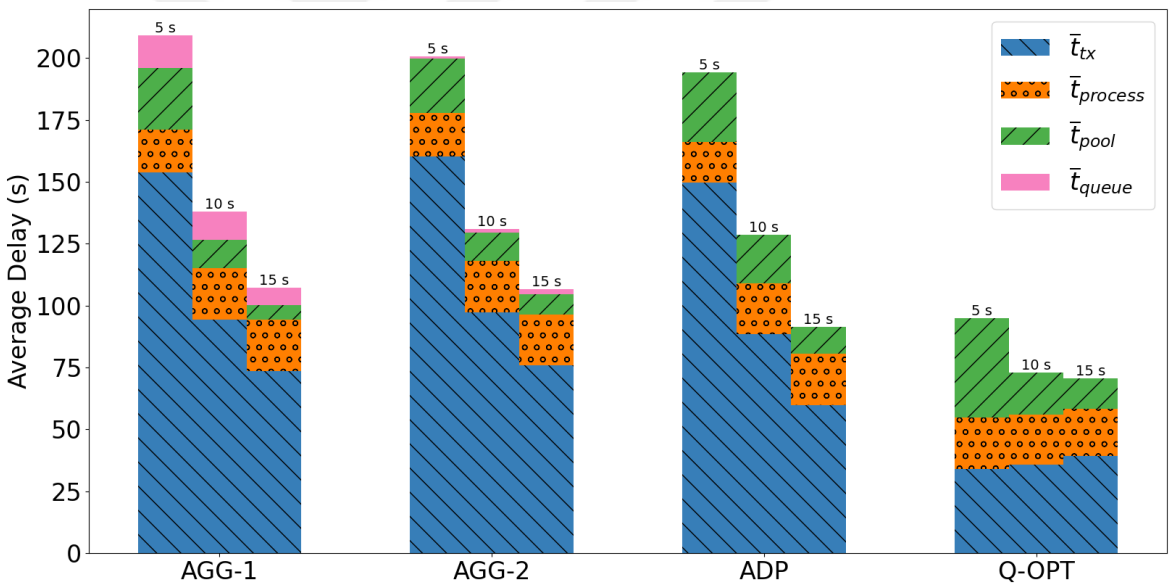


Figure 4.6. Impact of task inter-arrival time on task delay composition.

As a final analysis of this scenario, the system delay and task deadline of the tasks from simulations using the ADP assignment algorithm are scattered in Figure 4.7 for the Case 1.1, in Figure 4.8 for the the Case 1.2 and lastly in Figure 4.9 for the 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. High deviation around the average implies the system service instability at this level of load. Contrarily, in the Case 1.3, our system runs at a steady state.

Table 4.5. Impact of task inter-arrival time on temporal results for task completion.

	Case #	$1/\lambda$	$\bar{t}_{total}$	$\bar{t}_{tx}$	$\bar{t}_{queue}$	$\bar{t}_{process}$
AGG-1	*1.1*	5	209.27	153.71	13	17.46
AGG-1	*1.2*	10	138.02	94.43	12	20.8
AGG-1	*1.3*	15	107.07	73.59	7	20.76
AGG-2	*1.1*	5	200.54	160.24	1	17.45
AGG-2	*1.2*	10	131.07	97.19	2	20.77
AGG-2	*1.3*	15	106.64	75.82	2	20.56
ADP	*1.1*	5	194.17	149.76	0	16.27
ADP	*1.2*	10	128.57	88.57	0	20.4
ADP	*1.3*	15	91.34	59.75	0	20.87
Q-OPT	*1.1*	5	94.96	33.96	0	20.73
Q-OPT	*1.2*	10	73.05	35.7	0	20.35
Q-OPT	*1.3*	15	70.44	39.34	0	19.06

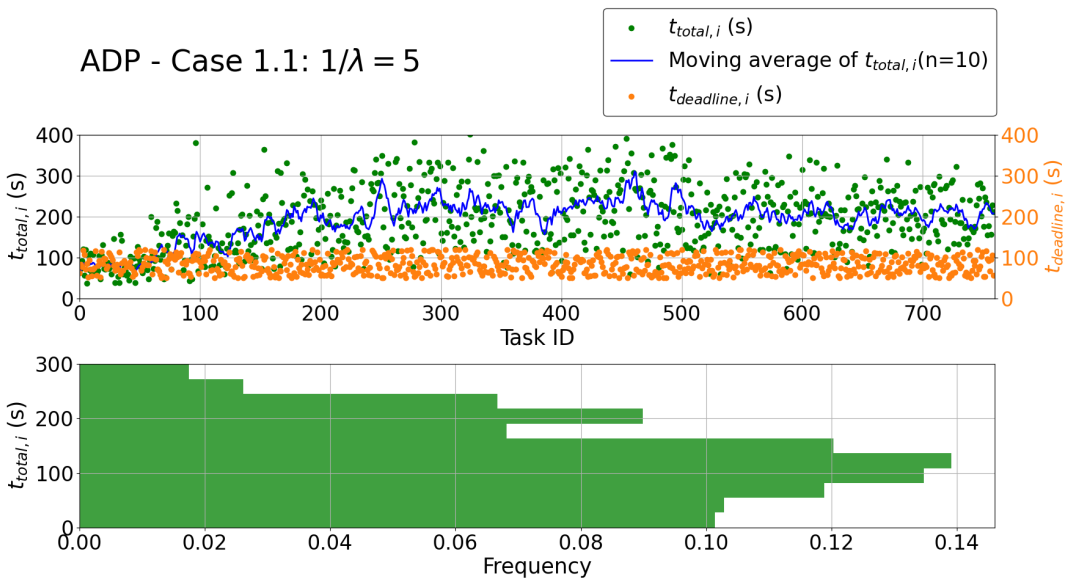


Figure 4.7. System delay and deadline distribution for the Case 1.1 using ADP.

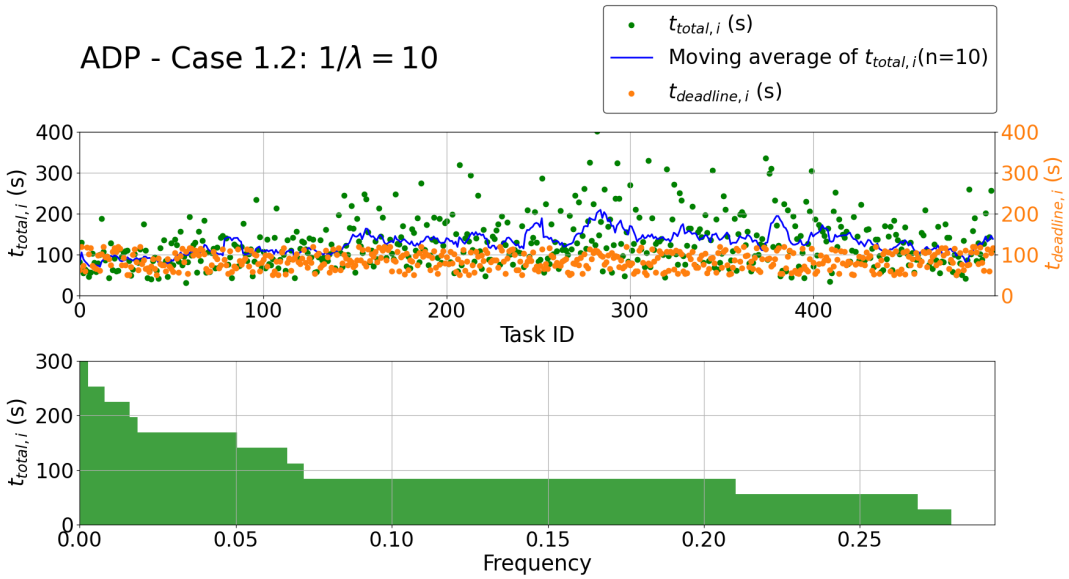


Figure 4.8. System delay and deadline distribution for the Case 1.2 using ADP.

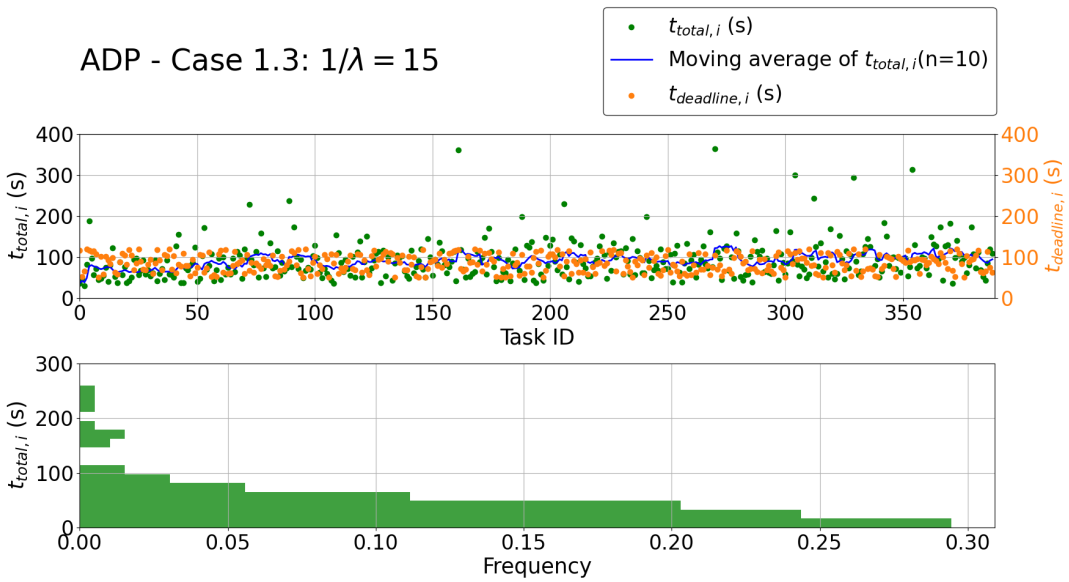


Figure 4.9. System delay and deadline distribution for the Case 1.3 using ADP.

4.4.2. Impact of Average Vehicle Speed $\overline{V}$

Vehicle mobility inherently has significant effect on our framework since it changes the network topology. During handover, all the flows related to the user should be re-configured. This function is handled by our offloading orchestrator, still there are connection and discovery based delays being observed. In the baseline case from the previous scenario, vehicles were moving at 20 km/h so for comparison we had chosen human walk speed, 5 km/h for low mobility and 40 km/h for the double speed comparison. In the high speed case, the average time the vehicle spends in the covered area has been dropped to 265 seconds per average to observe more dynamic environment.

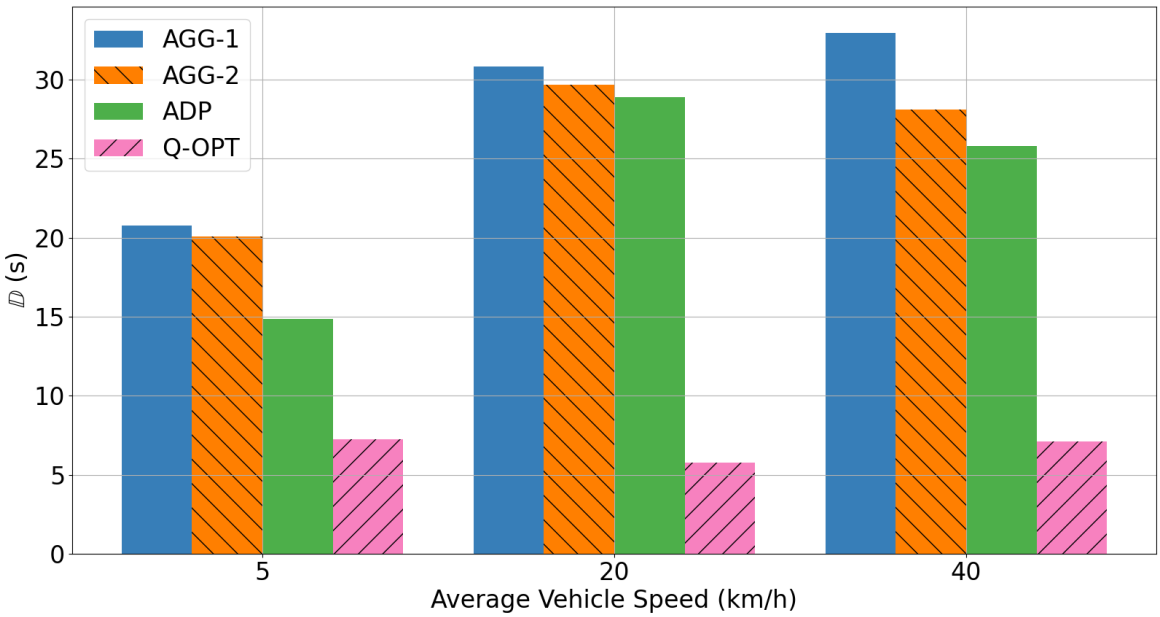


Figure 4.10. Impact of average vehicle speed on the objective function $\mathbb{D}$.

4.4.2.1. Penalty Results. From Figure 4.10, starting with the Case 2.1, where the speed is 5 km/h, our heuristic decision algorithms gave significantly better results compared to the other two cases. There are several main reasons, according to the experiment data analysis. The first is that there has been less handover inference to data transfers, and the second is that decision validity has lasted much longer; for example, task-assigned processors with good network connectivity have stayed in this

condition much longer.

In Table 4.6, we can see the service-related details. Especially, for the Case 2.1, ADP seems to be significantly efficient compared to the other heuristics and close to the quasi-optimal levels while serving more from the local resources. The task failure ratio for the Case 2.3 is high in general. It is due to a higher rate of departures since heuristics are programmed to serve each task owner. In contrast, this is not the case for Q-OPT. It can skip tasks if there is a better opportunity cost. The interesting thing is that 20% of its tasks are failed. Nevertheless, it has comparably low $\mathbb{D}$ values, due to its decision flexibility.

Table 4.6. Impact of average vehicle speed on QoS.

	Case #	$\overline{V}$	$\mathbb{D}$	Task Failure Ratio	Cloud Ratio
AGG-1	*2.1*	5	20.76	0.029	0.026
AGG-1	*2.2*	20	30.84	0.193	0.024
AGG-1	*2.3*	40	32.98	0.381	0.112
AGG-2	*2.1*	5	20.07	0.026	0.018
AGG-2	*2.2*	20	29.67	0.201	0.029
AGG-2	*2.3*	40	28.09	0.348	0.095
ADP	*2.1*	5	14.87	0.013	0.074
ADP	*2.2*	20	28.9	0.19	0.123
ADP	*2.3*	40	25.79	0.307	0.222
Q-OPT	*2.1*	5	7.23	0.018	0.125
Q-OPT	*2.2*	20	5.78	0.114	0.128
Q-OPT	*2.3*	40	7.12	0.199	0.117

4.4.2.2. Task Processing Characteristics. In Table 4.7, task offloading characteristics related results are presented. Remembering that the task inter-arrival time is constant throughout the scenario, an increase in $\bar{t}_{pool}$ and cloud ratio indicates that offloading orchestrator has issues serving incoming requests.

Table 4.7. Impact of average vehicle speed on task processing characteristics.

	Case #	$\overline{V}$	Task Failure Ratio	Cloud Ratio	$\bar{t}_{pool}$
AGG-1	*2.1*	5	0.013	0.074	13.66
AGG-1	*2.2*	20	0.19	0.123	19.57
AGG-1	*2.3*	40	0.307	0.222	29.17
AGG-2	*2.1*	5	0.026	0.018	6.53
AGG-2	*2.2*	20	0.201	0.029	11.6
AGG-2	*2.3*	40	0.348	0.095	17.96
ADP	*2.1*	5	0.029	0.026	6.79
ADP	*2.2*	20	0.193	0.024	11.29
ADP	*2.3*	40	0.381	0.112	18.74
Q-OPT	*2.1*	5	0.018	0.125	21.79
Q-OPT	*2.2*	20	0.114	0.128	17
Q-OPT	*2.3*	40	0.199	0.117	15.5

4.4.2.3. Temporal Results for Task Completion. In Table 4.8, delay results are given while Figure 4.11 illustrates the data in more convenient way. System delays such as transmission time do not significantly change between the cases, but we have a high level of task failures and considerable load on the controller pool. These stated facts with some run-time debugging all lead to the same result. As the vehicles get faster, handover overhead occurs due to the disconnections during network changes, increasing the transmission times. On the other hand, their lifetime in the covered area is reduced by 20%. Processors get occupied by the tasks to be failed, but they keep them busy. Upon task owner's departure, they may receive an extra penalty which increases overall $\mathbb{D}$. At the same time, this event does not affect the system delay calculations since the failed tasks are excluded from the average delay calculation.

Table 4.8. Impact of average vehicle speed on temporal results for task completion.

	Case #	$\overline{V}$	$\bar{t}_{total}$	$\bar{t}_{tx}$	$\bar{t}_{queue}$	$\bar{t}_{process}$
AGG-1	*2.1*	5	121.09	83.09	11	20.57
AGG-1	*2.1*	20	138.02	94.43	12	20.8
AGG-1	*2.3*	40	136.57	89.15	9	19.81
AGG-2	*2.1*	5	115.59	85.77	3	20.71
AGG-2	*2.1*	20	131.07	97.19	2	20.77
AGG-2	*2.3*	40	126.33	86.78	2	19.87
ADP	*2.1*	5	105.56	71.13	0	20.6
ADP	*2.1*	20	128.57	88.57	0	20.4
ADP	*2.3*	40	127.39	79.05	0	19.07
Q-OPT	*2.1*	5	83.26	41.58	0	19.88
Q-OPT	*2.1*	20	73.05	35.7	0	20.35
Q-OPT	*2.3*	40	70.54	34.55	0	20.49

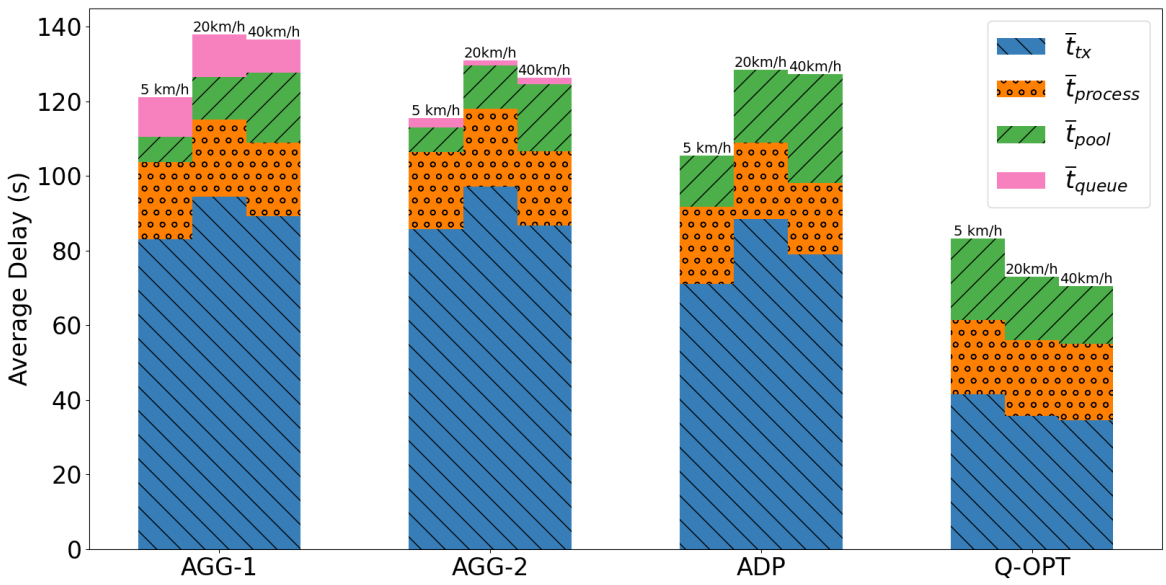


Figure 4.11. Impact of average vehicle speed on task delay distribution.

4.4.3. Impact of Processing Speed

In the previous two scenarios, we have not seen any significant effect of task processing times. The major bottleneck was data transmission, where processing was, on average, accounted only for 5% to 20% percent of the task delay.

4.4.3.1. Penalty Results. In Figure 4.12, it can be seen that processing time may significantly increase our objective function if it is set to a competitively high value with transmission; otherwise it may be negligible especially for some of our algorithms. Objective function performance of the ADP algorithm turns out be the most correlated one to the processing speed. In Table 4.9, detailed service quality results are given.

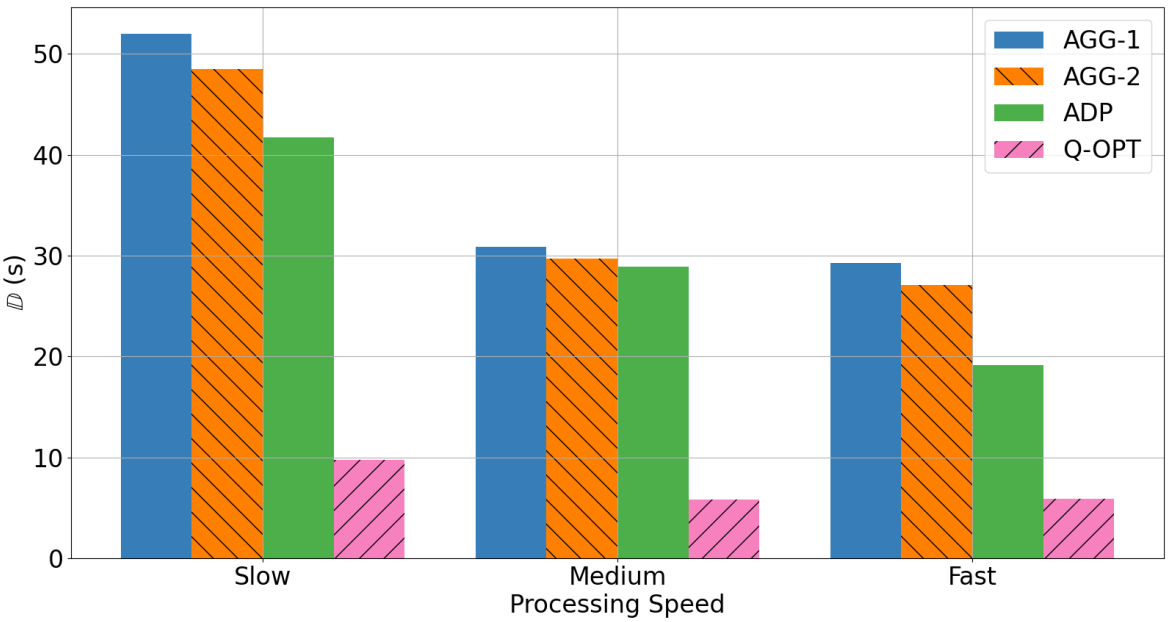


Figure 4.12. Impact of processing speed on the objective function $\mathbb{D}$.

On task failure ratios there is no noteworthy difference among the cases. For cloud offloading ratio, AGG-1 seems responsive to the changes on the process speed. That correlation is due to its logic: AGG-1 algorithm waits previously assigned other tasks to be downloaded and processed first.

Table 4.9. Impact of average processing speed on QoS.

	Case #	β_i	$\mathbb{D}$	Task Failure Ratio	Cloud Ratio
AGG-1	*3.1*	slow	52.01	0.192	0.095
AGG-1	*3.2*	medium	30.84	0.193	0.024
AGG-1	*3.3*	fast	29.25	0.183	0.04
AGG-2	*3.1*	slow	48.45	0.187	0.068
AGG-2	*3.2*	medium	29.67	0.201	0.029
AGG-2	*3.3*	fast	27.11	0.19	0.036
ADP	*3.1*	slow	41.74	0.151	0.162
ADP	*3.2*	medium	28.9	0.19	0.123
ADP	*3.3*	fast	19.17	0.129	0.111
Q-OPT	*3.1*	slow	9.77	0.089	0.113
Q-OPT	*3.2*	medium	5.78	0.114	0.128
Q-OPT	*3.3*	fast	5.9	0.081	0.117

4.4.3.2. Task Processing Characteristics. In Table 4.10, as the processing speed increases, we see very slight improvement for the heuristic algorithms. For Q-OPT, there is no correlation we could say in terms of these parameters.

4.4.3.3. Temporal Results for Task Completion. From Table 4.11, it is easy to detect direct effect of the processing speed changes on $\bar{t}_{process}$. Slow processing speed affects AGG-1's delay metrics at the worst level. Moreover, processor queue delays significantly increase for AGGs. The visualization of Table 4.11 is given in Figure 4.13.

4.5. Discussions

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

the processing speed of the task processors.

Table 4.10. Impact of average processing speed on task processing characteristics.

	Case #	β_i	Task Failure Ratio	Cloud Ratio	$\bar{t}_{pool}$
AGG-1	*3.1*	slow	0.192	0.095	18.44
AGG-1	*3.2*	medium	0.193	0.024	11.29
AGG-1	*3.3*	fast	0.183	0.04	11
AGG-2	*3.1*	slow	0.187	0.068	16.31
AGG-2	*3.2*	medium	0.201	0.029	11.6
AGG-2	*3.3*	fast	0.19	0.036	8.62
ADP	*3.1*	slow	0.151	0.162	27.01
ADP	*3.2*	medium	0.19	0.123	19.57
ADP	*3.3*	fast	0.129	0.111	15.58
Q-OPT	*3.1*	slow	0.089	0.113	14.8
Q-OPT	*3.2*	medium	0.114	0.128	17
Q-OPT	*3.3*	fast	0.081	0.117	19.66

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 bitrates and it leads to a low level of capacities. In the second scenario, we have seen 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. As our last scenario, we investigated the effect of changes in the system elements constituting the overall delay.

After investigating the performance of our algorithms, we clearly observe that ADP outperformed the other heuristic algorithms. This result relates 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, in each of the three cases, AGG-2 outperforms AGG-1, which concludes the competition among them.

Table 4.11. Impact of average processing speed on temporal results for task completion.

	Case #	β_i	$\bar{t}_{total}$	$\bar{t}_{tx}$	$\bar{t}_{queue}$	$\bar{t}_{process}$
AGG-1	*3.1*	slow	198.84	101.35	23	56.2
AGG-1	*3.2*	medium	138.02	94.43	12	20.8
AGG-1	*3.3*	fast	125.65	91.52	10	12.73
AGG-2	*3.1*	slow	187.84	97.65	15	58.47
AGG-2	*3.2*	medium	131.07	97.19	2	20.77
AGG-2	*3.3*	fast	116.17	99.23	0	8.07
ADP	*3.1*	slow	174.1	88.8	4	54.77
ADP	*3.2*	medium	128.57	88.57	0	20.4
ADP	*3.3*	fast	103.55	79.57	0	8.39
Q-OPT	*3.1*	slow	94.13	27.28	0	52.05
Q-OPT	*3.2*	normal	73.05	35.7	0	20.35
Q-OPT	*3.3*	fast	70.72	42.26	0	8.8

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 less data in real-time. Q-OPT highly utilizes cloud server processing from the beginning, and thus slowdowns when the network is congested. Also, it fails to complete 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. On the high-load cases, the algorithm also skips some tasks for the sake of better overall score.

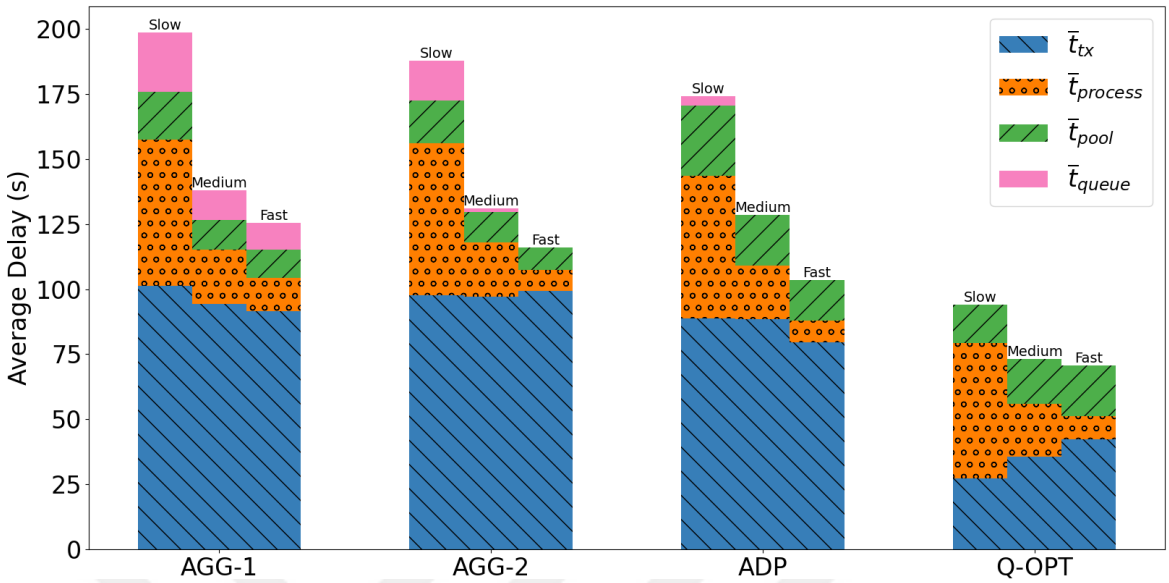


Figure 4.13. Impact of processing speed on task delay distribution.

5. CONCLUSION AND FUTURE DIRECTIONS

UAV networks are considered highly suitable as a substitute or an auxiliary fast-deployable network to provide low-latency services, baseline connectivity and facilitate a capacity boost during transient high service load situations due to their great mobility and flexibility. In this thesis, we elaborated on a critical use case for UAVs in post-disaster rescue scenarios with an emulation environment for our UAV-aided Software-Defined Network Architecture. In that environment, we considered how task offloading can be flexibly used for efficient services.

A mathematical model has been proposed to make centralized offloading decisions for incoming task processing requests. The model has focused on the centralized assignment decisions for task offloading requests from mobile ground user units to other ground vehicles or a cloud processor service. The model's objective function is to reduce average penalty scores due to job completion time, i.e., to minimize total task completion time for better QoS.

From the development side, several thesis related artifacts have been developed. 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 contribution to Mininet-Wifi and `wmediumd` has been made on a crucial feature which allows users to operate in isolated Wi-Fi mediums. Moreover, the LoS probability based channel model is implemented for physical layer simulation in addition to these outcomes.

Three heuristic algorithms and a branch&bound style quasi-optimal task offloading algorithm were developed, and their performance analysis along with the proposed network model has been done in a detailed way.

We have managed end-to-end flows to bypass mesh protocol-related issues about the ARP packets but have not used paths in a traffic-based way. For future work, the

first direction is to extend our framework for smart routing. The biggest problem we have faced in our model 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 to develop smarter heuristic approaches for task offloading. This study is focused more on analyzing the characteristics of the network with proposed heuristics.

Lastly, better management of the UAV mobility could be used to handle area-based congestions. We have observed that some of the APs are more likely to be active during our simulations. According to our analysis, it is because of the traffic model, which leads vehicles, especially to the central areas in the simulated geographical area.

REFERENCES

1. Kalınağaç, O., S. S. Kafiloğlu, F. Alagöz and G. Gür, "Caching and D2D Sharing for Content Delivery in Software-Defined UAV Networks", *2019 IEEE 90th Vehicular Technology Conference (VTC2019-Fall)*, pp. 1–5, 2019.

2. Shi, W., 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. Wang, X., 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. Zhao, L., 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. Selvi, H., G. Gür and F. Alagöz, "Cooperative Load Balancing for Hierarchical SDN Controllers", *2016 IEEE 17th International Conference on High Performance Switching and Routing (HPSR)*, pp. 100–105, 2016.

6. Zheng, K., Q. Zheng, P. Chatzimisios, W. Xiang and Y. Zhou, "Heterogeneous Vehicular Networking: A Survey on Architecture, Challenges, and Solutions", *IEEE Communications Surveys and Tutorials*, Vol. 17, pp. 2377–2396, 10 2015.

7. Alzahrani, B., O. S. Oubbati, A. Barnawi, M. Atiquzzaman and D. Alghazzawi, "UAV Assistance Paradigm: State-of-the-Art in Applications and Challenges", *Journal of Network and Computer Applications*, Vol. 166, p. 102706, 9 2020.

8. Jia, S. and L. Zhang, "Modelling Unmanned Aerial Vehicles Base Station in

Ground-to-Air Cooperative Networks", *IET Communications*, Vol. 11, pp. 1187–1194, 2017.

9. Pourbaba, P., K. B. Manosha, S. Ali and N. Rajatheva, "Full-Duplex UAV Relay Positioning for Vehicular Communications with Underlay V2V Links", *IEEE Vehicular Technology Conference*, Vol. 2019-April, pp. 1–6, 2019.

10. Fan, X., C. Huang, B. Fu, S. Wen and X. Chen, "UAV-Assisted Data Dissemination in Delay-Constrained VANETs", *Mobile Information Systems*, Vol. 2018, pp. 1–12, Oct. 2018.

11. Zhang, R., F. Zeng, X. Cheng and L. Yang, "UAV-Aided Data Dissemination Protocol with Dynamic Trajectory Scheduling in VANETs", *IEEE International Conference on Communications*, Vol. 2019-May, pp. 1–6, 2019.

12. Seçinti, G., P. B. Darian, B. Canberk and K. R. Chowdhury, "SDNs in the Sky: Robust End-to-End Connectivity for Aerial Vehicular Networks", *IEEE Communications Magazine*, Vol. 56, pp. 16–21, 1 2018.

13. Misra, S. and N. Saha, "Detour: Dynamic Task Offloading in Software-Defined Fog for IoT Applications", *IEEE Journal on Selected Areas in Communications*, Vol. 37, pp. 1159–1166, 5 2019.

14. Al-Turjman, F., M. Abujubbeh, A. Malekloo and L. Mostarda, "UAVs Assessment in Software-Defined IoT Networks: An Overview", *Computer Communications*, Vol. 150, pp. 519–536, 2020.

15. Zhang, X., H. Wang and H. Zhao, "An SDN Framework for UAV Backbone Network Towards Knowledge Centric Networking", *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pp. 456–461, 2018.

16. Alioua, A., S. M. Senouci, S. Moussaoui, H. Sedjelmaci and M. A. Messous, "Ef-

ficient Data Processing in Software-Defined UAV-Assisted Vehicular Networks: A Sequential Game Approach", *Wireless Personal Communications*, Vol. 101, pp. 2255–2286, 2018.

17. Zhu, Y., S. Wang, X. Liu, H. Tong and C. Yin, "Joint Task and Resource Allocation in SDN-Based UAV-Assisted Cellular Networks", *2020 IEEE/CIC International Conference on Communications in China, ICCC 2020*, pp. 430–435, 8 2020.

18. Chen, W., Z. Su, Q. Xu, T. H. Luan and R. Li, "VFC-Based Cooperative UAV Computation Task Offloading for Post-disaster Rescue", *IEEE INFOCOM 2020 - IEEE Conference on Computer Communications*, pp. 228–236, 2020.

19. Alioua, A., 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.

20. Feng, G., C. Wang, B. Li, H. Lv, X. Zhuang, H. Lv, H. Wang and X. Hu, "UAV-Assisted Wireless Relay Networks for Mobile Offloading and Trajectory Optimization", *Peer-to-Peer Networking and Applications*, Vol. 12, No. 6, pp. 1820–1834, Nov 2019.

21. Bouachir, O., A. Abrassart, F. Garcia and N. Larrieu, "A Mobility Model for UAV Ad Hoc Network", *ICUAS 2014, International Conference on Unmanned Aircraft Systems*, pp. 383–388, Orlando, United States, May 2014.

22. Al-Hourani, A., S. Kandeepan and S. Lardner, "Optimal LAP Altitude for Maximum Coverage", *IEEE Wireless Communications Letters*, Vol. 3, No. 6, pp. 569–572, 2014.

23. Zhang, S., H. Zhang, B. Di and L. Song, "Cellular UAV-to-X Communications: Design and Optimization for Multi-UAV Networks", *IEEE Transactions on Wireless Communications*, Vol. 18, pp. 1346–1359, 2 2019.

24. Al-Hourani, A., S. Kandeepan and S. Lardner, "Optimal LAP Altitude for Maximum Coverage", *IEEE Wireless Communications Letters*, Vol. 3, No. 6, pp. 569–572, 2014.

25. Liu, C., 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.

26. Chen, W., 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.

27. Berde, P., 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", *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*, HotSDN '14, p. 1–6, Association for Computing Machinery, New York, NY, USA, 2014.

28. "Mininet-WiFi", `github.com/onurklngc/mininet-wifi/tree/multimedium`, Accessed in January 2022.

29. "wmediumd", `github.com/onurklngc/wmediumd/tree/multimedium`, Accessed in January 2022.

30. Fontes, R. R., S. Afzal, S. H. B. Brito, M. A. S. Santos and C. E. Rothenberg, "Mininet-WiFi: Emulating Software-Defined Wireless Networks", *Network and Service Management (CNSM), 2015 11th International Conference on*, pp. 384–389, Nov 2015.

31. Pfaff, B., 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", *NSDI*, 2015.

32. National Laboratory for Applied Network Research (NLANR), "IPerf Performance Measurement Tool", `github.com/esnet/iperf`, Accessed in January 2022.

33. Lopez, P. A., 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", *The 21st IEEE International Conference on Intelligent Transportation Systems*, IEEE, 2018.

34. Lantz, B., B. Heller and N. McKeown, "A Network in a Laptop: Rapid Prototyping for Software-Defined Networks", *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, Association for Computing Machinery, New York, NY, USA, 2010.

35. Consortium, I. E., *Broadband Wireless and WiMAX*, Comprehensive Report, International Engineering Consortium, 2005.

36. Moreira, M. J. M., *yawmd: Multiple Medium Support and Performance Improvements for wmediumd*, Master's Thesis, University of Porto, 2020.

37. Carrano, R. C., C. S. Magalhães, D. C. M. Saade and C. V. N. Albuquerque, "IEEE 802.11s Multihop MAC: A Tutorial", *IEEE Communications Surveys&Tutorials*, Vol. 13, 2011.