NetSim®

Accelerate Network R & D

Dynamic Clustering in WSN

MATLAB and Python Workflows

Version 15.0

A Network Simulation & Emulation Software

By

TETCOS LLP

Contents

Software: NetSim Standard v15.0 (64 bit), Visual Studio 2022, MATLAB R2019 or later, Python 3.11 or later.

MATLAB project download link:
https://github.com/NetSim-TETCOS/Dynamic-Clustering-MATLAB-v15.0/archive/refs/heads/main.zip

Python project download link:
https://github.com/NetSim-TETCOS/Dynamic-Clustering-Python-v15.0/archive/refs/heads/main.zip

1 Introduction to Clustering

1.1 Clustering in WSN

Clustering in Wireless Sensor Networks (WSN) involves the division of a sensor group into smaller clusters. In environments with mobile sensors, static clusters are impractical. Cluster heads within each cluster are dynamically elected, and members in each cluster are identified dynamically. Consequently, the size of each cluster is not fixed and can vary based on sensor positions.

Dynamic Clustering efficiently groups sensors into clusters in real-time. There is no fixed cluster size, and sensors are divided into the required number of clusters with member assignments calculated dynamically.

1.2 Clustering using the k-means Algorithm

`Kmeans(X,k)` partitions the points in the n-by-p data matrix X into k clusters. This iterative partitioning minimizes the sum, over all clusters, of the within-cluster sums of point-to-cluster-centroid distances. Rows of X correspond to points, columns correspond to variables. kmeans returns an n-by-1 vector IDX containing the cluster indices of each point. By default, kmeans uses squared Euclidean distances. When X is a vector, kmeans treats it as an n-by-1 data matrix, regardless of its orientation.

The sensor positions and number of clusters,

X - a matrix containing the x and y coordinates of the sensors in the scenario.

k - the number of clusters. are passed to the k-means algorithm. `[IDX,C] = kmeans(X,k)`.

IDX - Contains the cluster IDs of each sensor (i.e.) the cluster to which the sensor belongs.

C - Centroids of each cluster.

1.3 Clustering using the Fuzzy C-Means Algorithm

Fuzzy c-means (FCM) is a data clustering technique in which a dataset is grouped into n clusters with every data point in the dataset belonging to every cluster to a certain degree. For example, a certain data point that lies close to the centre of a cluster will have a high degree of belonging or membership to that cluster and another data point that lies far away from the centre of a cluster will have a low degree of belonging or membership to that cluster.

1.4 Cluster Head Election Based on Distance from Centroid

After grouping the sensors into different clusters, the cluster heads are determined based on the distance between the sensor and the centroid of the cluster to which it belongs.

The sensor which is closer to the centroid will be elected as the cluster head. Here the position values (i.e., the value of the x-coordinate and y-coordinate) of each sensor are passed from NetSim to MATLAB as a sole parameter.

1.5 Cluster Head Election Based on Distance and Power

After grouping the sensors into different clusters, the cluster heads are determined based on the distance between the sensor and the remaining power of each sensor. After that, the sensors are assigned to the respective cluster.

The sensor which is closer to the centroid and has more power than other sensors will be elected as the cluster head. Here the position values (i.e., the value of the x-coordinate and y-coordinate) of each sensor and power are passed from NetSim to MATLAB as a sole parameter.

2 MATLAB Workflow

2.1 Dynamic Clustering in NetSim with MATLAB Interfacing

Dynamic Clustering is implemented in NetSim by Interfacing with MATLAB for the purpose of mathematical calculation. The sensor coordinates are fed as input to MATLAB and the k-means algorithm that is implemented in MATLAB is used to dynamically perform clustering of the sensors into n number of clusters.

In addition to clustering, we also determine the cluster head of each cluster mathematically in MATLAB. The distance of each sensor from the centroid of the cluster to which it belongs is calculated. Then the sensor which has the least distance is elected as the cluster head.

From MATLAB we get the cluster id of each sensor, the cluster heads of each cluster, and the size of each cluster.

All the above steps are performed periodically which can be defined as per the implementation. Each time the cluster members and the cluster heads are determined based on the current position and they are not fixed.

The codes required for the mathematical calculations done in MATLAB are written to a `clustering.m` file and this file is available in the MATLAB folder under bin_x64 of Dynamic_Clustering_Workspace.

2.2 Implementation

The `clustering.m` file can be run in four different modes of cluster head election.

A `Dynamic_Clustering.c` file is added to the DSR project which contains the following functions:

- `fn_NetSim_dynamic_clustering_CheckDestination()` — determines whether the current device is the destination.

- `fn_NetSim_dynamic_clustering_GetNextHop()` — statically defines the routes within the cluster and from the cluster to the sink node, returning the next hop based on static routing.
- `fn_NetSim_dynamic_clustering_IdentifyCluster()` — returns the cluster id of the cluster to which a sensor belongs.
- `fn_NetSim_dynamic_clustering_run()` — calls the MATLAB interfacing function and passes the inputs from NetSim (sensor coordinates, number of clusters, and sensor count).
- `fn_netsim_dynamic_form_clusters()` — assigns each sensor to its respective cluster based on the cluster IDs obtained from MATLAB.
- `fn_netsim_assign_cluster_heads()` — assigns the cluster heads for each cluster based on the cluster head IDs obtained from MATLAB.
- `fn_NetSim_Dynamic_Clustering_Init()` — initializes all parameter values.

2.3 Static Routing

Static Routing is defined in such a way that the sensors in the cluster send the packets to the cluster head. The cluster head then directly sends the packets to the destination (sink node).

If the current sensor is the source device and if it is not a cluster head, then its next hop is its cluster head.

If the current sensor is the source device and if it is a cluster head, then its next hop is the destination (i.e.) the sink node.

If the current sensor is not the source, then the packet is sent to the destination (i.e.) the sink node.

NOTE: To run this code 64-bit version of MATLAB must be installed in your system.

2.4 Configuring Environment for MATLAB Execution

1. In Control Panel open > System > Advanced system settings > Edit the system environment variable > Environment Variables.

2. Add the following MATLAB install directory path in the Environment PATH variable:

```
<MATLAB_INSTALL_DIRECTORY>\bin\win64
```

For example:

```
C:\Program Files\MATLAB\R2023a\bin\win64
```

Note: If the machine has more than one MATLAB installed, the directory for the target platform must be ahead of any other MATLAB directory (for instance, when compiling a 64-bit application, the directory in the MATLAB 64-bit installation must be the first one on the PATH).

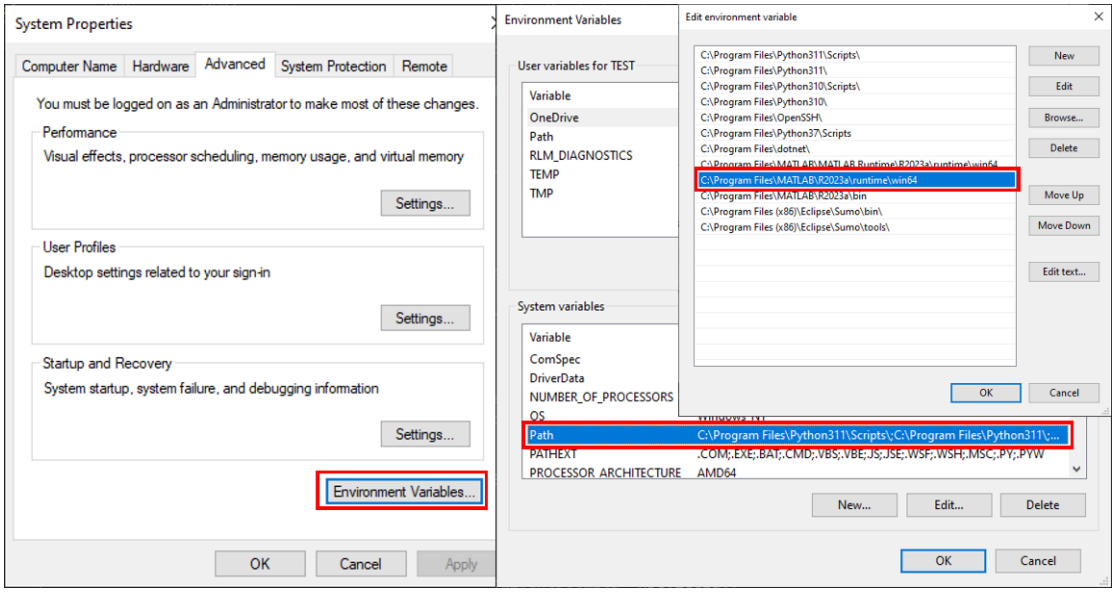


Figure 2-1: *Environment variable PATH*

2.5 Running the Example

1. Run NetSim in administrative mode.
2. Dynamic_Clustering comes with a sample network configuration that is already saved. To open this example, go to Your Work on the home screen of NetSim and click on **Dynamic_Clustering_Example** from the list of experiments.
3. The saved network scenario consists of 64 sensors uniformly distributed in the grid environment along with a sink node forming a Wireless Sensor Network. Traffic is configured from each sensor node to the Sink Node.

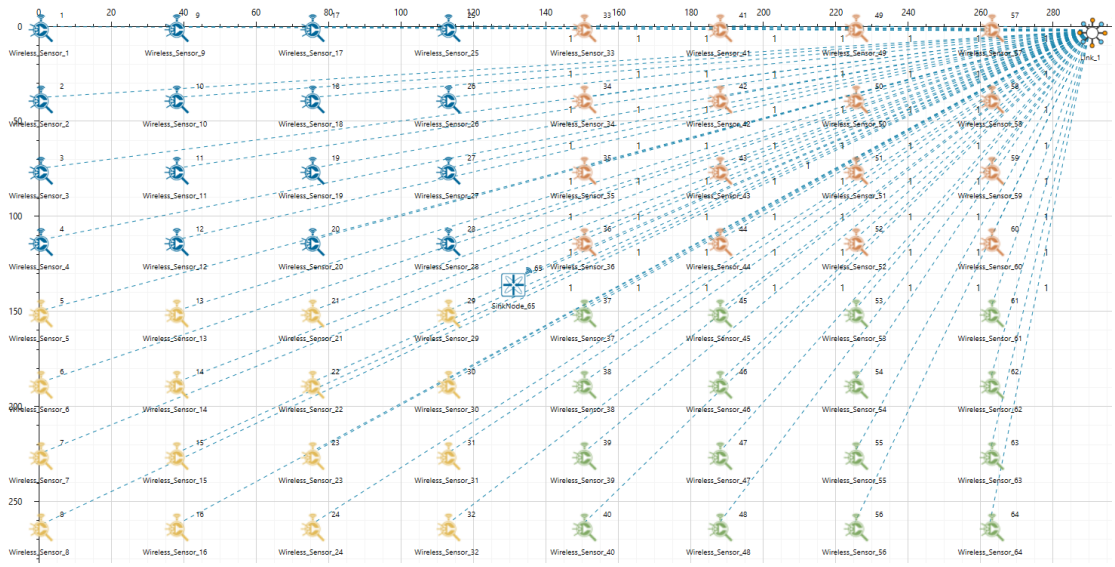


Figure 2-2: *Network scenario in this project*

4. Run the simulation and press any key to continue. The NetSim simulation console will show the message "Waiting for NetSim MATLAB Interface to connect...". NetSim will automatically open the MatlabInterface.exe console window.

5. The MatlabInterface.exe console window will open. As the simulation starts in NetSim, MATLAB gets initialized and the graph associated with energy consumption in the sensor network is plotted during runtime.

3 MATLAB Results

3.1 Example Scenario

The saved example contains 64 wireless sensors and one sink node. Each sensor sends sensor application traffic to the sink node. Dynamic clustering assigns the sensors to 4 clusters and elects the cluster heads during the simulation.

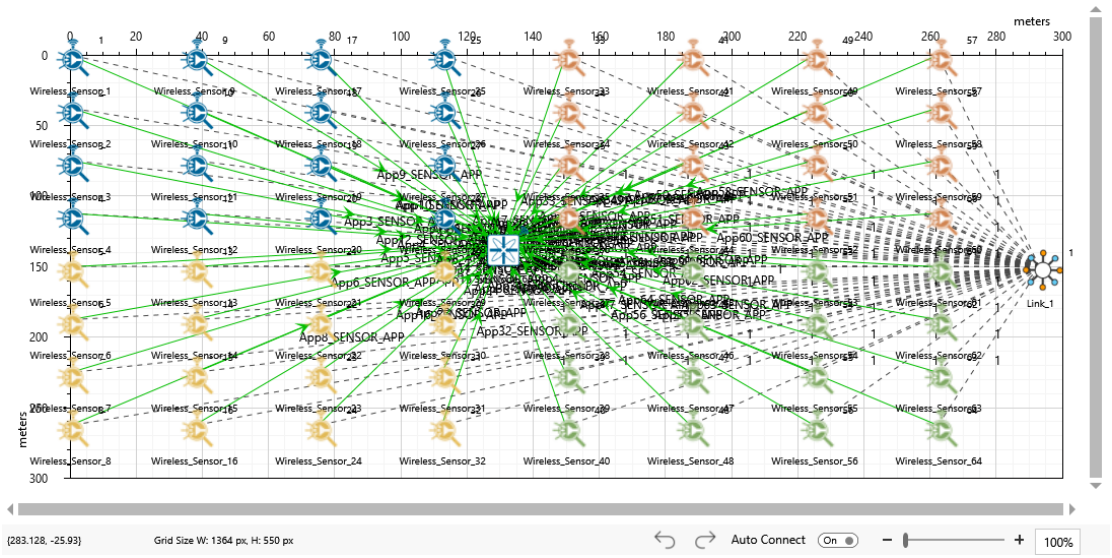


Figure 3-1: *Dynamic Clustering v15.0 network scenario with 64 sensors and sink node*

The example is located at:

```
<workspace>\Dynamic-Clustering-Example
```

To run through the command line:

```
<workspace>\bin_x64\NetSimCore.exe ^
  -iopath "<workspace>\Dynamic-Clustering-Example" ^
  -license 5053@192.168.0.4
```

3.2 Battery Model Metrics (MATLAB)

A total of 64 sensors are placed evenly on the grid environment and each sensor is set to have equal initial energy.

At the end of the simulation, NetSim provides Battery Model Metrics, which can be accessed by navigating to the Additional Metrics section. Here you can see detailed information related to energy consumption in each sensor node with respect to transmission, reception, idle mode, sleep mode, etc., as shown below:

Battery model

Device Name	Initial energy(mJ)	Consumed energy(mJ)	Remaining Energy(mJ)	Harvested Energy(mJ)	Transmitting energy(mJ)	Receiving energy(mJ)	Idle energy(mJ)	Sleep energy(mJ)
WIRELESS_SENSOR_1	6480.000000	567.958525	5912.041475	0.000000	29.672375	0.278692	538.007458	0.000000
WIRELESS_SENSOR_2	6480.000000	571.032694	5908.967306	0.000000	32.755219	0.378225	537.899251	0.000000
WIRELESS_SENSOR_3	6480.000000	563.940953	5916.059047	0.000000	25.626142	0.165888	538.148923	0.000000
WIRELESS_SENSOR_4	6480.000000	564.784872	5915.215128	0.000000	26.493192	0.172524	538.119157	0.000000
WIRELESS_SENSOR_5	6480.000000	563.385993	5916.614007	0.000000	24.951770	0.265421	538.168802	0.000000
WIRELESS_SENSOR_6	6480.000000	567.175519	5912.824481	0.000000	28.901664	0.238879	538.034976	0.000000
WIRELESS_SENSOR_7	6480.000000	563.520677	5916.479323	0.000000	25.144448	0.212337	538.163893	0.000000

Figure 3-2: *Battery Model Metrics from the MATLAB run*

This information can also be obtained at different points of simulation time, either to log or to send to other external tools. The battery information and the position coordinates are passed to MATLAB periodically for clustering (the number of clusters is set to 4), cluster head election, and to obtain energy consumption plots.

The clustering method can be customized as needed by modifying the `clustering.m` file located in the MATLAB folder within the workspace.

clustering.m

```
% ------------------------------------------------------------------------------%

function [A,B,C] = clustering(x,scount,num_cls,power,max_energy)
    % changed clustering function. New paramter power: column vector of
    % remaining power for each device
    % s_count is sensor_count


    % Clustering_Method = 1          KMeans using distance
    %                   = 2          Fuzzy C Means using distance
    %                   = 3          KMeans using distance and power
    %                   = 4          Fuzzy C Means using distance and power

    Clustering_Method = 1;

%     save dynamic_clustering.mat

    %change here for different algorithm
    if(Clustering_Method == 1 || Clustering_Method == 3)
        [IDX,C]= k_means(x,num_cls);
    else
        [IDX,C]= fuzzy(x,num_cls);
    end


    cl_count=zeros(1,num_cls);
    cl_dist=zeros(1,scount);

    if(Clustering_Method > 2)                % only when method involves power
        cl_max_dist = zeros(1,num_cls);      % max distance in each cluster
        cl_max_power = zeros(1,num_cls);     % max device power left in each cluster
    end

```

The clustering method lets you easily switch between four different approaches using methods 1, 2, 3, and 4.

Figure 3-3: *clustering.m MATLAB file*

3.3 MATLAB Energy Plots

3.3.1 Cluster Head Election Using Distance Alone (Methods 1 and 2)

Running simulations with Clustering Method set to 1 and 2 in the `clustering.m` file will provide energy consumption plots for k-means and fuzzy c-means algorithms respectively, as shown below:

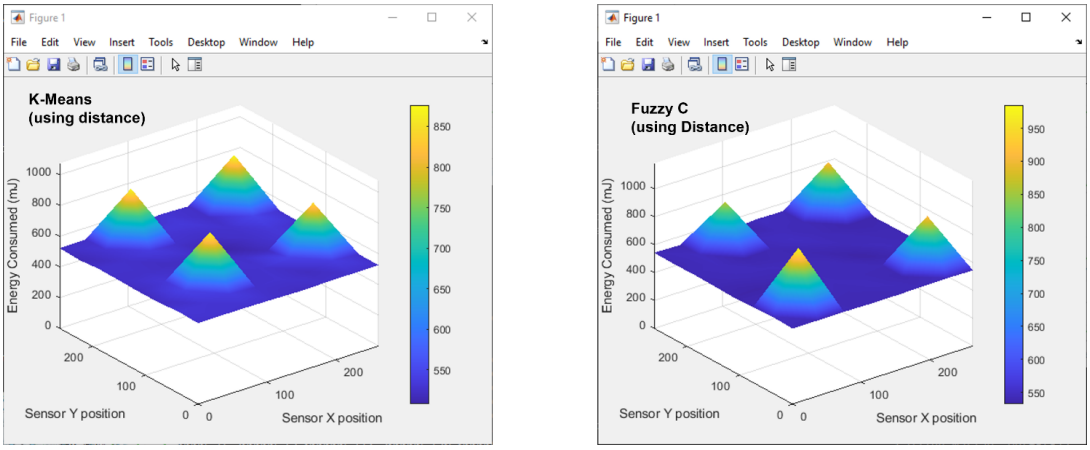


Figure 3-4: *MATLAB energy consumption plots for k-means and Fuzzy C-Means (Methods 1 and 2)*

As seen from the plot, there are 4 peaks corresponding to higher energy consumption in the nodes at the centre of each cluster, as they always become the cluster heads. This is because distance alone is used as the parameter for electing the cluster heads.

3.3.2 Cluster Head Election Using Distance and Remaining Energy (Methods 3 and 4)

Running simulations with the Clustering Method set to 3 and 4 in the `clustering.m` file will provide energy consumption plots for k-means and fuzzy c-means algorithms respectively, as shown below:

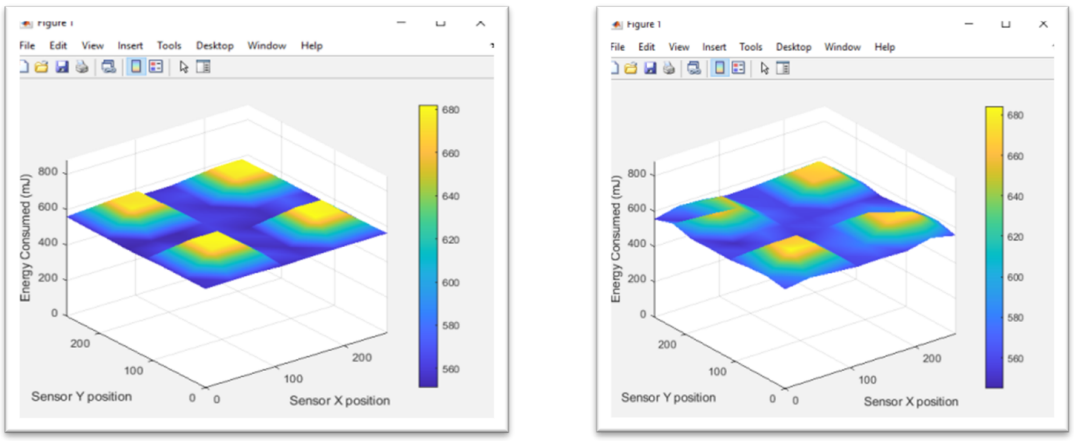


Figure 3-5: *MATLAB energy consumption plots for k-means and Fuzzy C-Means (Methods 3 and 4)*

In the initial phase the plot resembles the previous one. However, as time passes, it can be observed that power is consumed by all sensors at approximately the same rate. There are no sharp peaks in this plot because the modified K-Means considers the power level of each sensor, allowing sensors other than those at the centre of the cluster to be elected as cluster head.

3.4 MATLAB First Node Failure Time

A common measure of network lifetime is the time at which the first sensor's residual energy reaches zero (the first-node-failure time). Table 3-1 reports this time for each of the four

clustering methods under the MATLAB workflow.

Table 3-1: *MATLAB: Time at which the first sensor's residual energy reaches zero (first-node-failure time) for each clustering method.*

Clustering method	Time (s)
K-Means using distance	2 082.00
Fuzzy C-Means using distance	1 898.00
K-Means using distance and power	3 241.00
Fuzzy C-Means using distance and power	3 245.01

By factoring in both distance and remaining battery power, the power-aware methods extend the network lifetime significantly. Under the MATLAB workflow, the first node stays alive for about 3241–3245 s—which is roughly 1.6× longer than the 1898–2082 s achieved by distance-only methods.

This improvement occurs because the energy workload is shared more evenly across all sensors. Instead of repeatedly electing and draining the same cluster-center nodes, the system rotates the cluster head role to other nodes with higher remaining energy. This prevents premature node failure and prolongs the life of the entire network.

4 Python Workflow

The Python extension replaces the MATLAB call with the NetSim Python interface. The DSR project still gathers the same simulation values, but it sends them to `dynamic_clustering_client.py` instead of MATLAB.

4.1 Python Client Folder in the Workspace

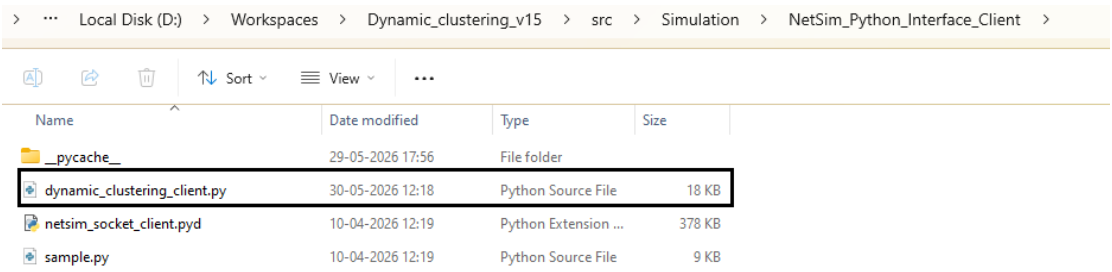


Figure 4-1: *Python interface client folder under the v15 workspace source tree*

The Python files are located in the workspace source tree:

```
<workspace>\src\Simulation\NetSim_Python_Interface_Client
```

4.2 Python Client File

The dynamic clustering Python script is:

```
<workspace>\src\Simulation\NetSim_Python_Interface_Client\dynamic_clustering_client.py
```

This file receives sensor coordinates and battery values from NetSim, calculates cluster IDs and cluster heads, and returns the result to the DSR project.

4.3 Changing the Clustering Method in Python

When NetSim starts the Python client, the console displays the clustering method selector. Use the Up/Down arrow keys to highlight the required method and press Enter. The selected method is then used for the simulation run. The available methods are:

1. K-Means using distance
2. FUZZY C Means using distance
3. K-Means using distance and power
4. FUZZY C Means using distance and power

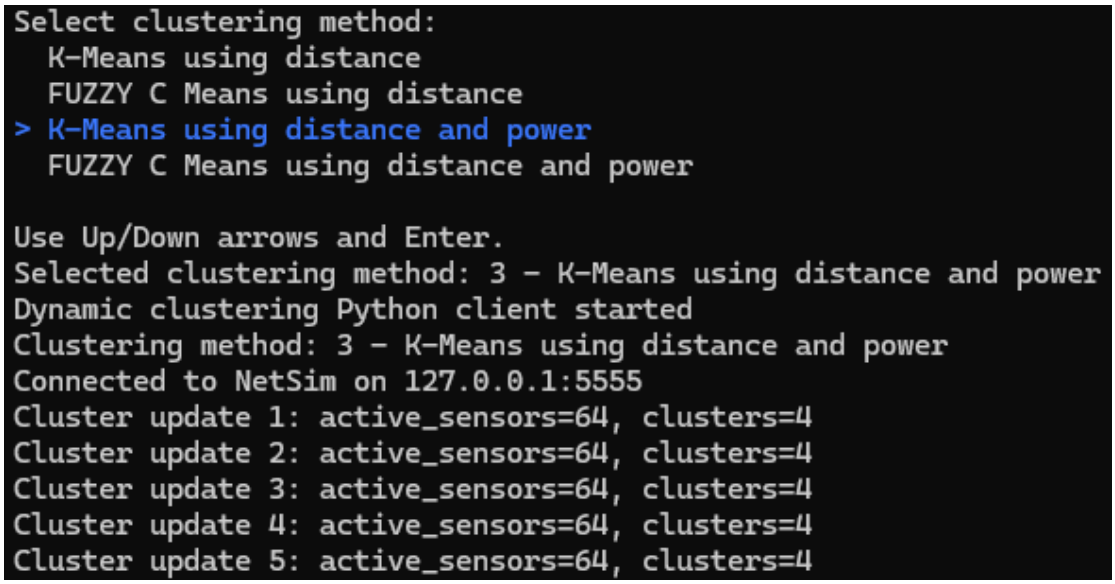


Figure 4-2: *Python console showing the clustering method selector*

4.4 Runtime Socket Log

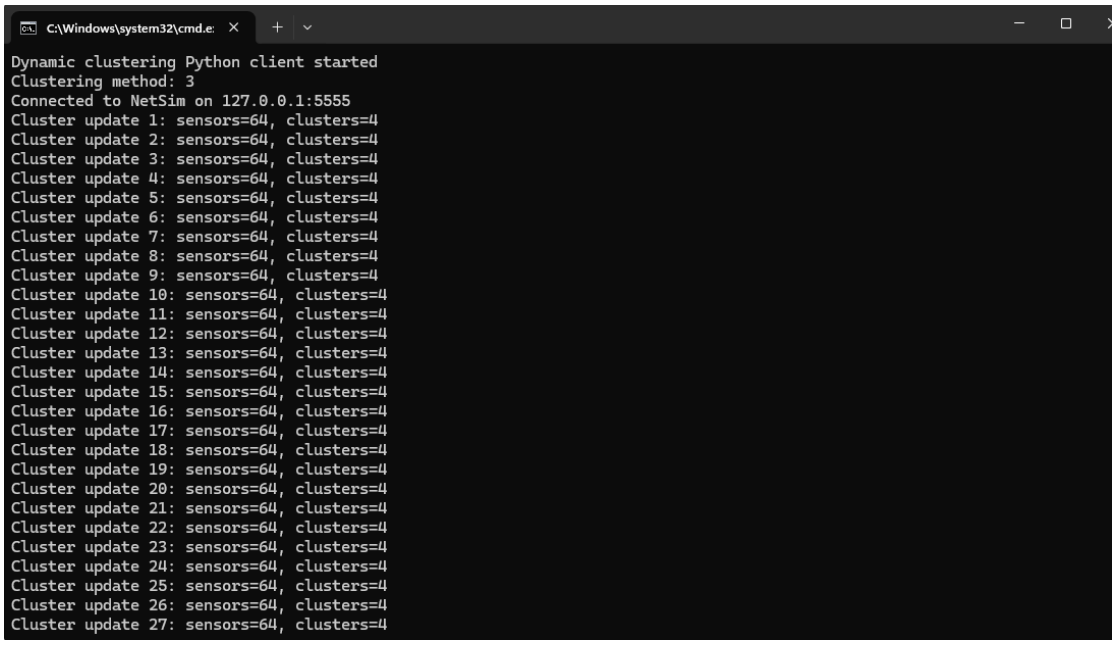


Figure 4-3: *Python runtime console while NetSim sends clustering data*

The log confirms that NetSim has started the Python client and is exchanging clustering data during runtime.

5 Python Results

5.1 Battery Model Metrics (Python)

A total of 64 sensors are placed evenly on the grid environment and each sensor is set to have equal initial energy.

At the end of the simulation, NetSim provides Battery Model Metrics, which can be accessed by navigating to the Additional Metrics section. Here you can see detailed information related to energy consumption in each sensor node with respect to transmission, reception, idle mode, sleep mode, etc., as shown below:

Battery model

Device Name	Initial energy(mJ)	Consumed energy(mJ)	Remaining Energy(mJ)	Harvested Energy(mJ)	Transmitting energy(m	Receiving energy(mJ)	Idle energy(mJ)	Sleep energy(mJ)
WIRELESS_SENSOR_1	6480.000000	568.328306	5911.671694	0.000000	30.057731	0.298598	537.971977	0.000000
WIRELESS_SENSOR_2	6480.000000	566.986972	5913.013028	0.000000	28.708986	0.258785	538.019201	0.000000
WIRELESS_SENSOR_3	6480.000000	565.668291	5914.331709	0.000000	27.263903	0.338412	538.065977	0.000000
WIRELESS_SENSOR_4	6480.000000	565.902458	5914.097542	0.000000	27.552920	0.291963	538.057576	0.000000
WIRELESS_SENSOR_5	6480.000000	568.090771	5911.909229	0.000000	29.865053	0.245514	537.980204	0.000000
WIRELESS_SENSOR_6	6480.000000	564.362159	5915.637841	0.000000	26.011498	0.238879	538.111783	0.000000
WIRELESS_SENSOR_7	6480.000000	569.056826	5910.943174	0.000000	30.732103	0.378225	537.946199	0.000000

Figure 5-1: *Battery Model Metrics from the Python run*

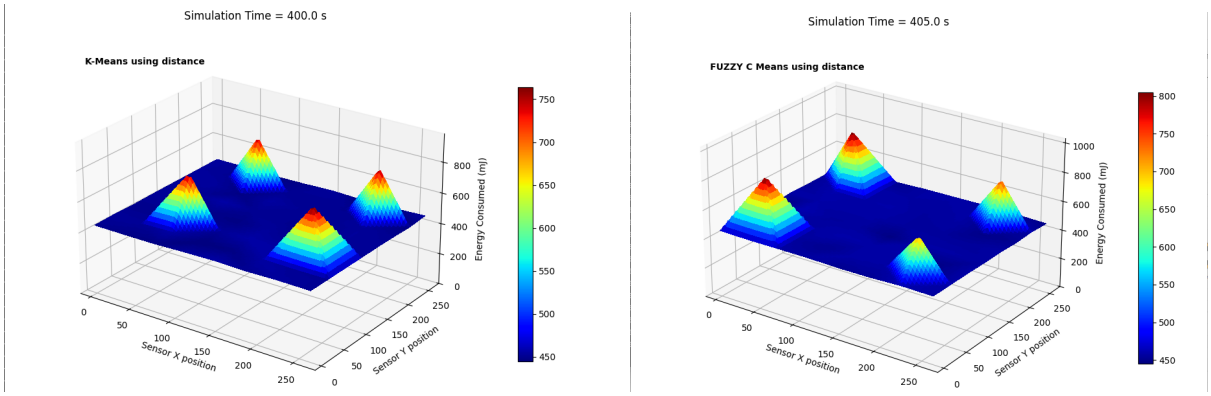
This information can also be obtained at different points of simulation time, either to log or to send to other external tools. The battery information and the position coordinates are passed to Python periodically for clustering (the number of clusters is set to 4), cluster head election, and to obtain energy consumption plots.

The clustering method can be customized as needed by modifying the `dynamic_clustering_client.py` file located in the Python folder within the workspace.

5.2 Python Energy Plots

5.2.1 Energy Plot for Method 1 and Method 2

Method 1 uses K-Means with distance as the cluster head election parameter. Method 2 uses Fuzzy C-Means with distance as the cluster head election parameter.

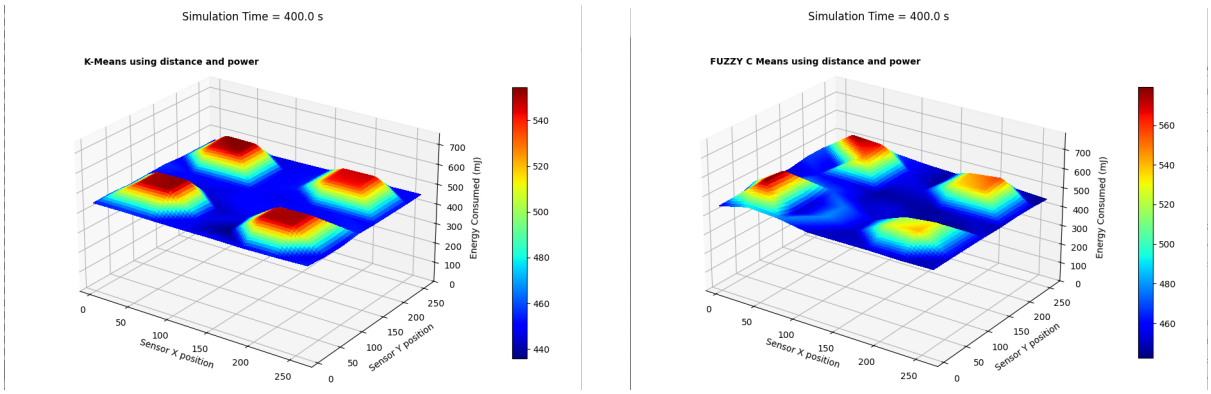


Method 1: K-Means using distance **Method 2: Fuzzy C-Means using distance**

Figure 5-2: *Python energy consumption plots for Method 1 and Method 2*

5.2.2 Energy Plot for Method 3 and Method 4

Method 3 uses K-Means with distance and battery power. Method 4 uses Fuzzy C-Means with distance and battery power.



Method 3: K-Means using distance and battery power **Method 4: Fuzzy C-Means using distance and battery power**

Figure 5-3: *Python energy consumption plots for Method 3 and Method 4*

5.3 Python First Node Failure Time

A common measure of network lifetime is the time at which the first sensor's residual energy reaches zero (the first-node-failure time). Table 5-1 reports this time for each of the four clustering methods under the Python workflow.

Table 5-1: *Python: Time at which the first sensor's residual energy reaches zero (first-node-failure time) for each clustering method.*

Clustering method	Time (s)
K-Means using distance	1 994.11
Fuzzy C-Means using distance	1 896.00
K-Means using distance and power	3 246.01
Fuzzy C-Means using distance and power	3 245.00

By factoring in both distance and remaining battery power, the power-aware methods extend the network lifetime significantly. Under the Python workflow, the first node stays alive for about 3245–3246 s—which is roughly 1.7× longer than the 1896–1994 s achieved by distance-only methods.

This improvement occurs because the energy workload is shared more evenly across all sensors. Instead of repeatedly electing and draining the same cluster-center nodes, the system rotates the cluster head role to other nodes with higher remaining energy. This prevents premature node failure and prolongs the life of the entire network.

6 MATLAB and Python Result Notes

The Battery Model is calculated by NetSim in both workflows. MATLAB or Python only supplies the cluster head and cluster ID arrays. If MATLAB and Python produce the same cluster heads and cluster IDs at every clustering interval, the Battery Model values should be close. If either output differs, packet routing can change, and transmission, reception, and idle energy values can shift.

Differences can come from:

- MATLAB `kmeans()` and Python/SciPy K-Means implementation details.
- Fuzzy C-Means membership update and stopping rule differences.
- Cluster label ordering.
- Tie handling near the centroid.
- Energy feedback across later clustering intervals.

