Key Points for the algorithm to make it robust:

INITIAL ALGO AND LOGICS:
Def Algo(workload, latency, RAN_loc):

Latency:
T = d/v
3* 10^8
0.01ms - 0.03ms - (double)
D = 3km - 9km - (int)

Workload:
140 - 150 task - (int)
Each RAN - Task limit - 2
MEC Task limit = Workload_cap
Max number in the cluster can be = Wokload_cap/RAN_cap

RAN's = 1. Max_cap_RAN(global var) - 5 - 10
2. Instant_Workload_RAN(generated in dataset)
MEC's = 1. Max_cap_MEC (global var) - 150
2. Instant_Workload_MEC(calculate and apply in logic)

Added_RAN = [ ]
Remaining_RAN = [ ]
Cooking_Cluster = [(coordinate, no. of RAN,)]

K = 5;
k means - static
K - 4

Dynamically Optimal K-Means

Link for further clarification: Step Explanation

---

1) The cluster's numbers should be hardcoded. And there is no case of soft clustering. The RANs will be associated with 1 and only 1 MEC. Also no declaration of noise.
Algorithm:
r^- = RANs not in any cluster
r^- = {Φ}
Logic:
Distance-based clustering.
Haversine distance to be considered.
A clustering algo to be implemented

2) Data set contains the RAN's location and each RAN's individual workload.
Given, Max_cap_MEC and Max_cap_RAN values, the max number of RANs in the cluster is given by:
Algorithm:
i) ∑ instant_workload_RAN <= Max_cap_MEC;
ii) Number of RANs in the cluster = (MAX_cap_MEC)/(Max_cap_RAN);
iii) ∑ instant_workload_RAN = Instant_MEC_workload;
Logic:
i) instant_workload_RAN is the workload of an individual RAN at a particular instant and that can keep varying.
instant_workload_RAN < Max_cap_RAN
ii) In a cluster the number of RANs are only calculated assuming, both MEC and RAN are working at their full capacity ---> Max_workload
iii) Once we know the number of RANs, we can easily sum the instant_workload_RAN to find the instant_Workload_MEC at that point.
--

```
Num_RAN = (MAX_cap_MEC)/(Max_cap_RAN) // to find the number of RANs in cluster
Int sum_workload_RAN = 0;
for(int i=1; i<=Num_RAN; i++){
        sum_workload_RAN += instant_workload_RAN; //
        }
Instant_MEC_workload = sum_workload_RAN;
```

---

3) **Outliers(Budget constraint)** There will be the presence of outliers in the dataset, hence our algo will have to compromise on either latency or workload to include them due to budget limits. For these points, workload and latency constraints will be surpassed. They will be given the connection to the nearest MEC server. Latency and workload

constraints need to be relaxed.
Algorithm:

```
 o   = Outlier //subset of remaining r^-
r^+ = RANs already in a cluster
for(int i = 0; len(o) == 0; i++)
        for(int j = 0; i < len(cooked_cluster); j++)
                If (args min oi.distance(mj) )
                        mj.add(oi)
        { neglect the latency and workload for o;
```

4) If there is a target that is moving, then the signal will be bounced. For us to be really good on the latency, the compromise has to be done on Workload. For this to be done, we need to make the user get connected to the nearest MEC and break the constraint budget in order to be fulfilled.
   Latency should not be compromised at any cost.

To achieve the above task, the concept of task scheduling is used. For achieving this, we make false workload constraints and keep some workload balance in reserve, to accommodate such situations.

The summation of workload_RAN for each cluster should be <=105(70% of 150 as to implement the reserve capacity) if there are remaining then they will be added to it. Make a new cluster if the limit is surpassed.

Algorithm:

Constrained_cap_MEC = 0.7 * Max_cap_MEC (0.7*150)
New_Max_cap_MEC = Constrained_cap_MEC (105)

```
Def Task_schedule(curr, workload, n,x):
        //n → reserve percentage that is being subtracted from the workload
        //We assume that curr is the current workload capacity of the MEC
        //Workload →  max workload of the MEC
        //Reserved →  The false 100% workload to follow the constraints
        //Users →  The UE already connected to the RAN
        //new_users→ If users travel from one place to another then, the positioning of
        the mass.



        Reserved = workload - n;
        Users = x;
```

```
Rem_users = 0;
for(int i=1;i<=new_users;i++){
        if(curr<=reserved && reserved<=workload){
                User++;
                Curr++;
                }
        Else if(curr > reserved){
                Reserved = reserved + n;
                }
        Else{
               Rem_users = users - i;
                Break;
                }
        }

Return (rem_users) ----------------> Create a new function for them
```

(OR)

Declaring Task_Schedule as a void function, then calling another function transfer function

5) We make MEC servers communicate with each other. Hence if there are tasks that surpass the load of 1 MEC, the mec would decide to share the task with another neighboring MEC server, that server would process the particular tasks and send it to the initial MEC and the MEC would send it to the user. In other words the MEC server gets the work done by another server but sends the processed work by itself to the user. //Threshold limit is customizable and variable input. It’s a recursive outcome.
   Algorithm:
   ```
   Def transfer_workload(Mw1, Mw2){

   }
   ```