

Dynamic Clustering in WSN

Software: NetSim Standard v14.1 (64 bit), Visual Studio 2022, MATLAB R2019 or higher

Project Download Link:

https://github.com/NetSim-TETCOS/Dynamic_Clustering_in_WSN-v14.1/archive/refs/heads/main.zip

Follow the instructions specified in the following link to download and set up the Project in NetSim:

<https://support.tetcos.com/en/support/solutions/articles/14000128666-downloading-and-setting-up-netsim-file-exchange-projects>

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.

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. $[IDX, C] = \text{kmeans}(X, k)$.

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

C – Centroids of each cluster.

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.

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.

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.

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

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()`//This function is used to determine whether the current device is the destination.
- `fn_NetSim_dynamic_clustering_GetNextHop()`//This function statically defines the routes within the cluster and from the cluster to the sink node. It returns the next hop based on the static routing that is defined.
- `fn_NetSim_dynamic_clustering_IdentifyCluster()`//This function returns the cluster id of the cluster to which a sensor belongs.
- `fn_NetSim_dynamic_clustering_run()`//This function makes a call to MATLAB interfacing function and passes the inputs from NetSim (i.e) the sensor coordinates, number of clusters, and the sensor count.
- `fn_netsim_dynamic_form_clusters()`//This function assigns each sensor to its respective

- clusters based on the cluster IDs obtained from MATLAB.
- `fn_netsim_assign_cluster_heads()`//This function assigns the cluster heads for each cluster based on the cluster head IDs obtained from MATLAB.
- `fn_NetSim_Dynamic_Clustering_Init()`//This function initializes all parameter values.

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.

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

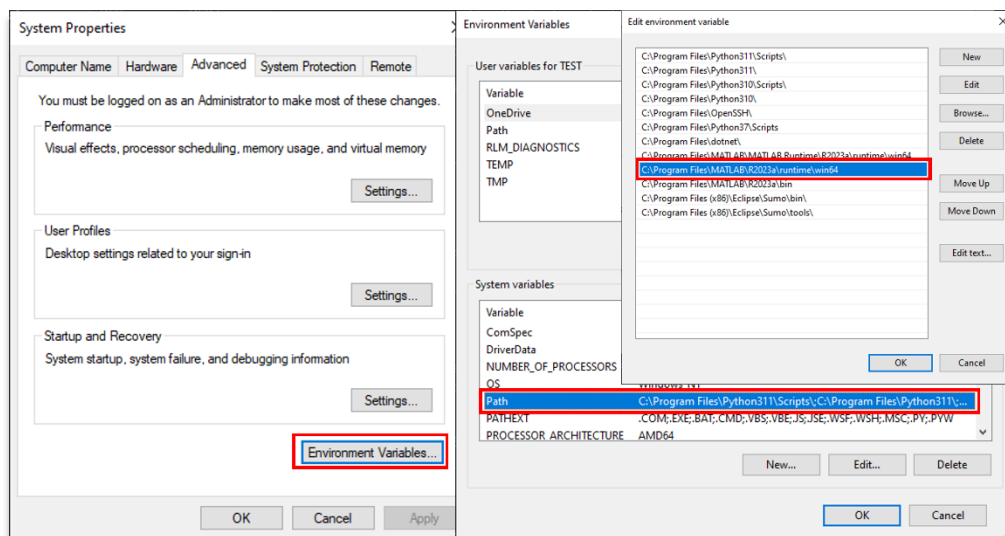


Figure 1: Environment variable PATH

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).

Example:

1. Run NetSim in administrative mode.

2. **Dynamic_Clustering_Workspace** 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 the **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.

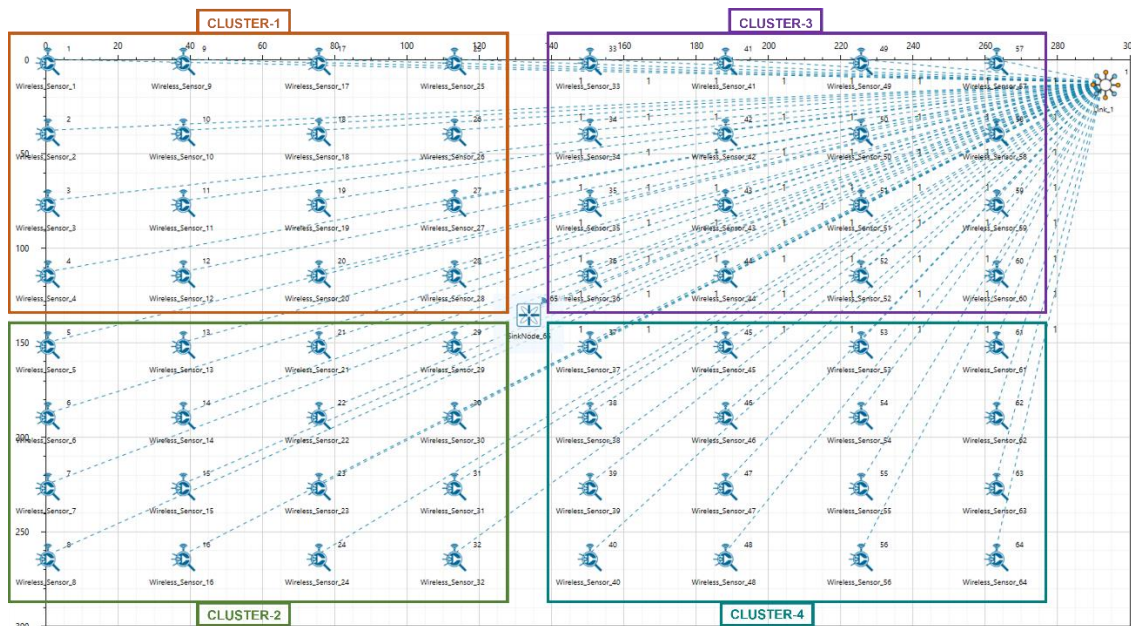


Figure 2: Network Scenario in this project

4. Run the simulation and press any key to continue. NetSim simulation console will show the following message in the console "Waiting for NetSim MATLAB Interface to connect...". NetSim will automatically open the MatlabInterface.exe console window.
5. It will open the MatlabInterface.exe console window. You will observe that as the simulation starts in NetSim, MATLAB gets initialized and the graph associated with energy consumption in the sensor network is plotted during runtime.

Results and discussion:

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 the 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_Table							
Battery model							
Device Name	Initial energy(mJ)	Consumed energy(mJ)	Remaining Energy(mJ)	Transmitting energy(mJ)	Receiving energy(mJ)	Idle energy(mJ)	Sleep energy(mJ)
WIRELESS_SENSOR_1	6480.000000	567.958525	5912.041475	29.672375	0.278692	538.007458	0.000000
WIRELESS_SENSOR_2	6480.000000	571.032694	5908.967306	32.755219	0.378225	537.899251	0.000000
WIRELESS_SENSOR_3	6480.000000	563.940953	5916.059047	25.626142	0.165888	538.148923	0.000000
WIRELESS_SENSOR_4	6480.000000	564.784872	5915.215128	26.493192	0.172524	538.119157	0.000000
WIRELESS_SENSOR_5	6480.000000	563.385993	5916.614007	24.951770	0.265421	538.168802	0.000000
WIRELESS_SENSOR_6	6480.000000	567.175519	5912.824481	28.901664	0.238879	538.034976	0.000000
WIRELESS_SENSOR_7	6480.000000	563.520677	5916.479323	25.144448	0.212337	538.163893	0.000000
WIRELESS_SENSOR_8	6480.000000	564.723959	5915.276041	26.396853	0.205701	538.121404	0.000000
WIRELESS_SENSOR_9	6480.000000	568.921212	5911.078788	30.635764	0.311869	537.973578	0.000000
WIRELESS_SENSOR_10	6480.000000	568.547617	5911.452383	27.794385	2.759270	537.993962	0.000000
WIRELESS_SENSOR_11	6480.000000	584.399637	5895.600363	34.601561	12.335432	537.462644	0.000000
WIRELESS_SENSOR_12	6480.000000	571.668771	5908.331229	28.999238	4.779786	537.889747	0.000000
WIRELESS_SENSOR_13	6480.000000	564.897211	5915.102789	26.589531	0.192430	538.115251	0.000000
WIRELESS_SENSOR_14	6480.000000	566.915640	5913.084360	28.612647	0.258785	538.044207	0.000000
WIRELESS_SENSOR_15	6480.000000	571.976239	5908.023761	29.698004	4.400456	537.877779	0.000000

Figure 3: NetSim provides Battery Model Metrics

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 bin_x64 directory of the workspace.

```

13 %
14
15 function [A,B,C] = clustering(x,scount,num_cls,power,max_energy)
16 % changed clustering function. New parameter power: column vector of
17 % remaining power for each device
18 % s_count is sensor_count
19
20
21 % Clustering_Method = 1      KMeans using distance
22 %                       = 2      Fuzzy C Means using distance
23 %                       = 3      KMeans using distance and power
24 %                       = 4      Fuzzy C Means using distance and power
25
26 Clustering_Method = 1;
27
28 % save dynamic_clustering.mat
29
30 %change here for different algorithm
31 if(Clustering_Method == 1 || Clustering_Method == 3)
32     [IDX,C]= k_means(x,num_cls);
33 else
34     [IDX,C]= fuzzy(x,num_cls);
35 end
36
37
38 cl_count=zeros(1,num_cls);
39 cl_dist=zeros(1,scount);
40
41 if(Clustering_Method > 2)
42     cl_max_dist = zeros(1,num_cls); % only when method involves power
43     cl_max_power = zeros(1,num_cls); % max distance in each cluster
44                                     % max device power left in each cluster
45 end

```

Figure 4: Clustering.m matlab file

Cluster head election using distance alone as a parameter:

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:

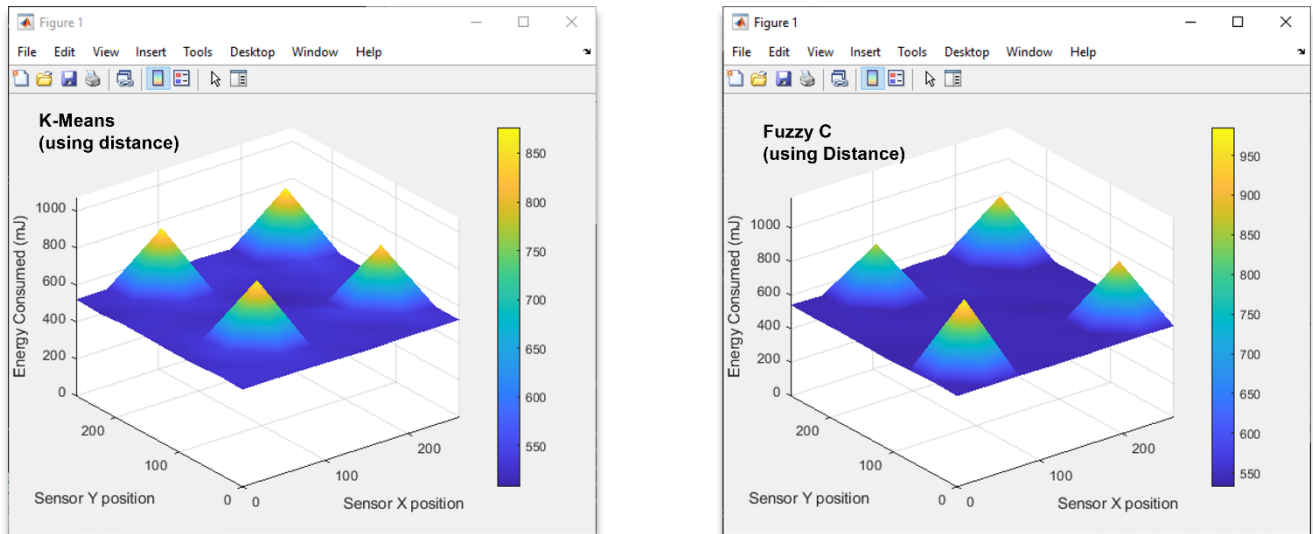


Figure 5: Energy consumption plots for k means and fuzzy c-means algorithms using methods 1 and 2

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

Cluster head election using distance and remaining energy as parameters.

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:

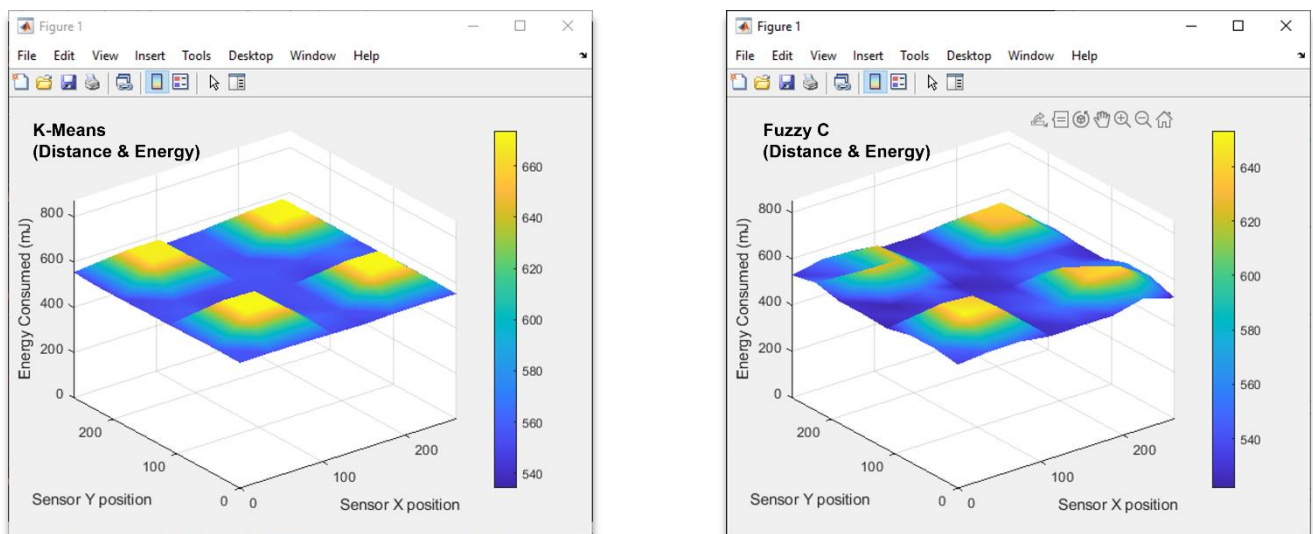


Figure 6: Energy consumption plots for k means and fuzzy c-means algorithms using method 3 and 4

In the initial phase the plot resembles the previous one. However, as time passes, it can be observed that the power is consumed by all the sensors at approximately the same rate.

There are no sharp peaks in this plot unlike the previous one because modified K-means consider the power level of each sensor and thus sensors other than those in the center of the cluster will also get a chance to be elected as the cluster head in their respective cluster.