Software-defined Network Based Resource Allocation in Distributed Servers for Unmanned Aerial Vehicles

Raj Mani Shukla*, Shamik Sengupta†, and Amar Nath Patra‡
Department of Computer Science and Engineering
University of Nevada, Reno, USA, 89557
Email: *rshukla@unr.edu, †ssengupta@unr.edu, ‡apatra@nevada.unr.edu

***Abstract*—Unmanned Aerial Vehicles (UAVs) are gaining a lot of popularity among research communities as well as various service providers. The UAVs deployed in mission-critical applications need to perform tasks with very less delay. They also need to conserve their on-board battery power so that their flight time can be increased. To accomplish both of these purposes UAVs may offload their computation and energy intensive jobs to some remote place like cloud servers, edge servers or another UAV node having abundant computing and energy resources. This paper proposes a Software Defined Networking (SDN) architecture to provision hybrid computing resources in an efficient manner. The proposed SDN architecture uses a greedy algorithm to satisfy Quality of Service (QoS) requirements of UAV's applications. The proposed method further leverages a simulated annealing based approach to minimize average latency of the applications and average energy consumption of UAV nodes. Simulation results illustrate a substantial improvement in average latency and energy consumption using proposed mechanism in contrast to processing applications on UAV node itself or offloading it only to the cloud.**

***Index Terms*—Unmanned aerial vehicles, Software defined network, computation offloading, cloud computing, Edge computing**

I. Introduction

Advances in communication technology have enabled the heterogeneous UAVs to connect with each other through multiple radio access technologies. These UAVs need to execute applications (like video processing) for various inspection and surveillance operations in different fields, e.g., agriculture, industries or smart cities. UAVs are also useful for various mission critical applications such as military or disaster management. For these applications, they need to sense some data, process it and take action on processed data very swiftly.

Since UAVs are mobile, they depend solely on their on-board battery power for performing various operations like flight control, sensing/transmission of data or running some applications. Besides, executing any computation intensive application while UAV is operating also consumes ample on-board battery power. Therefore, flight time or hovering time of a UAV is primarily dependent on its remaining battery level and applications that are being executed. It is not always possible to re-charge UAVs frequently, and thus energy management is an utmost crucial issue for their reliable operation in mission critical applications. The on-board processor in a UAV may also not have enough computing resources due to the small size of UAV. This puts a restraint on efficient execution of complex applications. The battery and computation performance of UAVs can be virtually enhanced leveraging computation offloading [1] approach. *Computation offloading* is sending energy demanding computation to a remote device and getting results back from that device. It serves two important purposes: 1) Reduction in application latency (execution time or round-trip time), if remote device has enormous computing resources 2) Improving battery performance because application is being executed at a remote device. A UAV node running out of power and requiring some application to run can offload its application to any nearby UAV node which has enough battery power and computing resources. Since UAVs are connected to the internet, they can also use cloud services like Amazon Web Service (AWS) to execute their application. They can also harness edge computing, which is an extension of cloud computing technology that uses end-user clients or devices at the edge of a network to carry out a substantial amount of computation and storage of data [2].

Each of these technologies have their inherent advantages and disadvantages. For example, while cloud computing can be helpful for long-term analysis or storage of data, edge computing is beneficial for the real-time analytics of transient data that require very low latency. Cloud may be having massive computing resources, but they are also generally located far away from the user. Therefore cloud services may not always be accessed in remote places (where UAVs are supposed to perform their operations). On the other hand, edge servers may not have enough resources, but they are located near the UAV and therefore can assist UAVs if cloud access is not possible. Therefore a UAV node has an option to either process application using its resources, collaborate with the other nearby UAV nodes or send its computation to edge server or a remote cloud for processing according to applications' quality of service requirements. Thus the location at which a UAV application is to be processed (on-board, edge server or cloud data center) is an important research problem which needs to be explored in detail.

Since UAVs, edge servers and cloud servers are spatially located at a large geographical distance the network management between these units is a challenging problem. Fortunately, the advances in Software-Defined Network (SDN) have enabled global view of network due to their centralized architecture. Due to its simplified centralized architecture, we propose SDN based approach to decide the place at which an application should be executed. The proposed approach minimizes operating delay and energy consumption while meeting applications' Quality of Service (QoS) requirements. Benefits of SDN-based approach is due to a large number of interconnected UAVs, coexistence of multiple radio access technology, dynamics of networking environment and varied quality of service requirement of applications. The SDN-based architecture provides flexibility, scalability, programmability and global view of UAV design thus avoiding the decision making at a single UAV node which otherwise may further exacerbate the on-board battery performance of UAVs. The main contributions of this work are as follows:

- The proposal of a novel SDN architecture which accepts the application processing requests from energy starved UAVs, requiring to process applications with low latency.
- The proposal of a greedy algorithm, running on SDN controller, which ensures that the QoS requirement of requested applications are met by assigning a set of servers/UAV nodes where applications can be offloaded.
- Proposal of a simulated annealing algorithm, which also runs on SDN controller, to optimize requested applications' average latency and energy consumption of UAV.

The rest of this paper is organized as follow. In section II, a brief literature survey has been described. Section III presents the proposed SDN architecture. In section IV we describe the system model and problem statement in detail. In section V simulation experiments and results are presented and in section VI we conclude this paper.

II. Literature Survey

There has been work done in the literature on energy management in UAV systems. In [3] Chmaj et al. has proposed collaborative data processing in UAVs to save their energy. Chmaz et al. has performed a set of experimentation on UAVs which were equipped with ARM processors. Though in this approach, the Chmaz et al. has considered cooperation between UAV nodes for data processing, use of state of the art technology edge computing and cloud computing, which are widely used to provide computing as a service to the user were not a part of this work. In [4] Mahmoud et al. has shown how the UAVs can be integrated with the cloud. The work has developed a client-server based web architecture where UAV resources and services can be accessed and monitored remotely. Although the attempt to efficiently use UAV services using cloud computing were well considered in this paper, the energy and latency improvement through remote computation offloading was not considered in this work.

Quereshi et al. has demonstrated the use of cloud computing for efficient processing of computation-intensive applications in UAVs in [5]. It was shown through testbed implementation that using cloud computing improves battery and performance

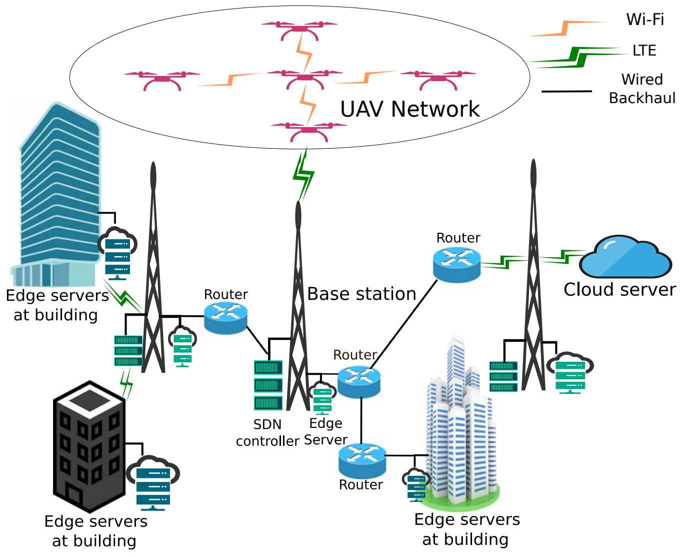


Fig. 1: An example system architecture

in UAVs when applications like image processing are offloaded to the cloud. Although cloud can be used to provide computing as a service, they have the inherent disadvantage for latency sensitive applications since clouds are typically located far away from the user. For these applications, the concept of edge computing was introduced in [2] by Patel et al. In [6] Motlagh et al. has described the use of UAVs and edge computing paradigm for the UAV surveillance operation.

Although, the aforementioned works have shown that the cloud/edge assist UAVs to perform their tasks efficiently, they have not considered which computing resources to be provisioned to UAVs from available pool of resources, for application processing, such that their latency and energy performance are optimized. In this paper, we propose a hybrid approach where SDN controller allocates computing resources to UAVs such that the QoS requirements of applications are met and average latency and energy performance of UAVs are optimized. With the best of our knowledge, for the first time, we have considered a hybrid approach utilizing cloud, edge, and collaboration among UAV nodes together for efficient application processing in UAV networks.

III. Proposed SDN architecture

A. System model

Figure 1 presents an example system model for the proposed research. We consider heterogeneous UAV nodes (U_i) hovering over a geographical area and controlled by a nearby base station [7]. The UAV nodes are heterogeneous regarding their computing capability and on-board battery power. The nodes are running different applications, and therefore they require different computing and battery power. UAV control, edge server, and SDN services are hosted in the base station over a set of computing clusters. As shown in figure 1, edge servers can also be located at various nearby places like university campus or buildings [8]. Although edge is designed to feed the local needs [9], they can be used for providing services to UAVs, if they are idle or not used to their full capacity, based on some agreement policy determined by edge and UAV service providers. Clouds in general are located far away from the UAV clusters. Virtual machines (VMs) are hosted in edge and cloud to process requested UAV applications.

Edge and cloud can be connected to the base station through LTE link or through the wired backhaul. UAVs send their application processing request, the QoS requirements and application parameters (like data transfer size) to the SDN controller located at the nearby base station through an LTE connection. Cloud, edge and UAV nodes broadcast their processing rate, tasks arrival rate and their willingness to process the request to the SDN controller located at base station. Based on these parameters and applications' QoS requirements, the SDN controller at the base station assigns the servers for requested applications. If SDN controller assigns another UAV node for some application, then two UAVs (client and server) directly communicate to each other (can be through Wi-Fi or similar connections). On the other hand, if it assigns some cloud or edge to an application then UAV node sends offloaded data to the base station through the LTE link. Base station transfers that data to cloud or edge through wired backhaul. Cloud or edge processes the data, and the processed data traverses back to the base station through wired backhaul. While data is traveling through routers and processed at remote server, UAV receiver's circuitry is in low power sleep mode and senses/probes for the processed data. Subsequently, base station transfers the processed data to the UAV node through LTE link.

B. Assumptions

- UAV's position w.r.t the base station remains constant while requested applications are being processed.
- Application requests coming from various UAV nodes are independent of each other.
- Every task is hosted on a single server (edge or cloud) or UAV node. If a task comprises of many sub-tasks, then without the loss of generality, these sub-tasks can be considered as separate individual tasks.

C. Software Architecture

Figure 2 presents the proposed software architecture which consist of three different modules hosted individually at UAV node, at SDN controller and at edge or cloud servers to facilitate efficient distributed resource allocation to a UAV. The connection between these three modules is enabled by SDN infrastructure layer. The software module at UAV node contains energy analysis, program partitioning and application core units. The energy analysis unit determines the available energy and future energy estimate to find if an application need to be offloaded to the remote server for processing. It also determines whether UAV can serve offloaded application from another UAV node depending upon its available energy. Program partitioning module partitions the application into

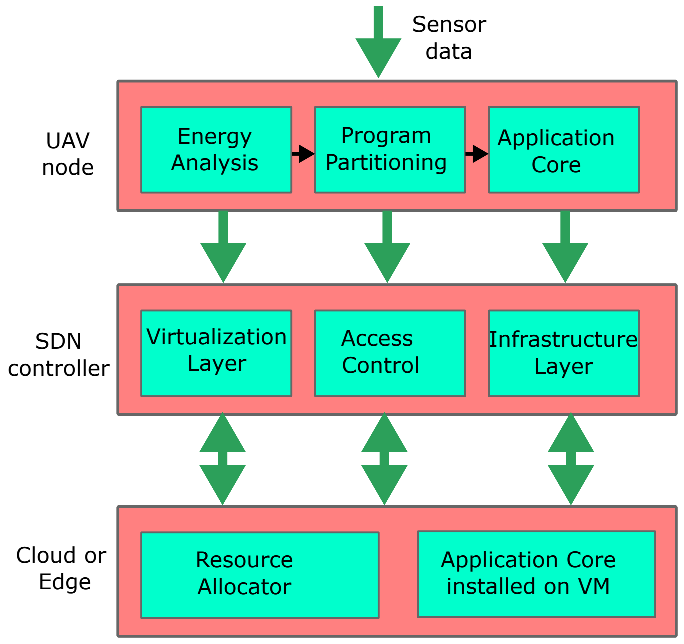


Fig. 2: Proposed software architecture

offloadable and non-offloadable segments. Offloadable segments run on remote servers while non-offloadable segment executes on UAV node itself. A program can be partitioned into multiple offloadable segments. Application core in the UAV node includes set of installed programs required for processing sensor data.

The SDN module virtualizes the UAV node, edge servers, and cloud servers to combine them into a single framework. SDN offers flexibility in where the framework performs the application processing: in UAV node itself, in another nearby UAV, at edge server, or at a cloud server. The SDN obtains this flexibility by interaction with various units which are spatially located at a vast geographical distance. The SDN module receives the application processing requests from UAVs. Based on the received requests it allocates virtual resources to the UAVs where an application is to be processed efficiently. Access control ensures that the SDN directs the requests only to the authentic cloud or edge servers. Infrastructure layer guarantees the coordination between various layers. It also implements optimized routing between UAV node to end points which include edge or cloud servers.

Resource allocator module at edge or cloud servers keeps track of available computing resources which include CPUs at the server. The module allocates the VM to the application processing request. The resource allocator also maps the virtual resources (VM) to physical resources. The module finds the placement of VM to a given application processing request. The mapping determines specific CPU resource where VM is to be allocated for processing application request. The module also contains the copy of the application core softwares which needs to be processed. These softwares are installed in the VM and process the application based on the received data from

UAV.

IV. Proposed Methodology

The proposed method to assign a server to every requested application consists of two distinct phases. In the first phase, we select the list of potential servers where an application service can be hosted to meet its' QoS requirements. Based on the results of the first phase, an optimum server for each request is chosen to minimize average application latency and energy consumption. Before proceeding on algorithms' description we begin by defining the problem statement.

A. Problem Statement

SDN controller accepts the request for processing applications which are denoted as $\{A_1, A_2, ...A_n\}$. Each request A_i has an associated vector V_i^{app} which determines the parameters associated with the application. These parameters include data send size d_{sen}^i, data received size d_{rec}^i, maximum tolerable latency l_{max}^i, and maximum tolerable energy e_{max}^i for offloading application. UAV nodes, edge server and cloud are denoted as U_j, E_j and C_j respectively. The edge and cloud servers receive requests not only from UAV nodes but other sources as well. For instance, cloud also receives application processing requests from mobile users. These requests from different sources can be considered as independent of each other. Therefore, requests arriving at a server can be modeled as a M/M/1 queue. Tasks are coming in servers at the rate $aRate_j$, and the rate at which they are processed is given as $pRate_j$. *Application latency* is given as the time delay between a request for application processing is made and the completion of application processing. For on-board processing, application latency is given as the execution time (t_{ex}^i) for i_{th} application. For offloaded applications, latency is given by equation 1, where L_{app}^i is the latency for i_{th} application, t_o^i is the overhead time to share the information to the SDN controller, t_{sen}^i is the time taken to send data from client to server, $t_{ex,r}^{i,j}$ is the execution time of the application at the remote server j and t_{rec}^i is the communication time to receive the results from the server.

$$L_{app}^i = t_o^i + t_{sen}^i + t_{ex,r}^{i,j} + t_{rec}^i \tag{1}$$

On board energy consumption is estimated to be $P_{ob}^i * t_{ex}^{i,j}$, where P_{ob}^j is the power supplied to the CPU and $t_{ex}^{i,j}$ is the execution time of the i_{th} application in j_{th} UAV. For offloaded application i, the energy consumption estimates are according to the following equation:

$$E_{app}^{off,i} = P_{sen}^j * t_{sen}^i + P_s^j * t_s^i + P_{rec}^j * t_{rec}^i \tag{2}$$

where, P_{sen}^j, P_s^j and P_{rec}^j are send, sense and received power requirement of UAV j. t_{sen}^i, t_s^i and t_{rec}^i are the time to send, sense and receive the data for application i from base station. Each router is assumed to have average bandwidth capacity of b_{avg}. The data to be send and received for offloading application i is d_{sen}^i and d_{rec}^i. The estimated time to transfer data for processing application i between two routers is given as $\frac{d_{sen}^i}{b_{avg}}$ and if number of hops between client and server are $nH_{i,j}$ then the estimated time taken to transfer file from a client to server is given as $\frac{nH_{i,j} * d_{sen}^i}{b_{avg}}$. Since maximum tolerable latency and energy consumption for application A_i are l_i^{max} and e_i^{max} equation 3 and 4 must hold to be true for applications' QoS requirement.

$$l_i^{max} > \frac{nH_{i,j} * (d_{sen}^i + d_{rec}^i)}{b_{avg}} + \frac{1}{pRate_j - aRate_j} + t_o^i \tag{3}$$

$$e_i^{max} > \frac{nH_{i,j} * d_{sen}^i}{b_{avg}} * P_{sen}^j + \frac{nH_{i,j} * d_{rec}^i}{b_{avg}} * P_{rec}^j + t_s^i * P_s^j \tag{4}$$

Therefore, the maximum hop distance at which a server is located to meet latency requirement of application i is given by equation 5

$$nH_{max}^i = min(l_i^{max} - \frac{1}{pRate_j - aRatej} \times \frac{b_{avg}}{(d_{sen}^i + d_{rec}^i)} - t_o^i, \\ (e_i^{max} - t_s^i * P_s^j) * \frac{b_{avg}}{d_{sen}^i + d_{rec}^i}) \tag{5}$$

The objective function to minimize is given by equation 6 and 7 subject to the conditions given by equations 3 and 4,

$$min\{\sum_{i=1}^{N} L_{app}^i\} \tag{6}$$

$$min\{\sum_{i=1}^{N} E_{app}^{off,i}\} \tag{7}$$

where summation is done over all requested applications $i \in 1, 2.....N$.

B. Partial matrix allocation

The problem of assigning servers (or UAV node) to minimize average latency and energy consumption is solved using a two-phase algorithm. We have considered an $N \times M$ matrix, where N represents the number of servers, and M denotes the number of requested applications. In phase I (algorithm 1), a set of servers for all applications are assigned such that the QoS requirement of the applications are met. This step involves the allocation of a set of matrix cells for each application. This phase guarantees that the specified performance criterion of each application is met. The results of the first phase are used in Phase II where algorithm 2 assigns an optimum server for each application such that the average latency and energy consumption is minimum. In this step, a particular matrix cell among assigned cells in phase I are selected for each application. Thus, while phase I ensures that the QoS requirements are met, phase II optimizes the average latency and energy consumption.

Figure 3 demonstrates an illustrative example of two-phase matrix allocation problem. Initial stage shows the pre-allocation phase. Red cells represent unallocated servers. In phase I, the marked cells denote the allocated servers which

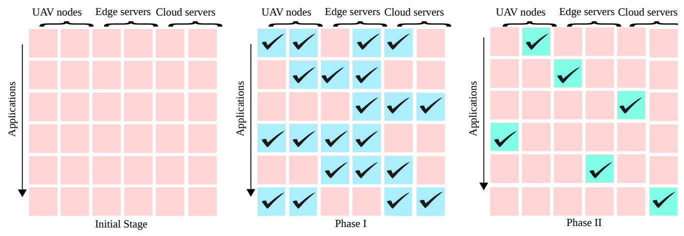


Fig. 3: Description of algorithm phases

Algorithm 1 Optimum server set for application processing using greedy algorithm

```
Input: $N_u$ - UAV nodes, $E_i$ - Edge servers, $C_i$ - cloud servers
    $nH_i$ - Hop distance of server $i$ from UAV set
    $aRate_i$ - Task arrival rate at server $i$
    $pRate_i$ - Task processing rate at server $i$
Output: $S_i^{app}$ - set of servers for application $i$
 1: $N = N_u + E_i + C_i$
 2: for $i \leftarrow 1$ to $N_{app}$ do
 3:     for $j \leftarrow 1$ to $N$ do
 4:         $l = \frac{(d_{sen} + d_{rec}) \times nH_j}{est(BW)} + \frac{1}{pRate_j - aRate_j} + T_0$
 5:         $E_{app}^{off,i} = P_{sen} * t_{sen} + P_{rec} * t_{rec} + P_s * t_s$
 6:         if $l < l_i^{max}$ and $E_{app}^{off,i} < e_i^{max}$ then
 7:             if $NodeId(i) = j$ or $\zeta = 0$ then
 8:                 continue
 9:             end if
10:             Append $j$ to $S_j^{app}$
11:         end if
12:     end for
13: end for
```

Algorithm 2 Optimal Sever Selection using Simulated Annealing

```
Input: $S_i^{app}$ - List of servers
Output: $M_a$ - Sever allocation matrix
 1: $T = T_{high}$
 2: $\alpha = 0.44$
 3: for $i \leftarrow 1$ to $N$ do
 4:     Assign a random server $S_i$ to application $A_i$ for matrix $M_a$
 5: end for
 6: $S_{opt} = M_a$
 7: Find average laletncy $L_{avg}$ and energy consumption $E_{avg}$ of all applications
 8: $L_{min} = L_{avg}$
 9: $E_{min} = E_{avg}$
10: repeat
11:     for $iter \leftarrow 1$ to $maxIter$ do
12:         Select a random application $A_i$
13:         Switch assigned server for $A_i$ to some other random server
14:         Find average laletncy $L_{avg}$ and energy consumption $E_{avg}$ of all applications
15:         $P = \frac{1}{1 + e^{\frac{-L_{min}}{T}}}$
16:         if $L_{avg} < L_{min}$ and $E_{avg} < E_{min}$ then
17:             $S_{opt} = M_a$
18:             $L_{min} = L_{avg}$
19:             $E_{min} = E_{avg}$
20:         else
21:             if $rand(0,1) < P$ then
22:                 $S_{opt} = M_a$
23:                 $L_{min} = L_{avg}$
24:                 $E_{min} = E_{avg}$
25:             end if
26:         end if
27:     end for
28:     $T = T \times \alpha$
29: until $T > T_{min}$
```

meet the QoS requirement of the applications. Phase II represents the final mapped matrix where marked cells denote the assigned servers for applications.

1) Phase I: Meeting QoS requirements: Algorithm 1 presents the steps involved in finding a set of possible servers to meet applications' QoS requirements. Since requests arriving at a particular server are independent of each other, the task arrival rate is considered to follow a Poisson distribution with a mean value of $pRate$. For each requested application, algorithm determines the latency and energy consumption based on the equation 1 and 2 for each server. If latency and energy consumption satisfies the performance requirement of the application, then the server is appended to the list S_j^{app} (blue cells in the figure 3) for application A_j. However, if selected server comes out to have same $NodeId$ which has requested the application processing, then that server is skipped. Selected server is also checked for its availability which is determined by a factor ζ.

2) Phase II: Latency and energy optimization: Algorithm 2 depicts the simulated annealing algorithm to minimize average latency and energy consumption. The algorithm takes the list of servers S_i^{app} for every application i obtained from phase I. In step 1 of the algorithm, the annealing temperature is set to a very high value (T_{high}) [10]. The cooling factor α ($0 < \alpha < 1$) which determines the number of outer loops (Step 28) for an optimized performance of algorithm [10] is set in step 2. A low value of α will result in convergence of the algorithm very quickly, but we may not reach an optimal solution. A high α value results in an optimum solution, but it will take a large number of outer loops. Since high α value (close to 1) gives optimal solution, we gradually changed α from a very low value to a high value and observed the value of α after which optimum solution given by the algorithm is equal to the one at high α value, and is constant from that value to all higher values. The lowest α value was found to be 0.90, which we have used that in our algorithm since it gives us an optimal solution with minimum number of iterations.

In steps 3-5 a random server is assigned for all applications i from its corresponding server list S_j^{app} (which is obtained

TABLE I: Simulation parameters for servers

	UAV node	Edge Server	Cloud Server
Hop distance	1	1-20	10-30
Task arrival rate per second	0-2	5-10	10-25
Task processing rate per second	0-5	30-50	200-300

TABLE II: Simulation parameters

Parameters	Value
Power supplied to CPU (mVA)	20-580
Transmission power (mVA)	80-140
Reception power (mVA)	2-5
Sense/probe power (mVA)	0.8-1.2
Wi-fi bandwidth (mbps)	54
LTE bandwidth (mbps)	100
Task size (MB)	0.001 - 20

from algorithm 1). The optimum server matrix, minimum average latency, and energy obtained are given as M_a, L_{min}, and E_{min} (Steps 6-9). Steps 10 - 29 describes the annealing process which is repeated several times to get minimum average latency and energy. After every iteration, the annealing temperature is reduced by a cooling factor α (Step 28). Within each iteration, a random server schedule is obtained (Steps 12 and 13). The average latency L_{avg} and energy E_{avg} are obtained for new server schedule. If average latency and energy are less than the minimum latency obtained so far then that state M_a with latency L_{min} and energy E_{min} is accepted as the optimal solution (Steps 16-19). Otherwise, state M_a is accepted with a probability P (Steps 21-24).

V. Simulation and results

We have conducted an extensive set of experiments in a python based framework to evaluate the improvement in application latency and energy performance of UAV nodes. We have compared (1) proposed mechanism with (2) only-cloud offloading and (3) only-on-board processing.

A. Simulation parameters

Table I presents the simulation parameters of the server. For individual nodes or servers, the parameters are randomly chosen from the specified range. Table II shows other parameters which are used in the experiment. We have considered topologies with a different number of UAV nodes and application requests. We have run the simulation for single-node and multiple-node offloading scenario. For single-node offloading case, only one UAV node sends application processing request to SDN controller. For multiple-node offloading, more than one UAV nodes can send requests to SDN controller to process their applications.

B. Single-node offloading

We run our experiment with the varied size of data transfer size and on-board processing rate. We have considered the scenario where different applications may have different size of data to be transferred for offloading, but their on-board execution time may be same. The applications are of similar or different kinds. For example, one application can be a video processing while another an image processing, both having equal execution time but their data transfer size are different.

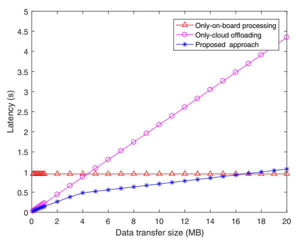
Figure 4a - 4d presents the latency and energy performance for the scenario where a single UAV node is requesting application processing from SDN controller. In figure 4a and 4b, we varied the size of offloaded data while on-board processing rate is kept constant and observed its' effect on latency and energy performance. Figures show that proposed method achieves better performance as compared to only-on-board processing or only-cloud offloading. For a particular on-board processing time and energy consumption there exists a threshold for the size of data transfer after which offloading doesn't have advantage regarding latency or energy improvement. As it can be seen from the figures, value of the threshold data transfer size is higher for the proposed method as compared to only-cloud offloading.

In figure 4c and 4d, the on-board processing rate is varied while keeping size of the offloaded data to be constant. The results reveal that proposed method achieves better latency and energy performance as compared to the contemporary only-cloud offloading. Further, there exists a threshold for data transfer size and on-board processing rate. If data transfer size and on-board processing rate are higher than the threshold then offloading doesn't have any advantage regarding latency or energy consumption. For the proposed method threshold values of data transfer size and processing rate are higher as compared to only-cloud offloading.

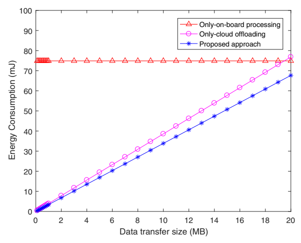
C. Multiple-node offloading

Figure 5a and 5b present the case where applications are offloaded from varied number of UAV nodes. We have considered ten applications and the number of UAV nodes are varied between 5 - 200. UAV nodes are closer to each other, therefore they involve lower latency due to single hop communication. When the number of UAV nodes are less, the latency is higher because all applications are offloaded either to the cloud or edge. As the number of UAV nodes are increased latency and energy consumption reduces because some applications are also offloaded to a nearby UAV node. On further increasing the number of UAV nodes, both latency and energy consumption becomes constant since enough UAVs are available for processing some applications. The proposed method achieves better performance over only-cloud or only-on-board processing.

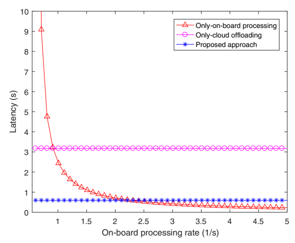
Figure 5c and 5d represent the mean latency and energy consumption as the number of edge servers are varied. We have chosen the number of UAV nodes as 5. When the number of edge servers are less, then only-cloud offloading and the proposed approach gives comparable performance. Both latency and energy consumption decreases on increasing number of edge servers since some of the applications that were offloaded to the cloud are now being offloaded to the edge servers. Latency and energy consumption saturates on further increasing edge servers since enough number of servers are available for optimally processing requested applications.



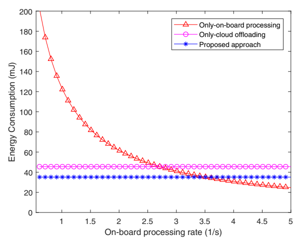
(a) Effect of data transfer size on latency with constant on-board processing rate



(b) Effect of data transfer size on energy consumption with constant on-board processing rate

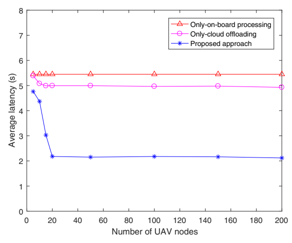


(c) Effect of processing rate on latency with constant data transfer size

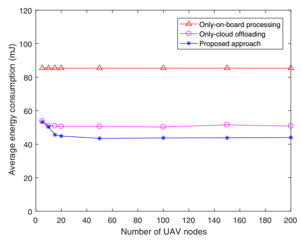


(d) Effect of processing rate on energy consumption with constant data transfer size

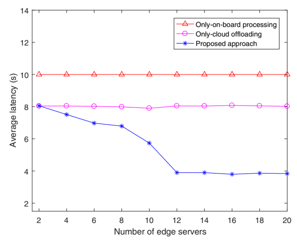
Fig. 4: Single node offloading



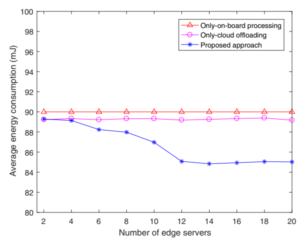
(a) Effect of number of UAV nodes on average latency with constant number of edge and cloud servers



(b) Effect of number of UAV nodes on average energy consumption with constant number of edge and cloud servers



(c) Effect of number of edge servers on average latency with constant number of UAV nodes and cloud servers



(d) Effect of number of edge servers on average energy consumption with constant number of UAV nodes and cloud servers

Fig. 5: Multiple-node offloading

VI. Conclusions

Quick processing of application and energy management of UAV nodes is a critical issue for UAVs operating in mission critical applications. State of the art edge and cloud computing technologies have been used for improving application latency and energy performance. In this paper, we have used UAV nodes' collaboration, edge computing, and cloud computing together in an SDN-based framework. Simulation results show that the proposed method achieves improved performance over only-cloud processing or only-on-board processing. Further, there exists a threshold regarding data transfer size and on-board processing rate above which offloading is not beneficial.

Acknowledgement

This research is supported by NSF CAREER grant CNS #1346600.

References

[1] K. Kumar and Y. H. Lu, "Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?" *Journal of Computer*, vol. 43, no. 4, pp. 51–56, April 2010.

[2] M. Patel, Y. Hu, P. Hédé, J. Joubert, C. Thornton, B. Naughton, J. R. Ramos, C. Chan, V. Young, S. J. Tan, D. Lynch, N. Sprecher, T. Musiol, C. Manzanares, U. Rauschenbach, S. Abeta, L. Chen, K. Shimizu, A. Neal, P. Cosimini, A. Pollard, and G. Klas, "Mobile-Edge Computing," in *ETSI White Paper*, September 2014. [Online]. Available: https://portal.etsi.org/portals/0/tbpages/mec/docs/mobile-edge_computing_-_introductory_technical_white_paper_v1%2018-09-14.pdf

[3] G. Chmaj and H. Selvaraj, "Uav cooperative data processing using distributed computing platform," in *Progress in Systems Engineering*. Springer, 2015, pp. 455–461.

[4] S. Mahmoud, N. Mohamed, and J. Al-Jaroodi, "Integrating uavs into the cloud using the concept of the web of things," *Journal of Robotics*, vol. 2015, pp. 10:10–10:10, Jan. 2015.

[5] B. Qureshi, Y. Javed, A. Koubâa, M.-F. Sriti, and M. Alajlan, "Performance of a low cost hadoop cluster for image analysis in cloud robotics environment," *Procedia Computer Science*, vol. 82, pp. 90–98, 2016.

[6] N. H. Motlagh, M. Bagaa, and T. Taleb, "Uav-based iot platform: A crowd surveillance use case," *IEEE Communications Magazine*, vol. 55, no. 2, pp. 128–134, 2017.

[7] A. Bürkle, F. Segor, and M. Kollmann, "Towards autonomous micro uav swarms," *Journal of intelligent & robotic systems*, vol. 61, no. 1-4, pp. 339–353, 2011.

[8] A. Ahmed and E. Ahmed, "A survey on mobile edge computing," in *Intelligent Systems and Control (ISCO), 2016 10th International Conference on*. IEEE, 2016, pp. 1–8.

[9] M. T. Beck, M. Werner, S. Feld, and S. Schimper, "Mobile edge computing: A taxonomy," in *Proc. of the Sixth International Conference on Advances in Future Internet*. Citeseer, 2014.

[10] [Online]. Available: https://en.wikipedia.org/wiki/Simulated_annealing