Step by Step Explanation of Implementation of MS-MAC in MATLAB with CODE

1 Navigate to https://matlab.mathworks.com/

2

In this Document, We are going to implement **MS-MAC protocol** with simulation and proper visualization in **MATLAB**

Firstly, Lets Have an Intro that what the MS MAC Protocol is used for: The MS-MAC (**Mobility-aware Sensor Medium Access Control**) protocol is designed for wireless sensor networks to address energy-efficient communication in scenarios with both stationary and mobile nodes. The protocol extends the ideas from the S-MAC (Sensor-MAC) protocol, which is based on the IEEE 802.11 standard but introduces periodic sleep/wakeup cycles to conserve energy in sensor nodes.

Objective: MS-MAC aims to work energy-efficiently in both stationary and mobile scenarios.

Base Protocol - S-MAC:

MS-MAC starts with the S-MAC protocol, which is a CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) MAC protocol based on IEEE 802.11. S-MAC introduces periodic coordinated sleep/wakeup duty cycles, extending battery lifetime for sensor nodes.

Mobility Support: MS-MAC extends S-MAC to support mobile sensors. When nodes are stationary, MS-MAC operates similarly to S-MAC to conserve energy.

Active Zones: MS-MAC introduces the concept of "active zones" for mobile nodes crossing virtual cluster borders. Active zones are regions where nodes stay awake longer to facilitate quick connection setups with new neighbors.

Mobility Detection: Nodes detect mobility within their neighborhood based on received signal levels of periodic SYNC messages from neighbors. If a change in signal level is detected, it is presumed that a node (or the node itself) is moving.

SYNC Message Enhancement: SYNC messages in MS-MAC not only include information on the schedule of the sender node but also information on the estimated speed of mobile neighbors.

Active Zone Formation: Active zones are created based on the speed of the mobile node. Faster movement leads to a larger active zone.

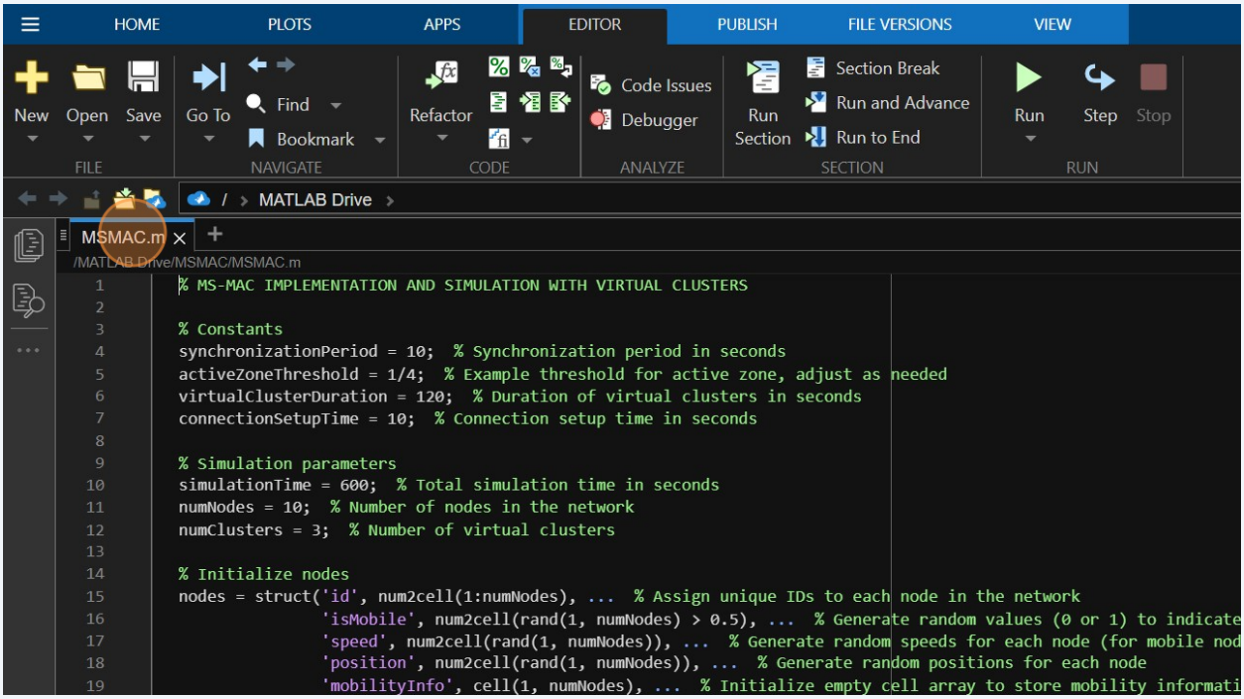
Mobility information in SYNC messages is used to create an active zone around a moving node when it moves from one cluster to another.

Synchronization Period Adjustment: Nodes in the active zone adjust their synchronization periods more frequently, resulting in higher energy consumption but faster connection setups.

Efficient Operation: MS-MAC allows nodes to work efficiently in both stationary and mobile scenarios. In stationary scenarios, nodes conserve energy by staying in a low-power mode for most of the time.

Connection Setups: Active zones expedite connection setups with new neighbors for mobile nodes before losing all existing neighbors.

Now lets Implement It...


HOME
PLOTS
APPS
EDITOR
PUBLISH
FILE VERSIONS
VIEW
New
Open
Save
Go To
Find
Bookmark
Refactor
Code Issues
Debugger
Run Section
Section Break
Run and Advance
Run to End
Run
Step
Stop
FILE
NAVIGATE
CODE
ANALYZE
SECTION
RUN
MATLAB Drive
MSMAC.m
/MATLAB Drive/MSMAC/MSMAC.m
% MS-MAC IMPLEMENTATION AND SIMULATION WITH VIRTUAL CLUSTERS
% Constants
synchronizationPeriod = 10; % Synchronization period in seconds
activeZoneThreshold = 1/4; % Example threshold for active zone, adjust as needed
virtualClusterDuration = 120; % Duration of virtual clusters in seconds
connectionSetupTime = 10; % Connection setup time in seconds
% Simulation parameters
simulationTime = 600; % Total simulation time in seconds
numNodes = 10; % Number of nodes in the network
numClusters = 3; % Number of virtual clusters
% Initialize nodes
nodes = struct('id', num2cell(1:numNodes), ... % Assign unique IDs to each node in the network
'isMobile', num2cell(rand(1, numNodes) > 0.5), ... % Generate random values (0 or 1) to indicate
'speed', num2cell(rand(1, numNodes)), ... % Generate random speeds for each node (for mobile nod
'position', num2cell(rand(1, numNodes)), ... % Generate random positions for each node
'mobilityInfo', cell(1, numNodes), ... % Initialize empty cell array to store mobility informati

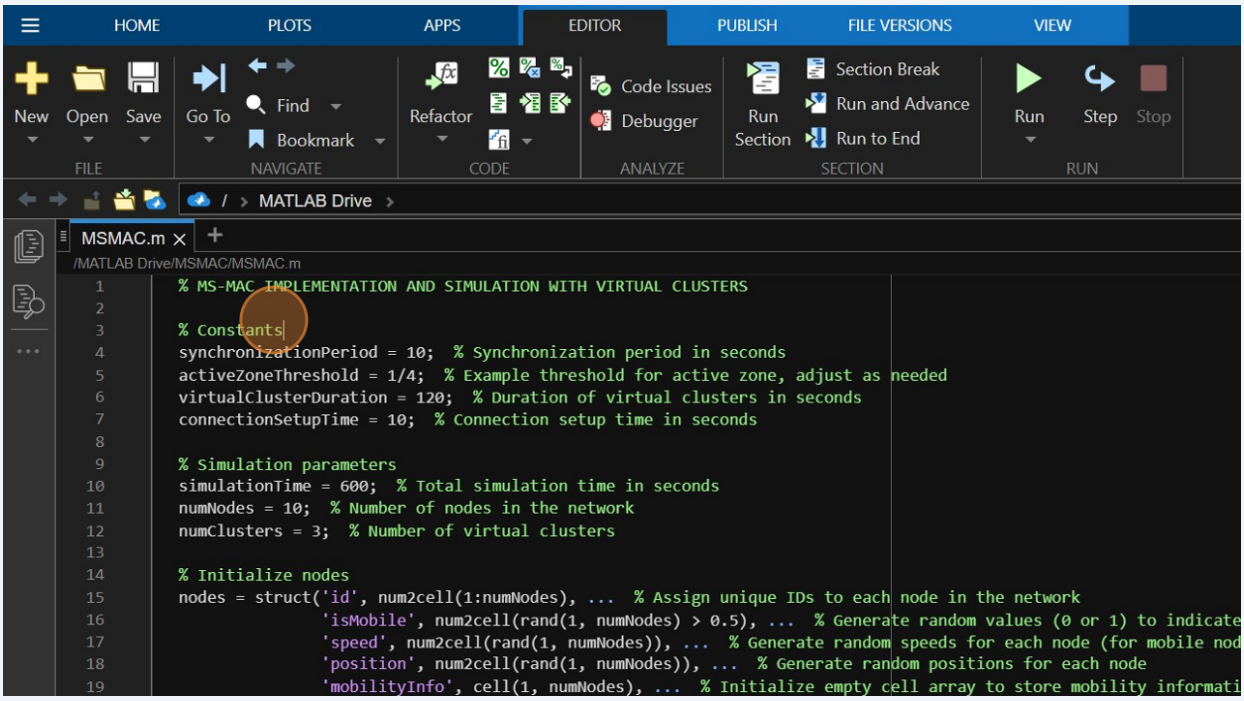
3 Here we are having the constants that will be used in the protocol implementation

1. synchronizationPeriod: This constant represents the time interval at which synchronization occurs between nodes. Nodes exchange information and adjust their behavior based on this synchronization period.

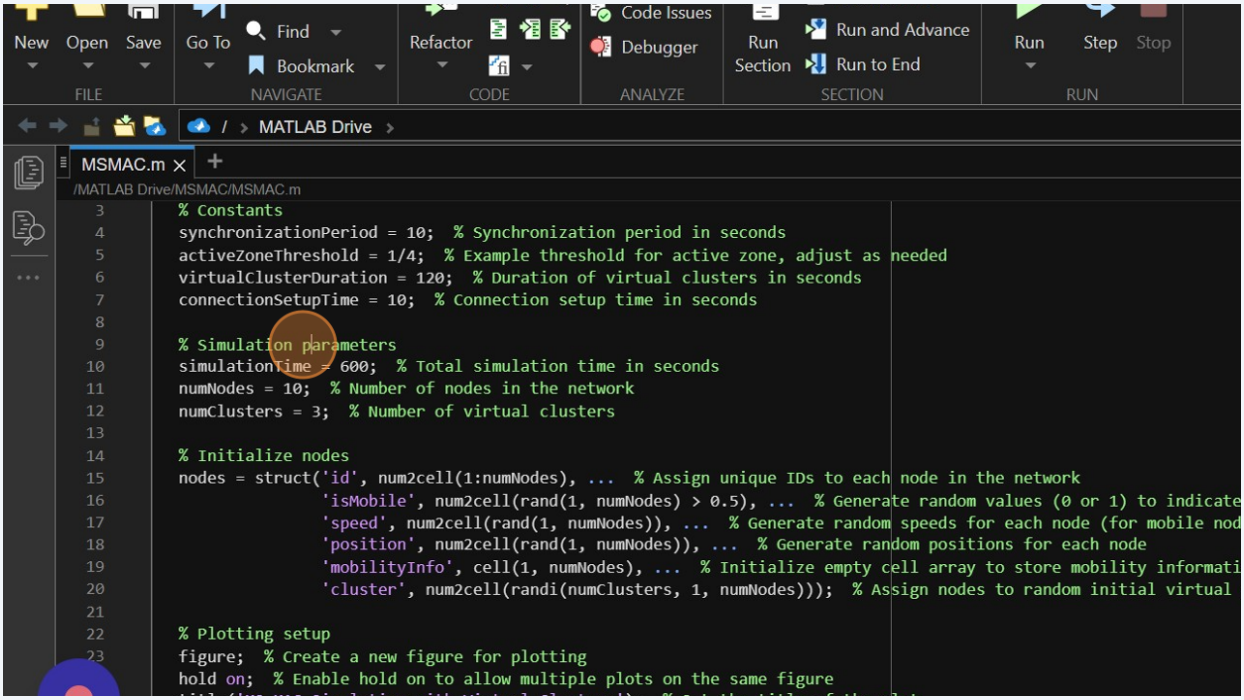
2. activeZoneThreshold: It defines a threshold for determining whether a node is in an active zone. Nodes with mobility information above this threshold are considered to be in an active zone, and special actions can be taken for them.

3. virtualClusterDuration: This constant sets the duration of virtual clusters. For stationary nodes, periodic synchronization actions occur at intervals defined by this duration.

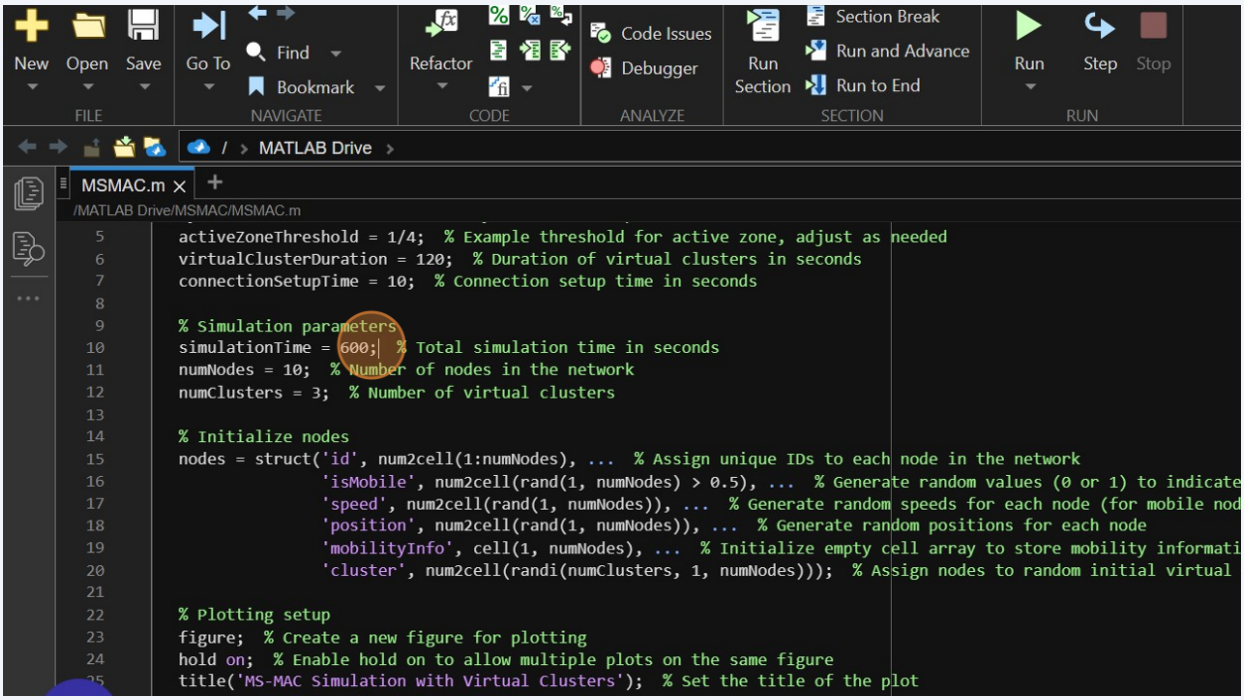
4. connectionSetupTime: It represents the time taken for setting up connections. This could be relevant for scenarios where there are connection-related actions or delays in the simulation



4 These lines of code define the parameters used in the simulation:



5 1. **simulationTime = 600;**: This parameter sets the total duration of the simulation in seconds. In this case, the simulation will run for 600 seconds (or 10 minutes).

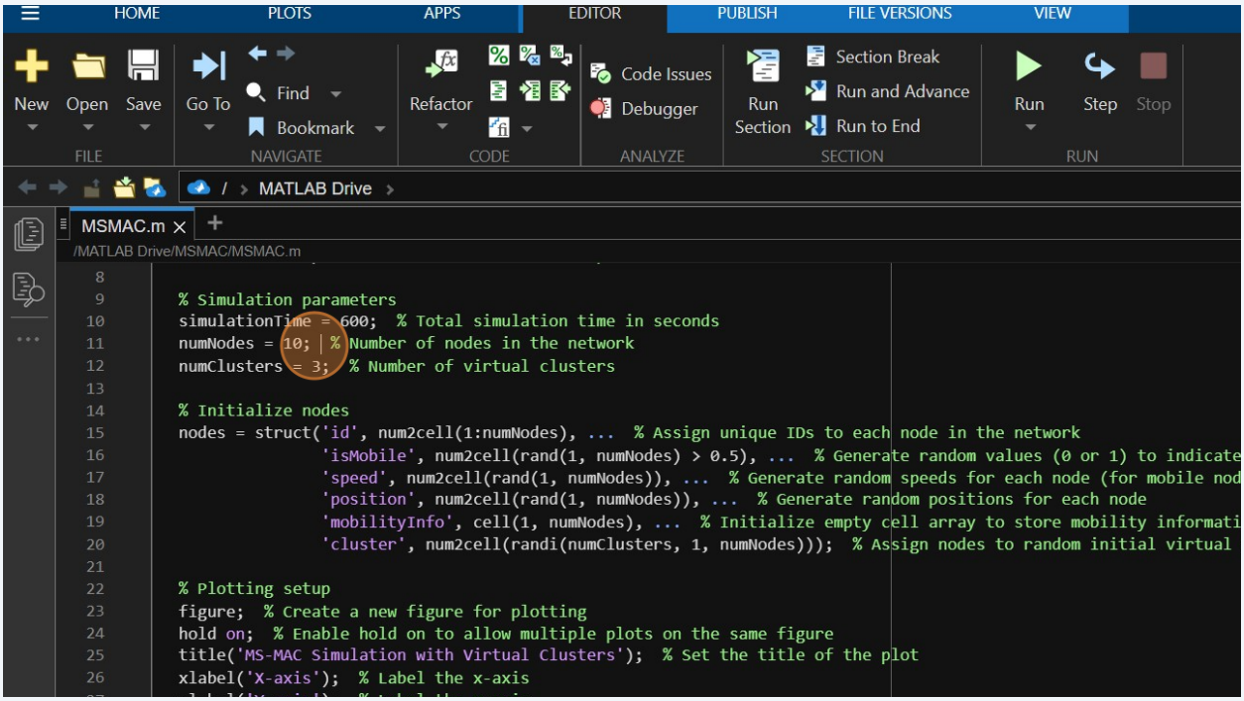


6

1. **numNodes = 10;**: This parameter specifies the number of nodes present in the network. In the simulation, there are 10 nodes participating in the network.

2. **numClusters = 3;**: This parameter determines the number of virtual clusters used in the simulation. Virtual clustering is a technique used to group nodes in a network for efficient communication and management. In this case, there are 3 virtual clusters established within the network.

These parameters are crucial as they define the basic characteristics and constraints of the simulation environment, such as the size of the network, the number of entities involved, and the duration of the simulation.



7 Let's go through each field of the nodes struct:

1. id: This field assigns a unique ID to each node in the network. IDs range from 1 to the total number of nodes (numNodes).

2. isMobile: This field indicates whether each node is mobile or not. It generates random values (0 or 1) using rand(1, numNodes) and checks if each value is greater than 0.5, effectively generating a binary value for mobility.

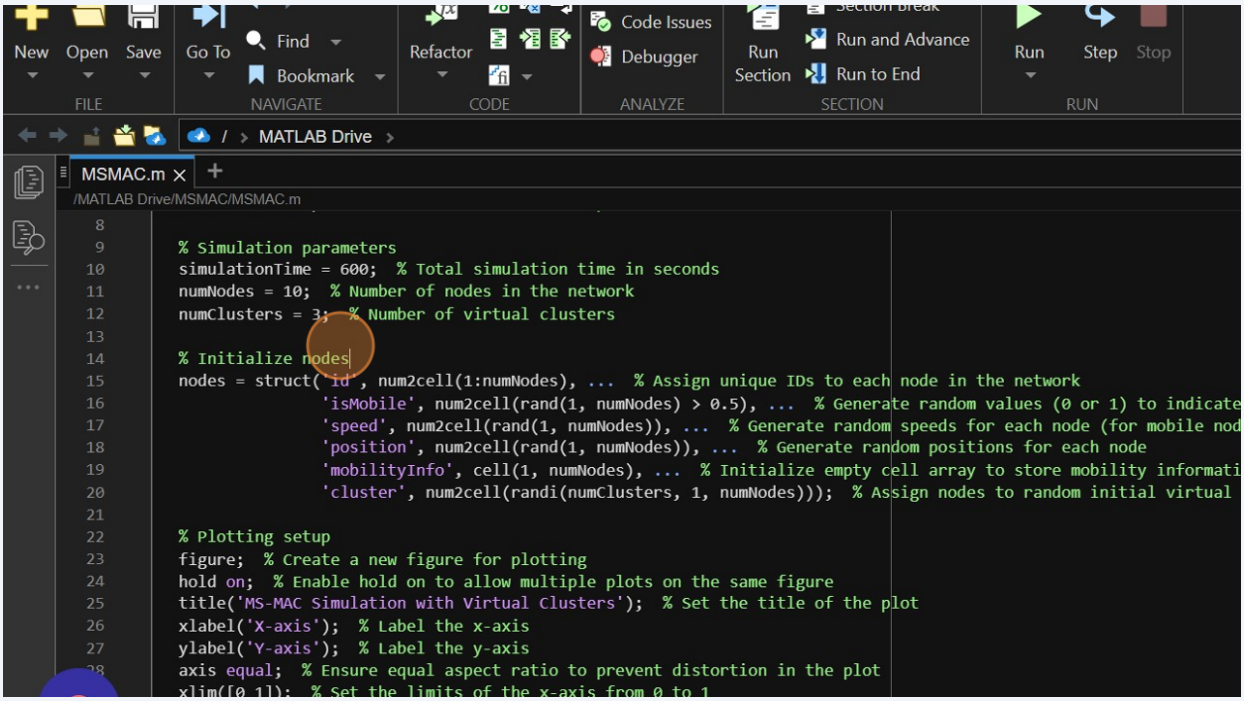
3. speed: This field assigns random speeds to each node. For mobile nodes, these speeds are randomly generated.

4. position: This field assigns random positions to each node. The positions are randomly generated within the range [0, 1].

5. mobilityInfo: This field is initialized as an empty cell array. It will later be populated with mobility information for each node during the simulation.

6. cluster: This field assigns each node to a random initial virtual cluster. The cluster IDs are randomly chosen from 1 to the total number of virtual clusters (numClusters). Each node is associated with one of these clusters.

Overall, this code segment sets up the initial state of the nodes in the simulation, including their IDs, mobility status, speeds, positions, initial mobility information, and assignment to virtual clusters.



8 Now comes the plotting setup

Here's what each line does:

1. figure;: This command creates a new figure window for plotting. All subsequent plots will be displayed within this figure.

2. hold on;: This command enables the hold on feature, which allows multiple plots to be overlaid on the same figure without clearing the existing content. This is useful for visualizing the trajectories of multiple nodes over time.

3. title('MS-MAC Simulation with Virtual Clusters');: This sets the title of the plot to "MS-MAC Simulation with Virtual Clusters".

4. xlabel('X-axis');: This labels the x-axis as "X-axis".

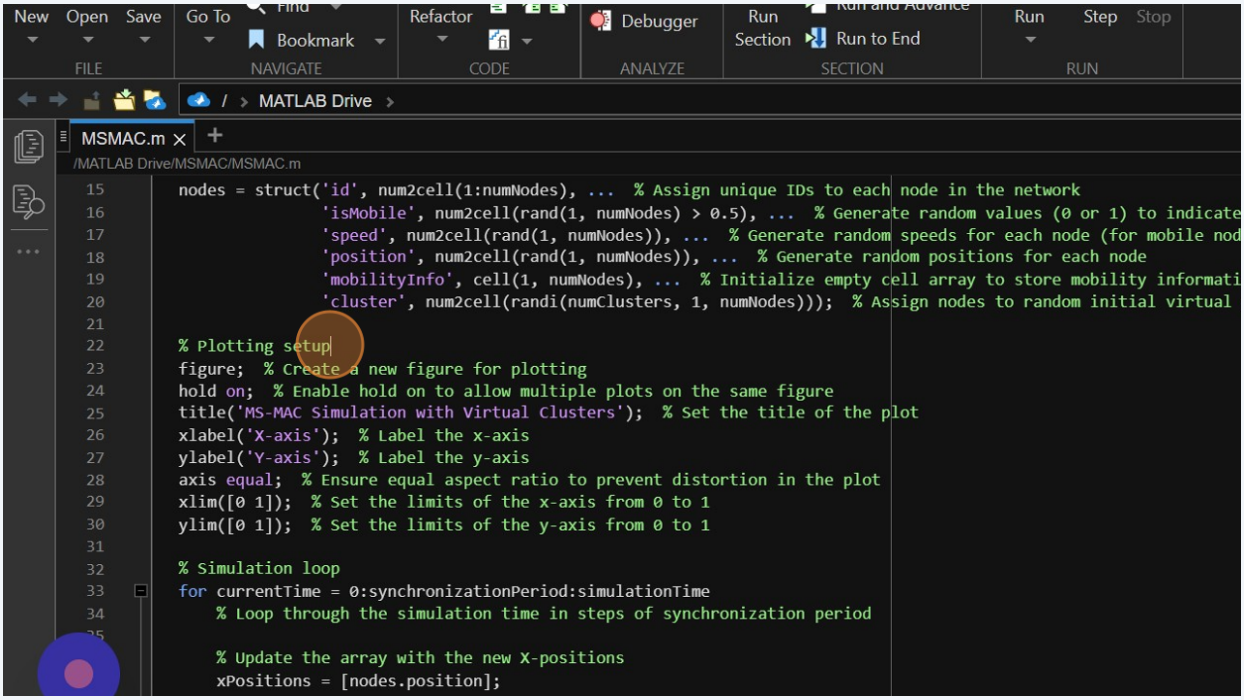
5. ylabel('Y-axis');: This labels the y-axis as "Y-axis".

6. axis equal;: This command ensures an equal aspect ratio in the plot, preventing distortion of shapes.

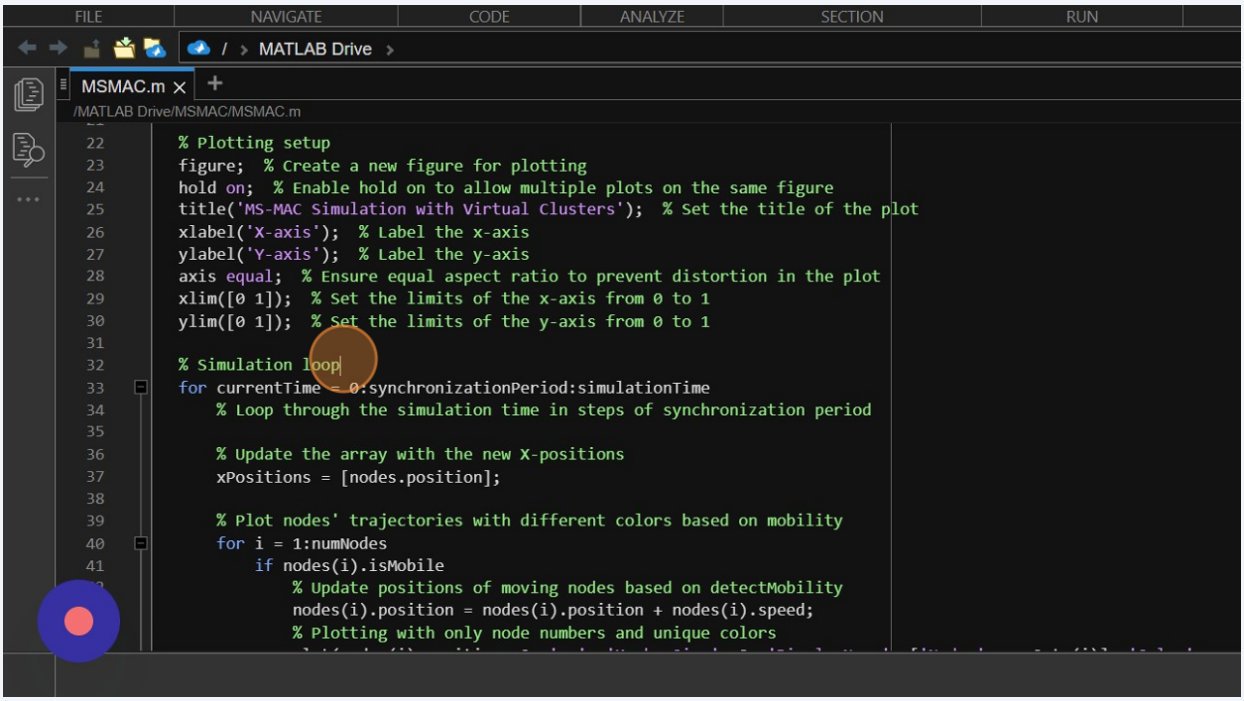
7. xlim([0 1]);: This sets the limits of the x-axis from 0 to 1, indicating that the x-coordinates of nodes will fall within this range.

8. ylim([0 1]);: This sets the limits of the y-axis from 0 to 1, indicating that the y-coordinates of nodes will fall within this range.

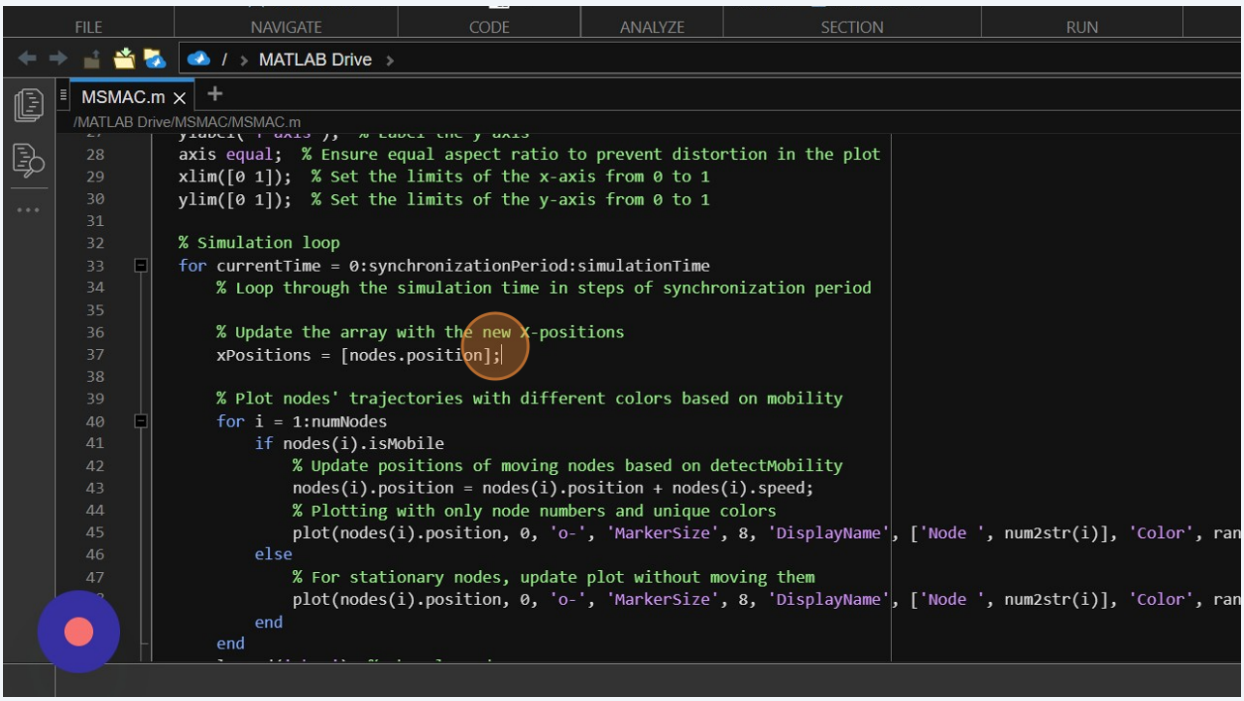
Overall, this setup prepares the plotting environment with appropriate labels, aspect ratio, and axis limits for visualizing the simulation results.



9 **This line initiates a loop that iterates through the simulation time in steps of the synchronization period** (synchronizationPeriod)**. It starts from time** 0 **and increments by** synchronizationPeriod **until it reaches or exceeds the** simulationTime**.**

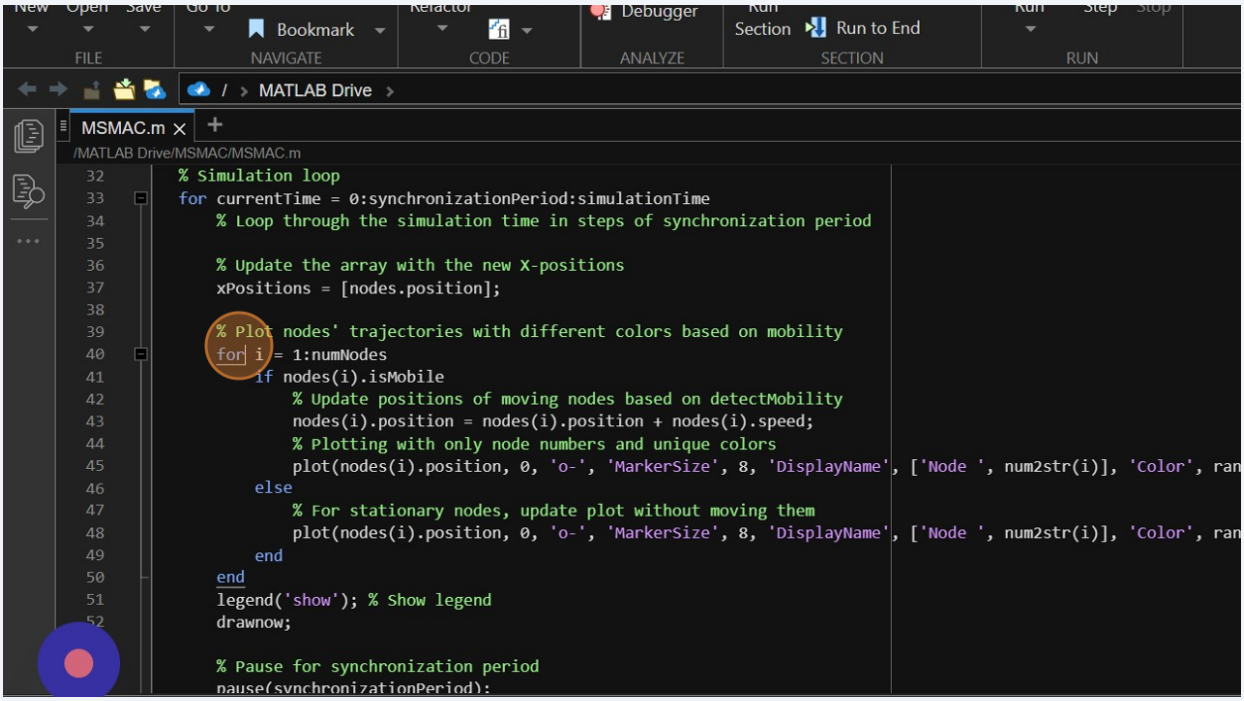


10 **Here,** xPositions **is updated to contain the x-coordinates of all nodes. This line collects the current positions of all nodes along the x-axis.**



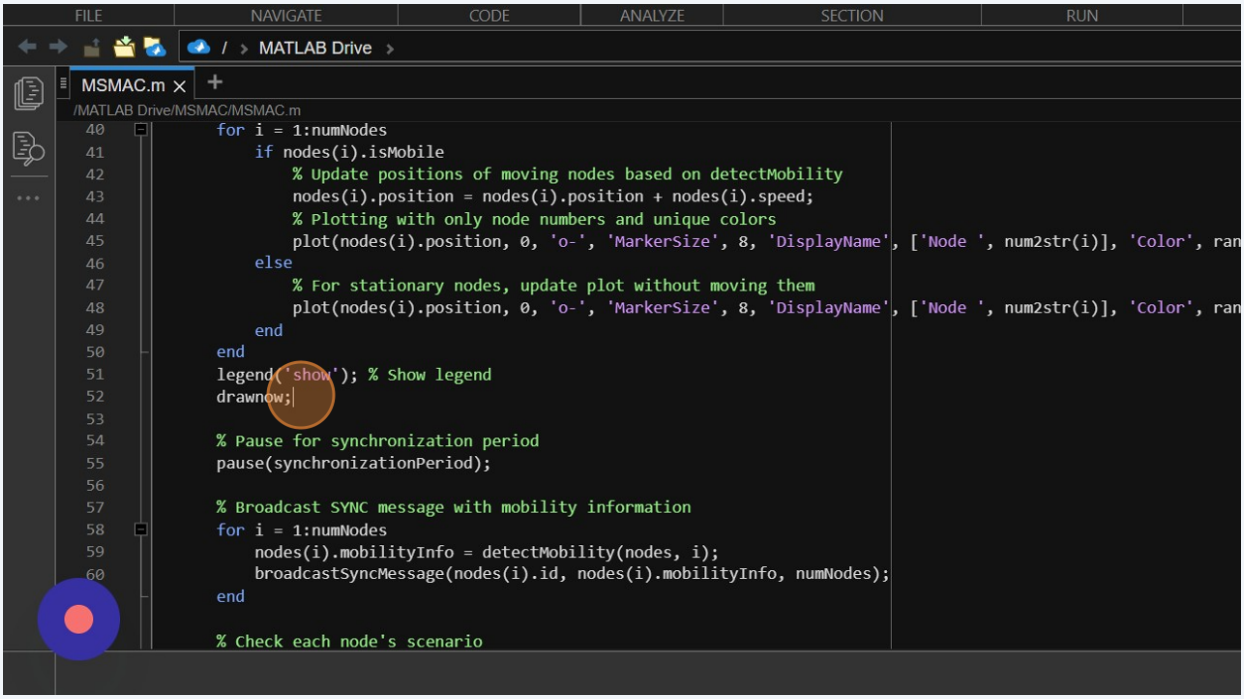
11 This loop iterates over each node in the network and updates their positions based on their mobility status. If a node is mobile (nodes(i).isMobile is true), its position is updated by adding its speed to the current position. Then, the node's trajectory is plotted on the graph using plot function with markers representing nodes, where mobile nodes are plotted with circles ('o-') and stationary nodes are also plotted but without moving them. Each node's trajectory is assigned a unique color for visualization using rand(1, 3) to generate a random RGB color. The DisplayName option adds a legend label with the node's ID for each plotted trajectory.

Overall, this section of the code drives the simulation forward, updating node positions and plotting their trajectories at each time step.



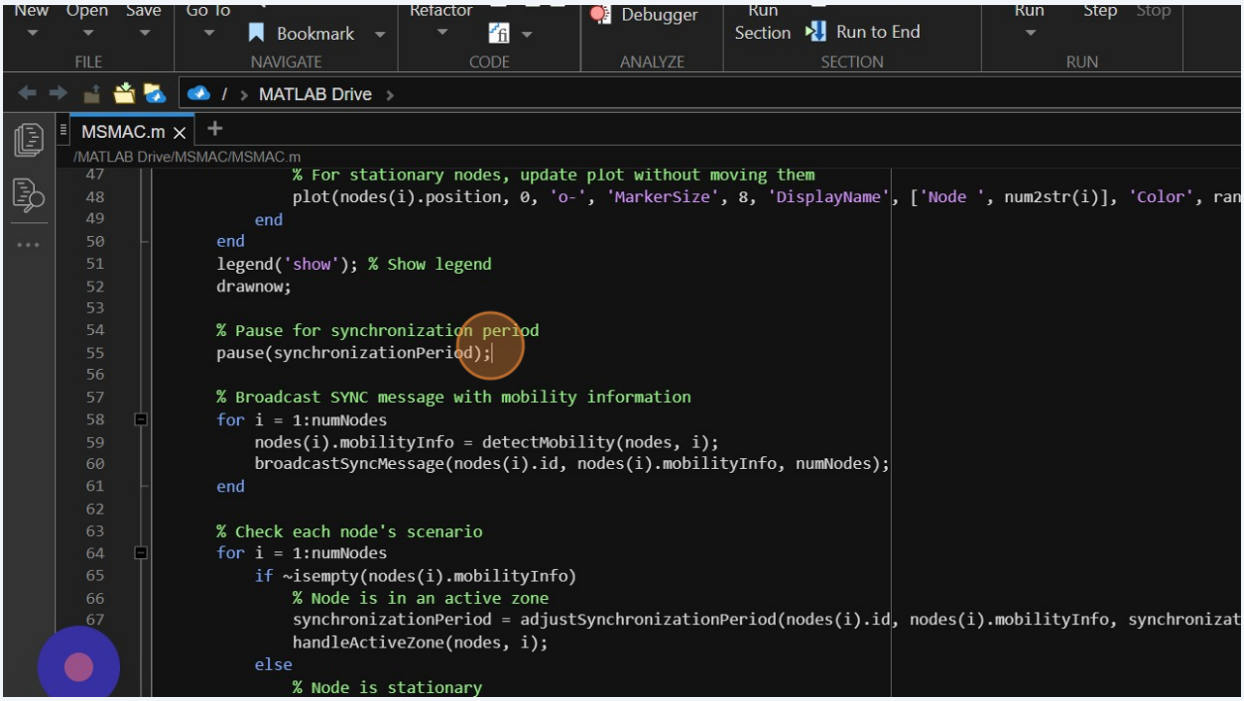
12 After plotting all the trajectories of the nodes in the simulation loop, the code calls the legend('show') function to display the legend on the plot. This function tells MATLAB to show the legend with the labels specified by the DisplayName property in the plot commands.

Then, the drawnow function is called. This function forces MATLAB to immediately update the figure window, ensuring that any changes made to the plot, such as adding the legend, are immediately displayed to the user.



13 This line of code pauses the execution of the simulation for the duration specified by the synchronization period (synchronizationPeriod).

During this pause, the simulation effectively waits for a specified amount of time before proceeding to the next iteration of the simulation loop. This pause ensures that the simulation progresses in discrete time steps, allowing for synchronized actions and updates to occur at regular intervals defined by the synchronization period.



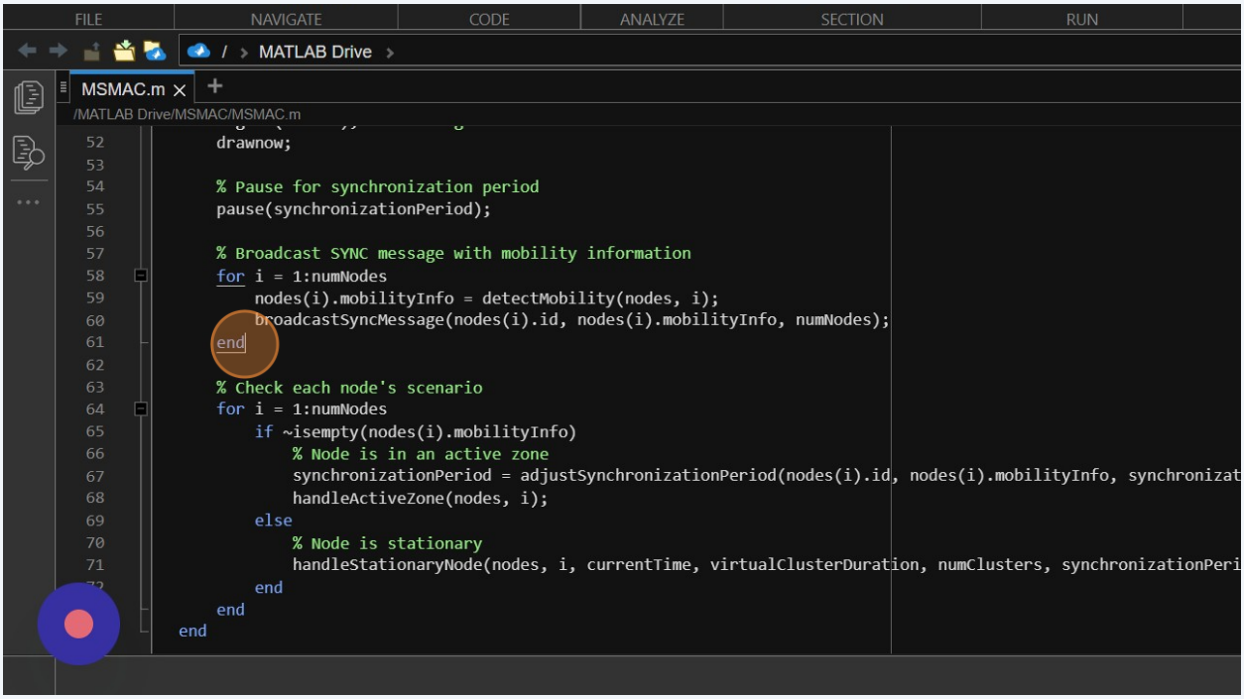
14 Here's what happens:

1. The loop iterates over each node in the network (numNodes).

2. For each node, it calls the detectMobility function to determine the mobility information of that node and assigns it to the mobilityInfo field of the node struct.

3. It then calls the broadcastSyncMessage function to broadcast the SYNC message with the node's ID and mobility information to all nodes in the network.

This part of the code ensures that each node receives the latest mobility information during synchronization, allowing them to adjust their behavior or parameters accordingly based on the current mobility state of the network.



15 Here's what's happening:

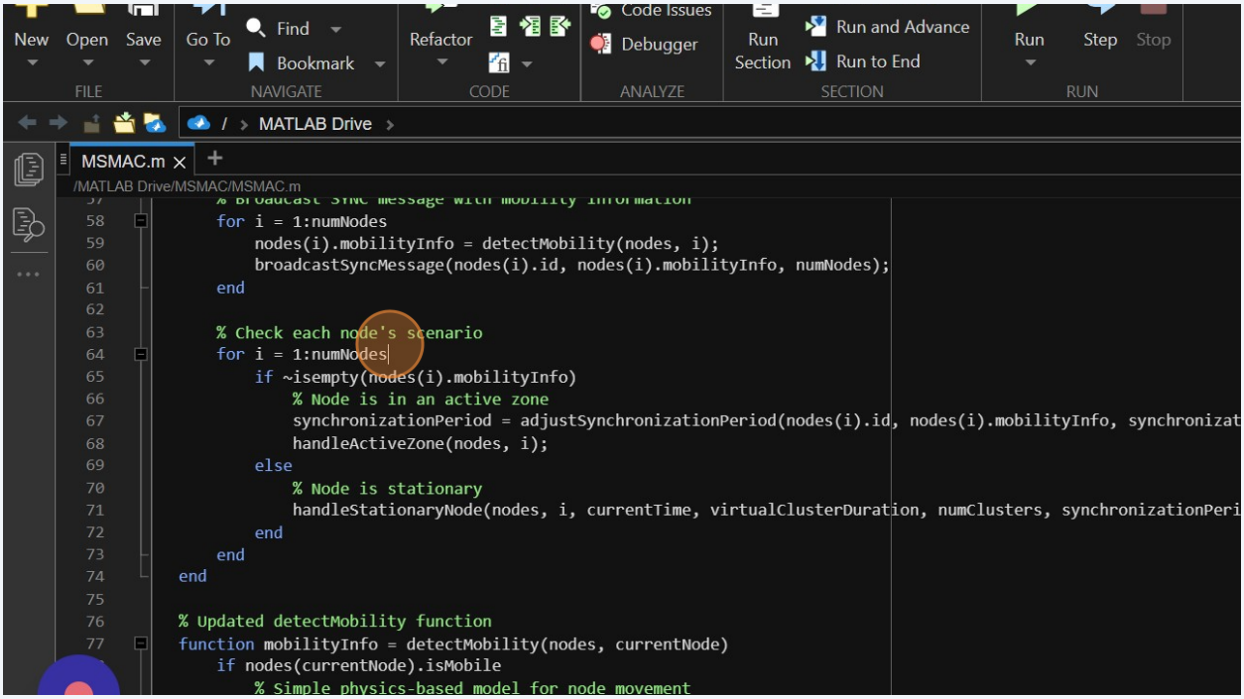
1. The loop iterates over each node in the network (numNodes).

2. For each node, it checks if the mobilityInfo field is not empty. If it's not empty, it means that the node is in motion and considered to be in an active zone.

3. If the node is in an active zone, it adjusts the synchronization period based on the node's mobility information by calling the adjustSynchronizationPeriod function. Then, it handles the active zone scenario by calling the handleActiveZone function.

4. If the node is stationary (i.e., mobilityInfo is empty), it handles the stationary node scenario by calling the handleStationaryNode function.

This part of the code ensures that each node's behavior is appropriately handled based on its mobility state, synchronization period adjustments are made as necessary, and actions specific to active zones or stationary nodes are performed accordingly

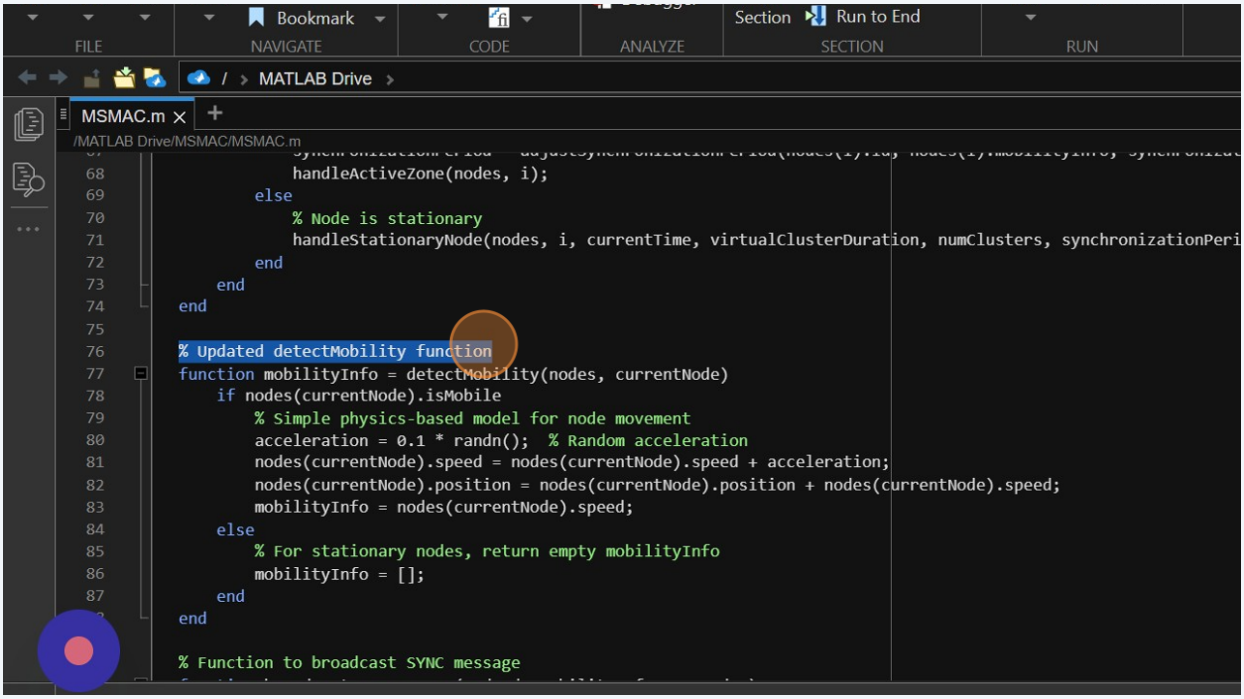


16 Here's how it works:

1. The function takes two arguments: nodes, which is the array of node structures representing the network, and currentNode, which is the index of the node whose mobility information needs to be determined.

2. It checks if the node (currentNode) is mobile by examining the isMobile field of the corresponding node structure in the nodes array.

3. If the node is mobile, it simulates node movement using a simple physics-based model. It generates a random acceleration value (acceleration) using a normal distribution with mean 0 and standard deviation 0.1. It then updates the node's speed by adding this acceleration value to the current speed. Finally, it updates the node's position based on its updated speed.

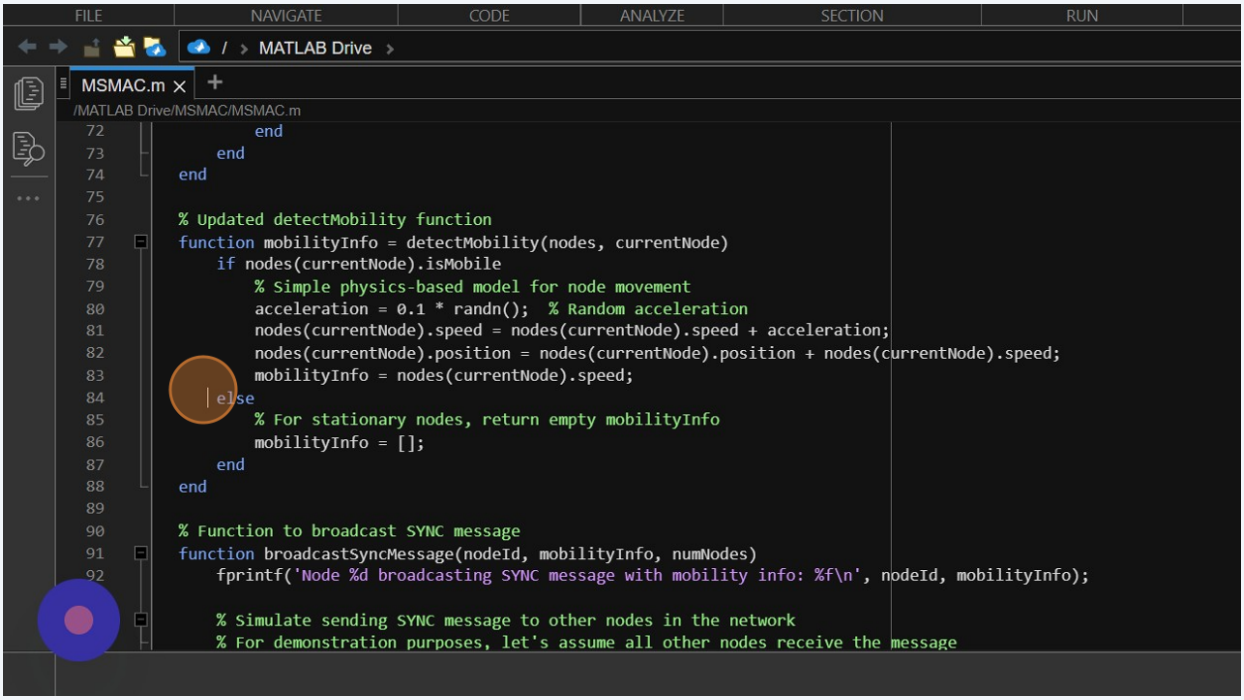


17

1. If the node is stationary, it returns an empty mobilityInfo, indicating that the node is not moving.

2. The function returns the mobility information (speed in this case) of the node. If the node is mobile, it returns the updated speed value; otherwise, it returns an empty array.

This function provides a basic simulation of node mobility, allowing for the determination of mobility information such as speed for each node in the network.



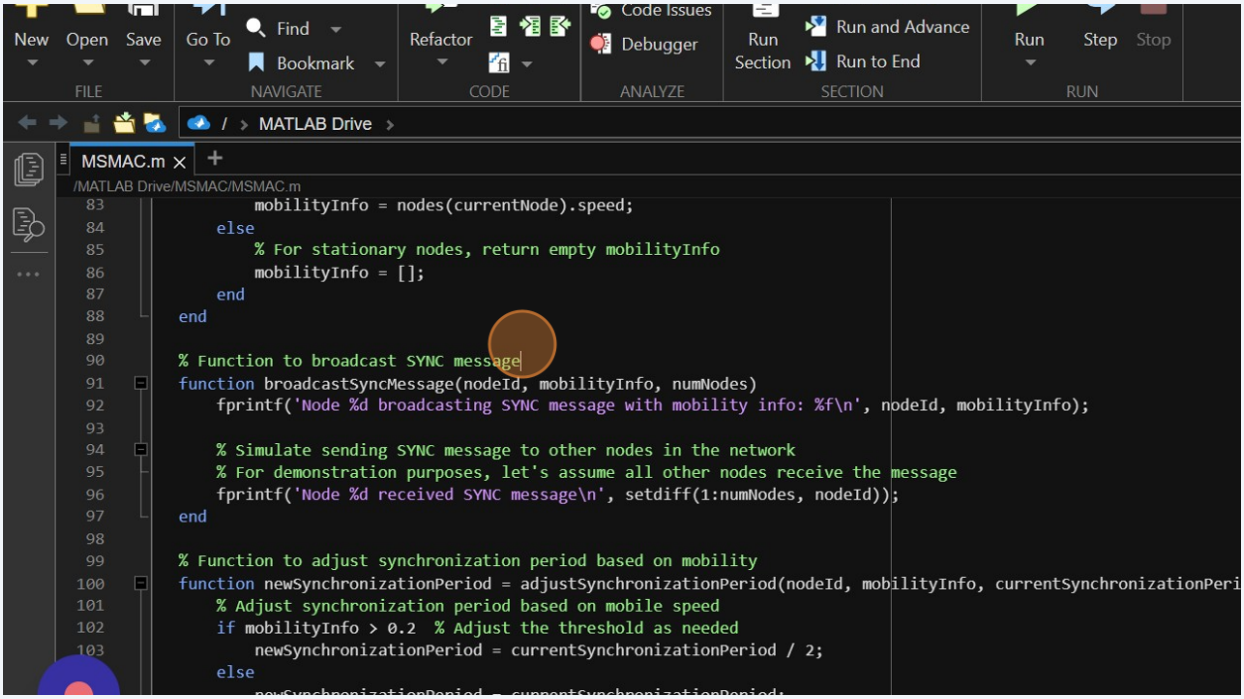
18

1. The function takes three arguments: nodeId, which is the ID of the node broadcasting the SYNC message, mobilityInfo, which is the mobility information being broadcasted, and numNodes, which is the total number of nodes in the network.

2. It prints a message indicating that the node with nodeId is broadcasting a SYNC message along with the mobility information (mobilityInfo). This message is printed to the console for debugging and informational purposes.

3. It simulates the reception of the SYNC message by all other nodes in the network. For demonstration purposes, it prints a message for each node in the network (excluding the broadcasting node) to indicate that it received the SYNC message. This is achieved using the setdiff function to obtain the set difference between all nodes in the network and the broadcasting node.

This function provides a simple simulation of the SYNC message broadcasting process, allowing for the dissemination of mobility information from one node to all other nodes in the network.



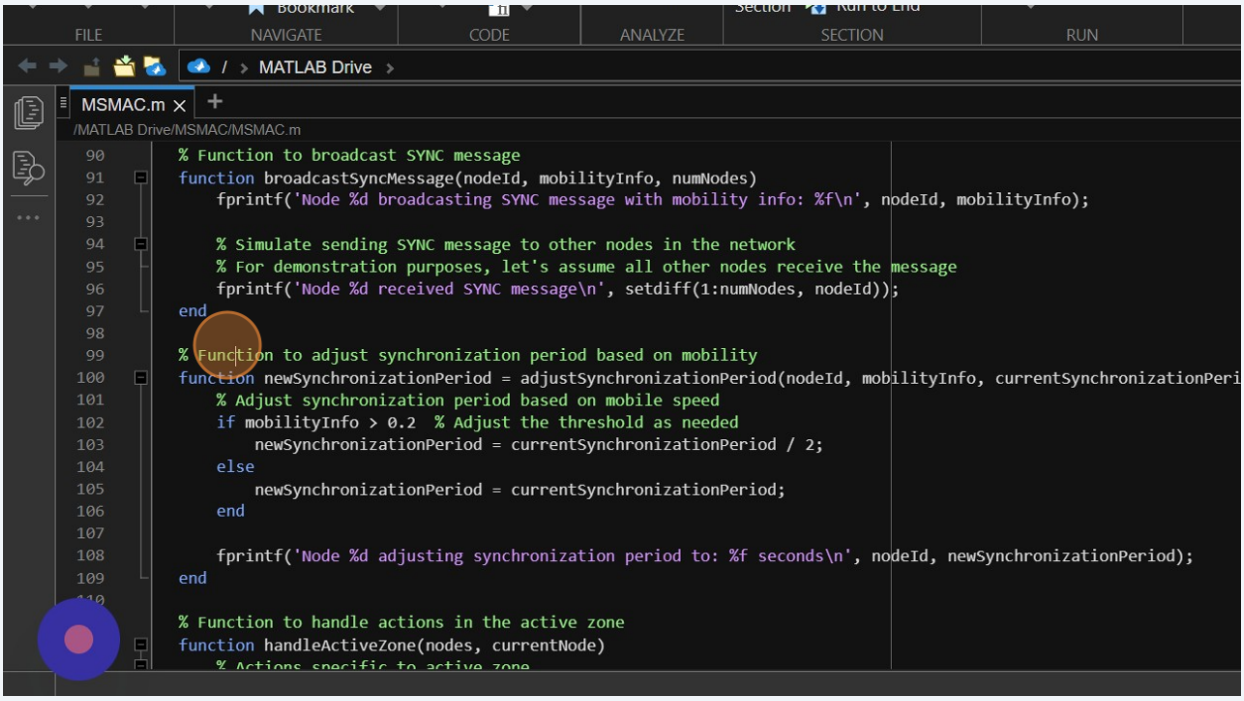
19

1. The function takes three arguments: nodeId, which is the ID of the node for which the synchronization period is being adjusted, mobilityInfo, which is the mobility information (in this case, speed) of the node, and currentSynchronizationPeriod, which is the current synchronization period being used in the simulation.

2. It adjusts the synchronization period based on the mobility information of the node. In this example, if the mobility information (speed) of the node is greater than 0.2, the synchronization period is halved (currentSynchronizationPeriod / 2). This means that if a node is moving faster (e.g., high speed), it requires more frequent synchronization to maintain coordination within the network. Otherwise, if the mobility information is less than or equal to 0.2, the synchronization period remains unchanged.

3. It prints a message indicating the adjustment of the synchronization period for the node. This message provides debugging and informational output to indicate the new synchronization period that has been set for the node.

This function allows for dynamic adjustment of the synchronization period based on the mobility characteristics of nodes in the network, facilitating efficient coordination and communication within the network.



20 Here's what this function does:

1. It takes two arguments: nodes, which is the array of node structures representing the network, and currentNode, which is the index of the node for which actions in the active zone need to be handled.

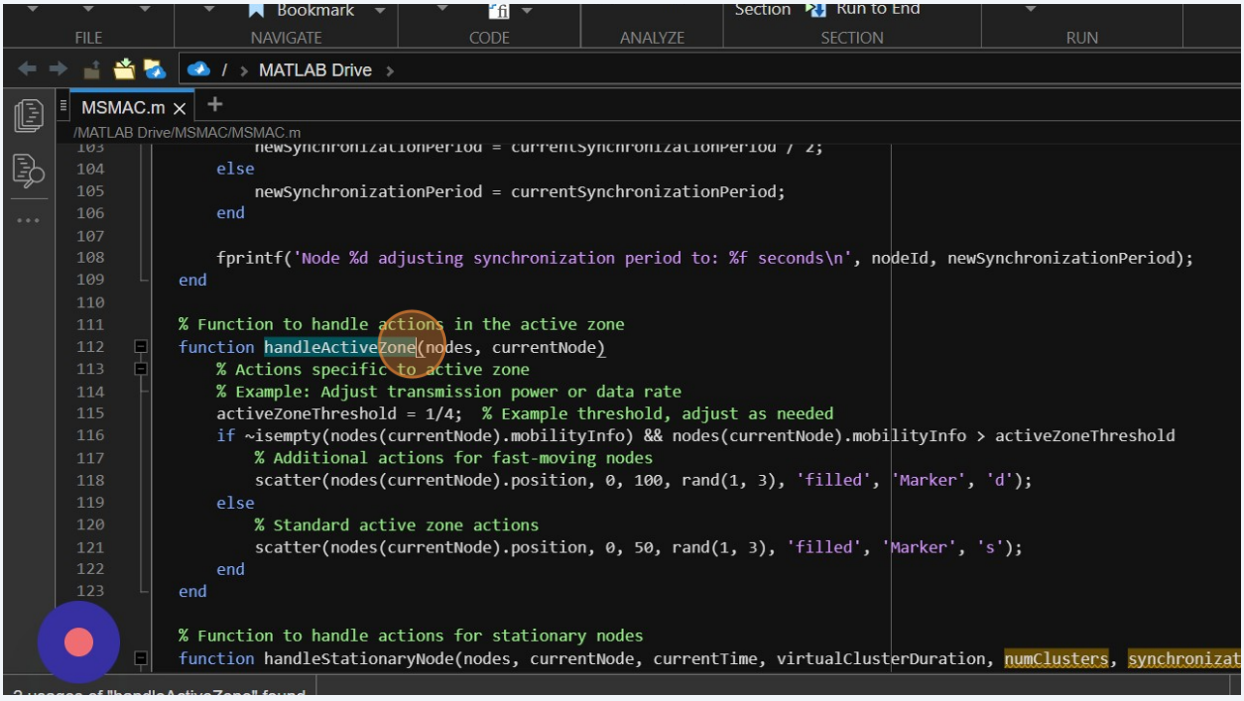
2. It defines the threshold for the active zone. In this example, activeZoneThreshold is set to 1/4. You can adjust this threshold based on the specific requirements of your simulation.

3. It checks if the node is in motion (i.e., mobilityInfo is not empty) and if its mobility information (speed) exceeds the active zone threshold.

4. If the node is in the active zone and moving fast (above the threshold), it performs additional actions specific to fast-moving nodes. In this example, it uses the scatter function to plot a diamond marker ('d') at the node's position with a size of 100 and a random color.

5. If the node is in the active zone but not moving fast (below the threshold), it performs standard active zone actions. In this example, it uses the scatter function to plot a square marker ('s') at the node's position with a size of 50 and a random color.

This function allows for handling different actions based on whether a node is in the active zone and its mobility characteristics, providing flexibility in adapting node behavior in response to varying conditions within the network



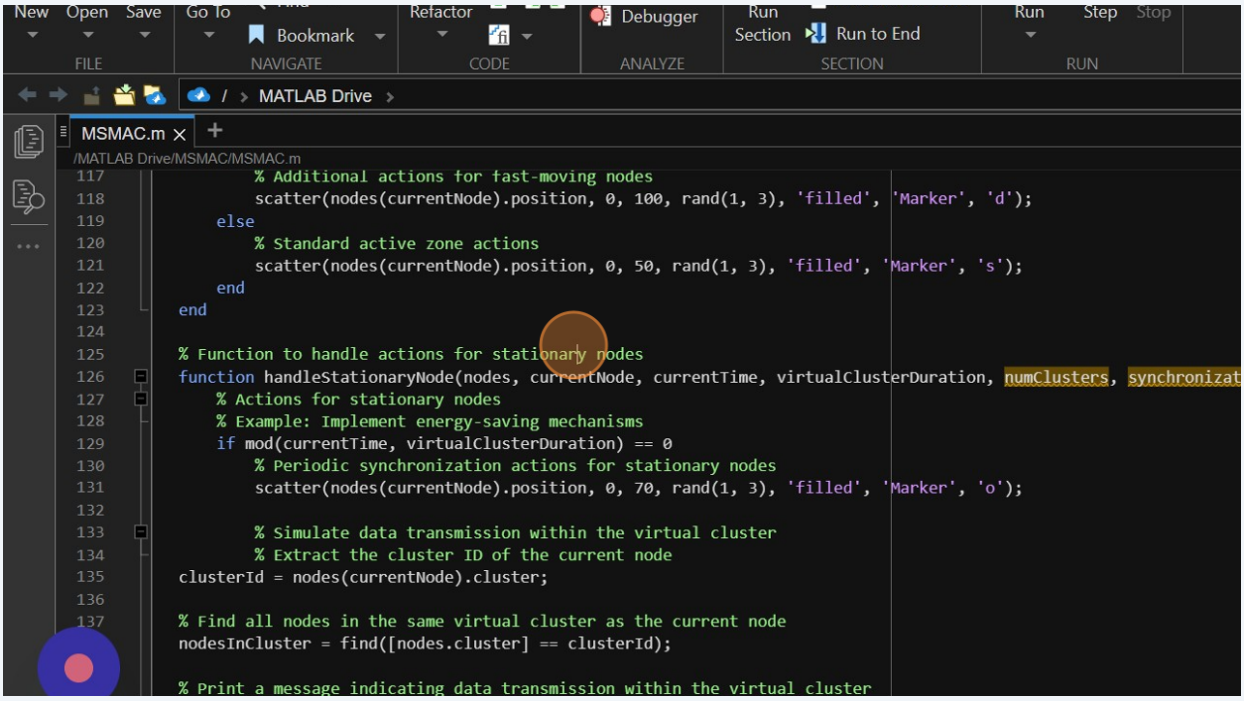
21 Here's what this function does:

1. It takes six arguments: nodes, which is the array of node structures representing the network, currentNode, which is the index of the stationary node for which actions need to be handled, currentTime, which is the current time in the simulation, virtualClusterDuration, which is the duration of virtual clusters, numClusters, which is the number of virtual clusters in the network, and synchronizationPeriod, which is the synchronization period used in the simulation.

2. It checks if the current time in the simulation (currentTime) is a multiple of the virtual cluster duration (virtualClusterDuration). This condition ensures that the actions within this block are executed periodically at intervals equal to the virtual cluster duration.

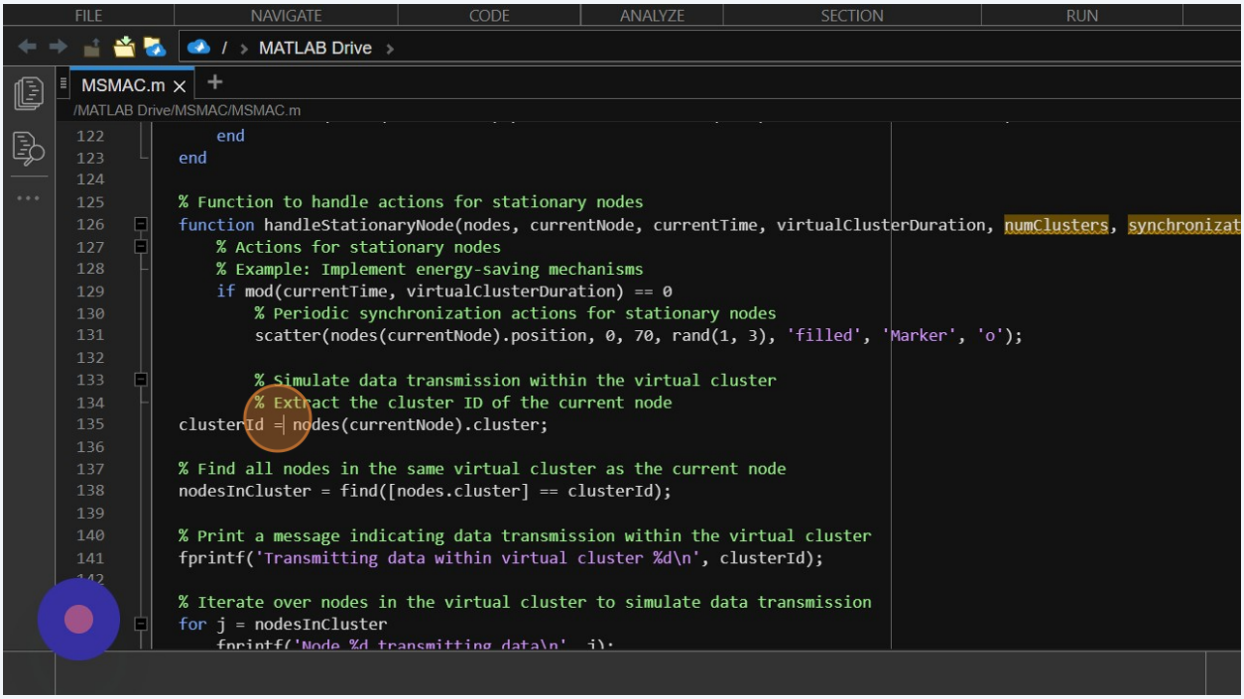
3. If the current time is a multiple of the virtual cluster duration, it performs periodic synchronization actions for stationary nodes. In this example, it uses the scatter function to plot a circular marker ('o') at the position of the stationary node with a size of 70 and a random color. This action could represent any periodic maintenance or synchronization tasks specific to stationary nodes.

Overall, this function allows for defining specific actions to be taken for stationary nodes in the network, such as periodic synchronization tasks or energy-saving mechanisms, and provides flexibility in customizing node behavior based on the requirements of the simulation



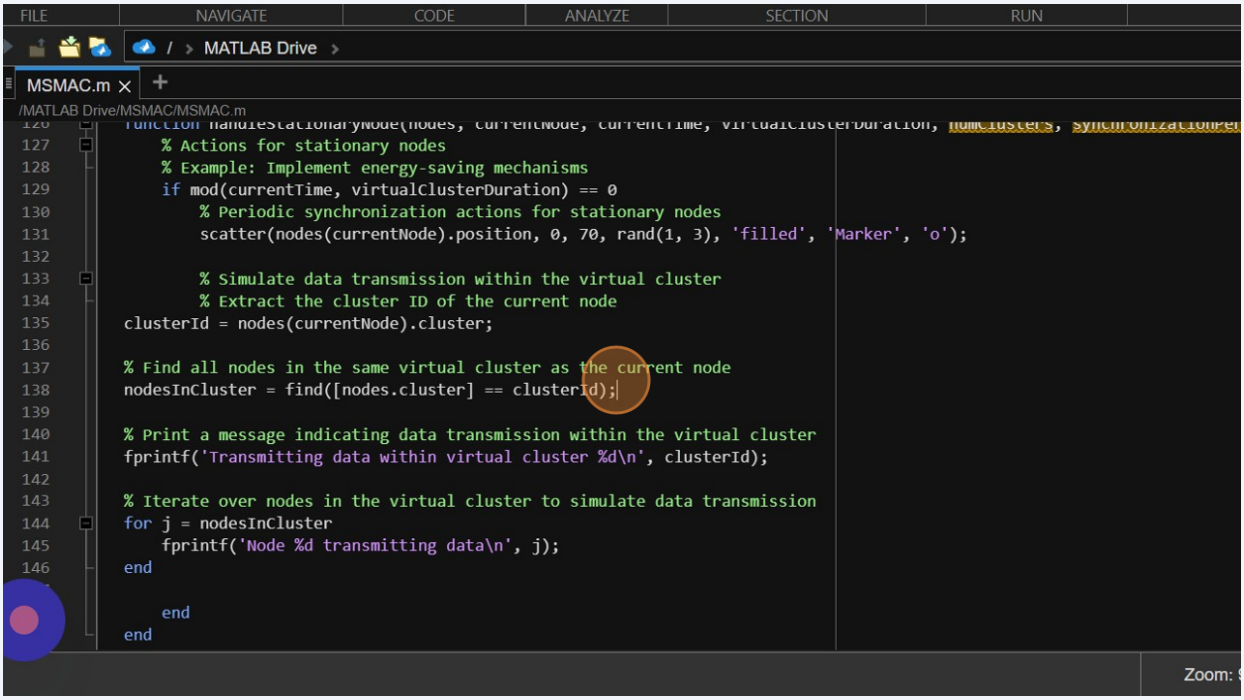
22

1. It then simulates data transmission within the virtual cluster to which the current node belongs. It extracts the cluster ID of the current node and proceeds to simulate data transmission within that virtual cluster. However, it appears that the implementation of this data transmission simulation is incomplete in the provided code snippet.

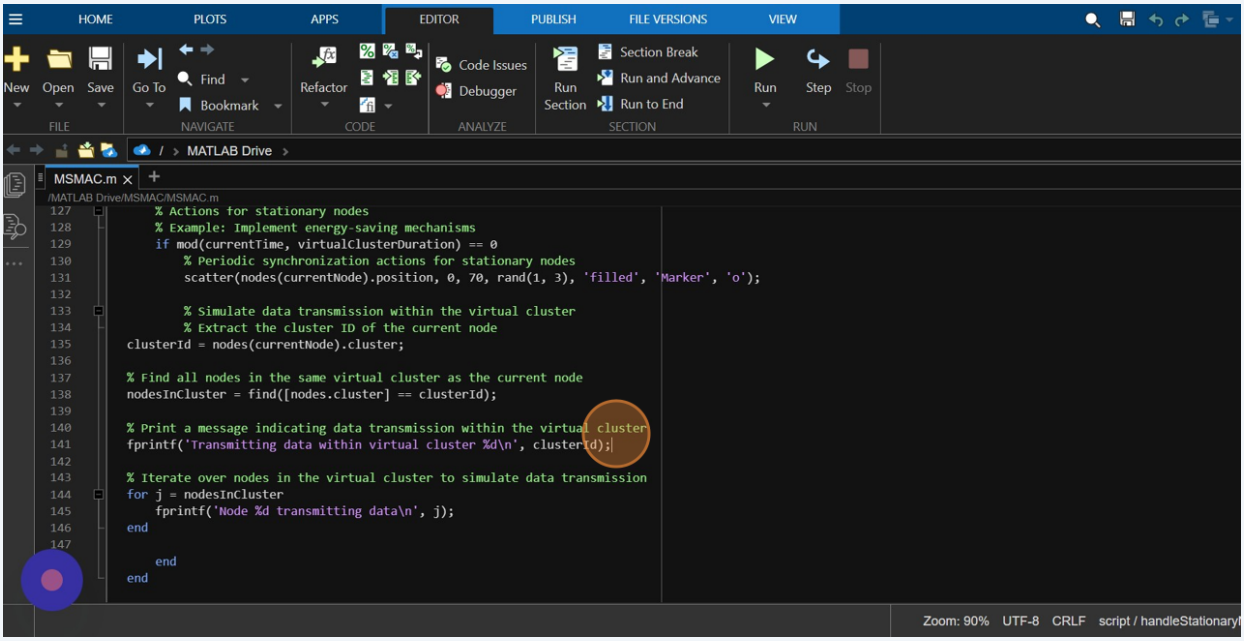


23

This line uses the find **function to identify all nodes in the** nodes **array whose** cluster **field matches the** clusterId **of the current node. The** [nodes.cluster] **construct creates a vector containing the cluster IDs of all nodes, and** find([nodes.cluster] == clusterId) **returns the indices of nodes whose cluster IDs match** clusterId

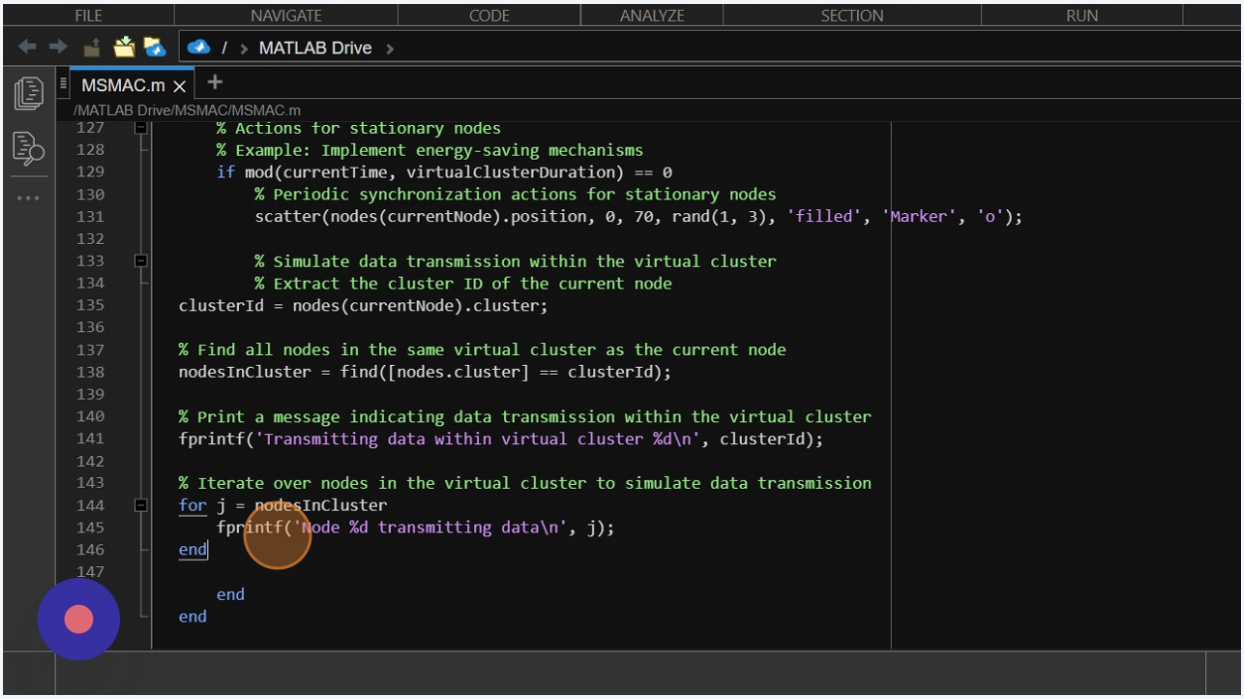


24 This line prints a message to the console indicating that data transmission is occurring within the virtual cluster with ID clusterId

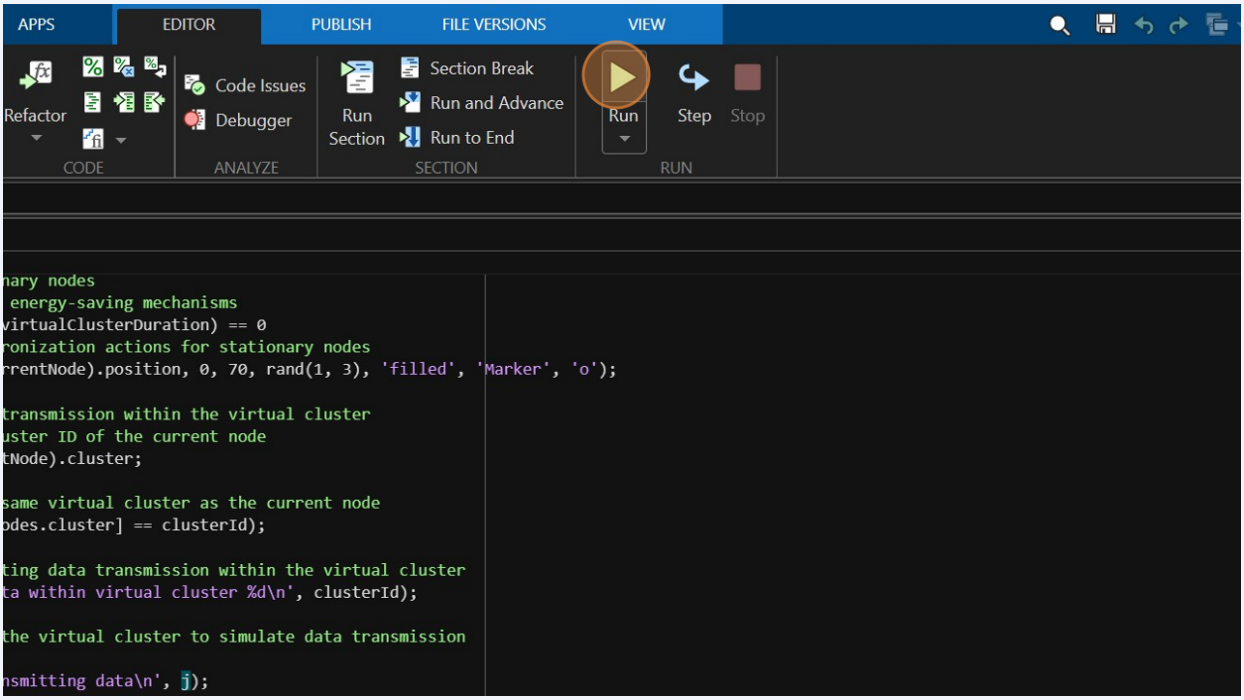


25 This loop iterates over each node in the virtual cluster (nodesInCluster) and prints a message indicating that each node is transmitting data. The loop variable j represents the index of each node in the virtual cluster.

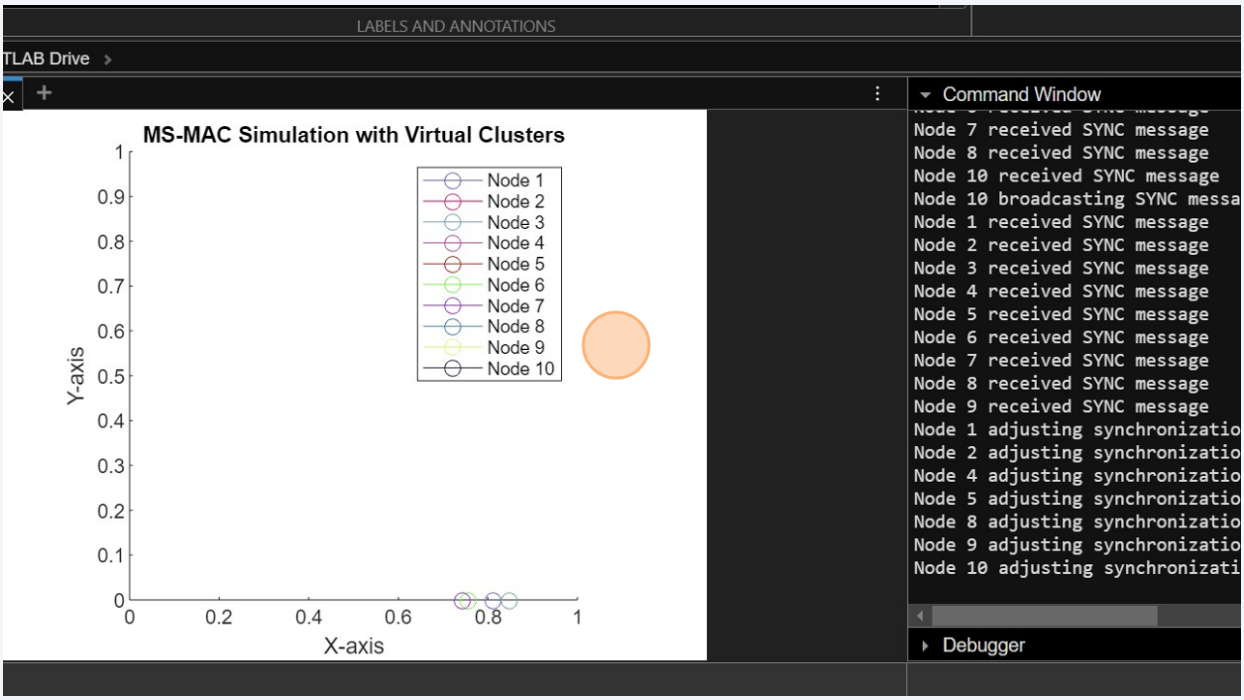
Overall, this section of the code simulates data transmission within the virtual cluster to which the current node belongs by identifying all nodes in that cluster, printing a message to indicate data transmission, and iterating over each node to simulate transmission.



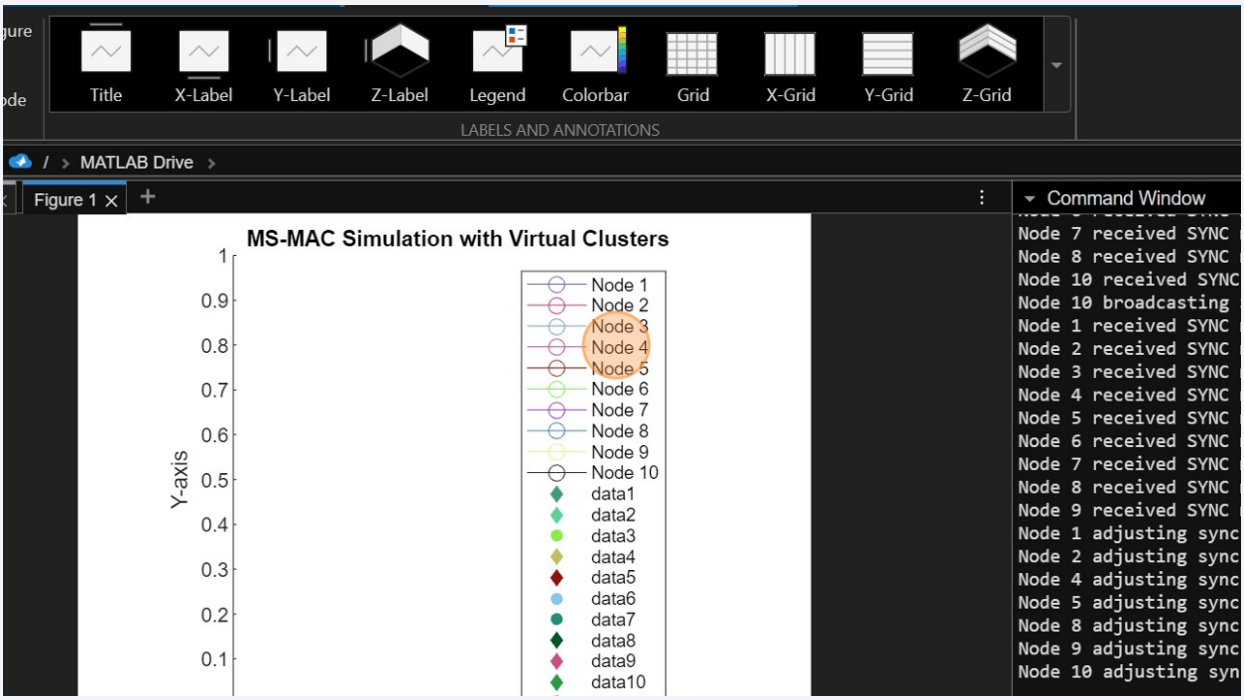
26 Click this icon. to run the simulation of MS-MAC Protocol



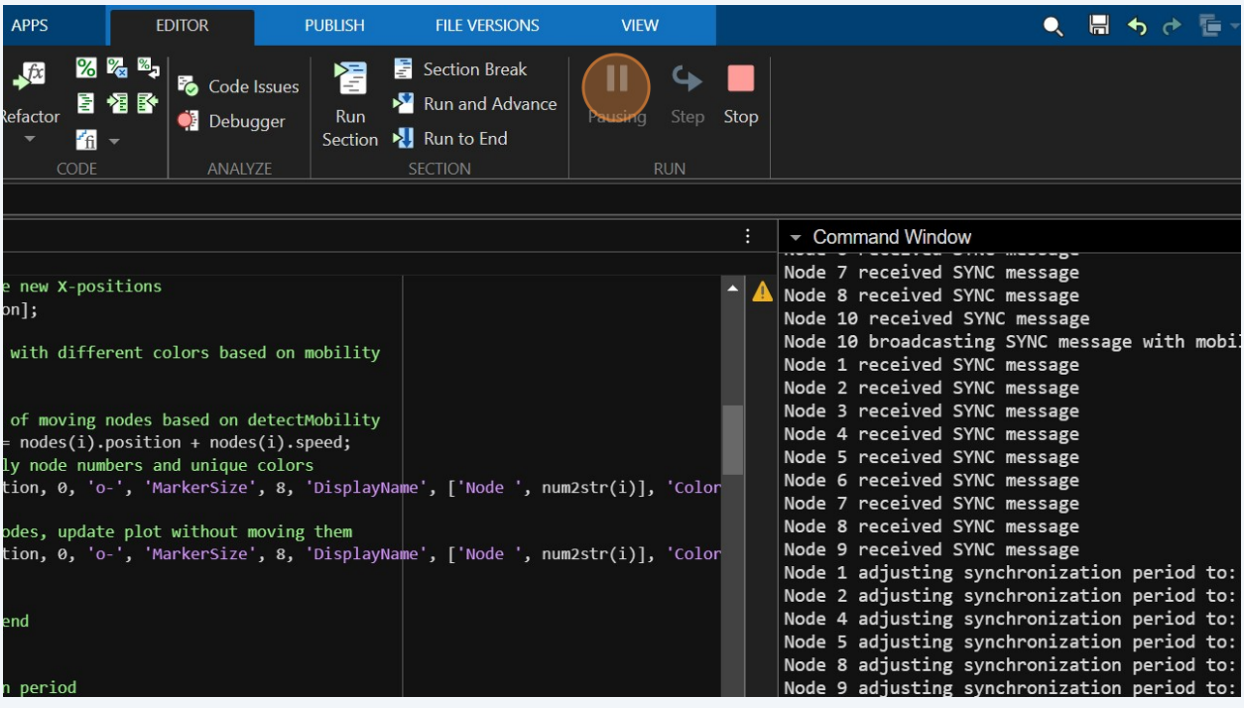
27 Output figure and commands will be like this



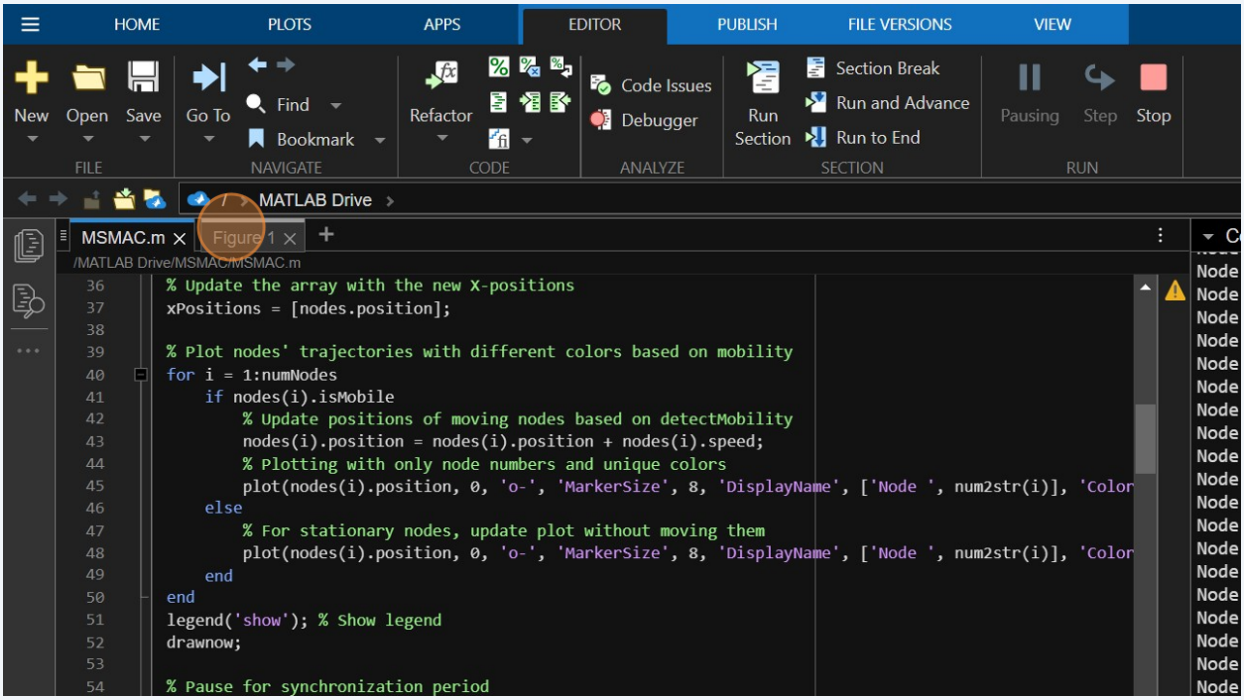
28 Here you can see initial 10 nodes that we previously set in the numofNodes initialization and on the right command window has the process steps that how and which nodes are sending and receiving messages.



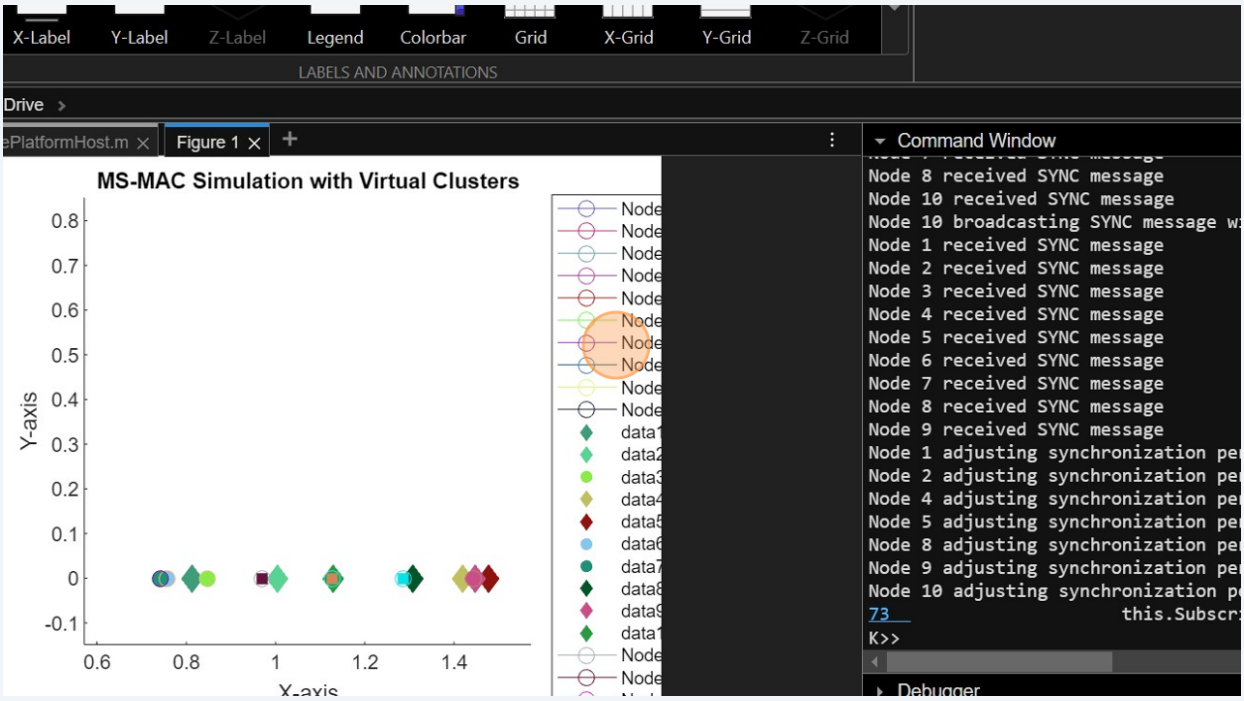
29 Lets pause it to observe some steps



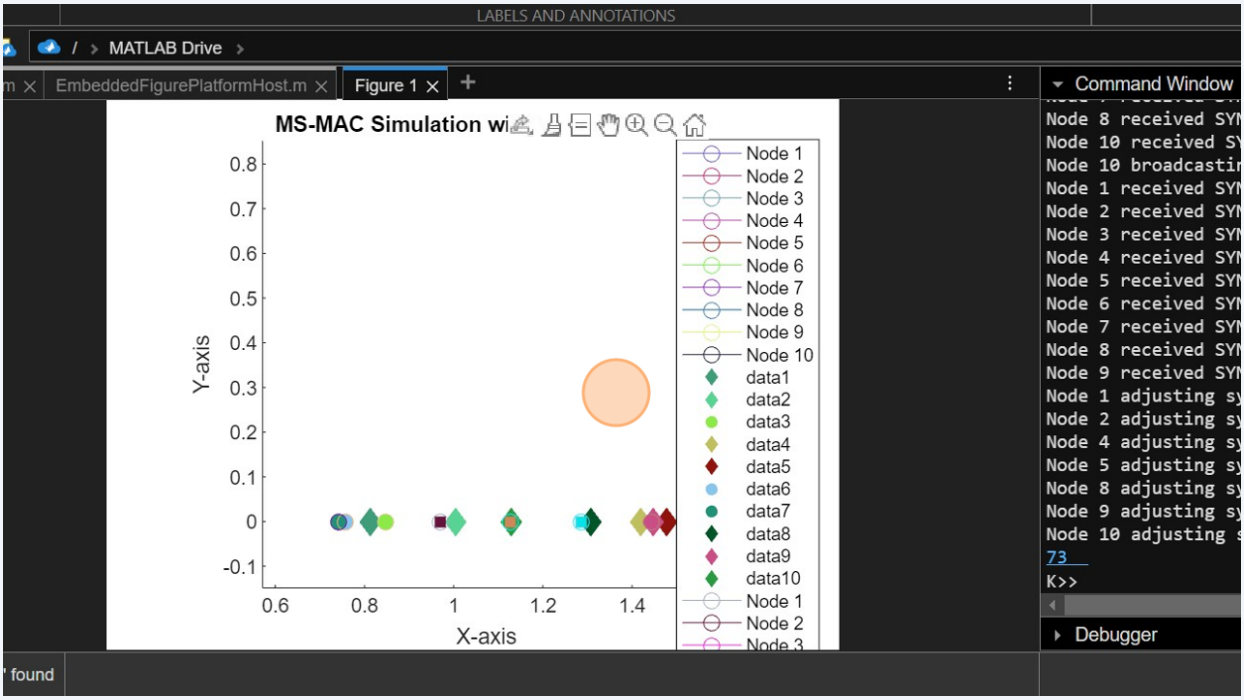
30 Moving back to the figure



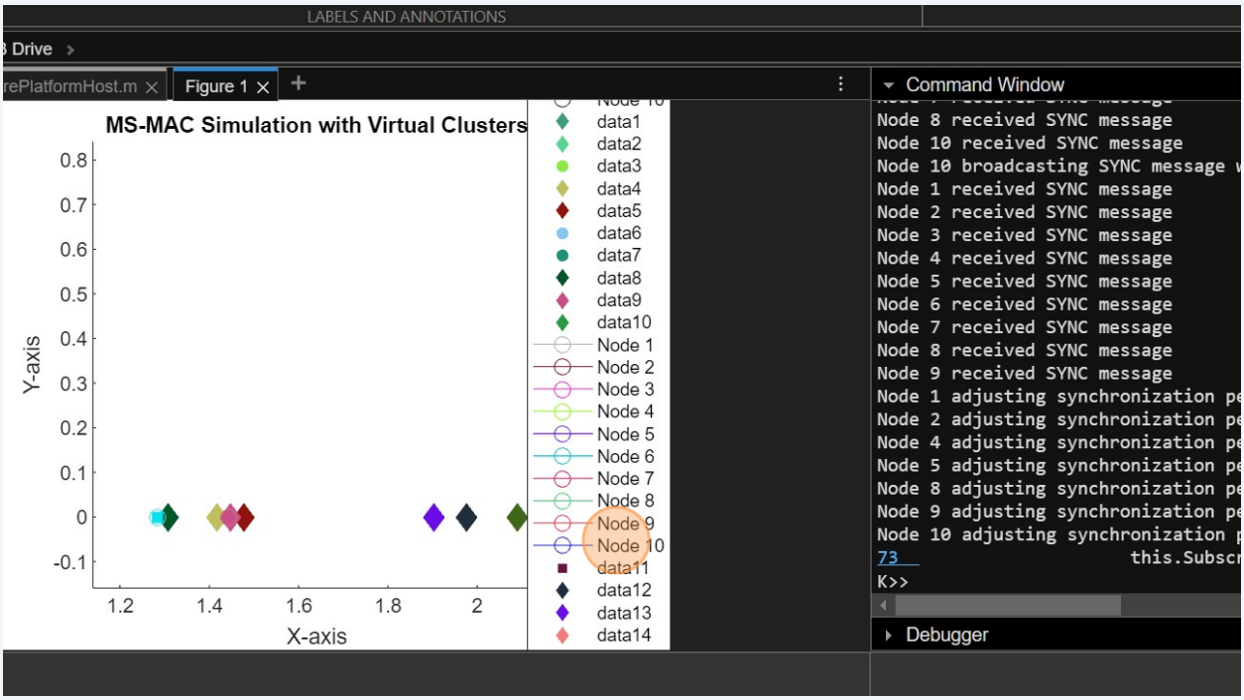
31 Here you can see the nodes as well as the data (message) send and received.



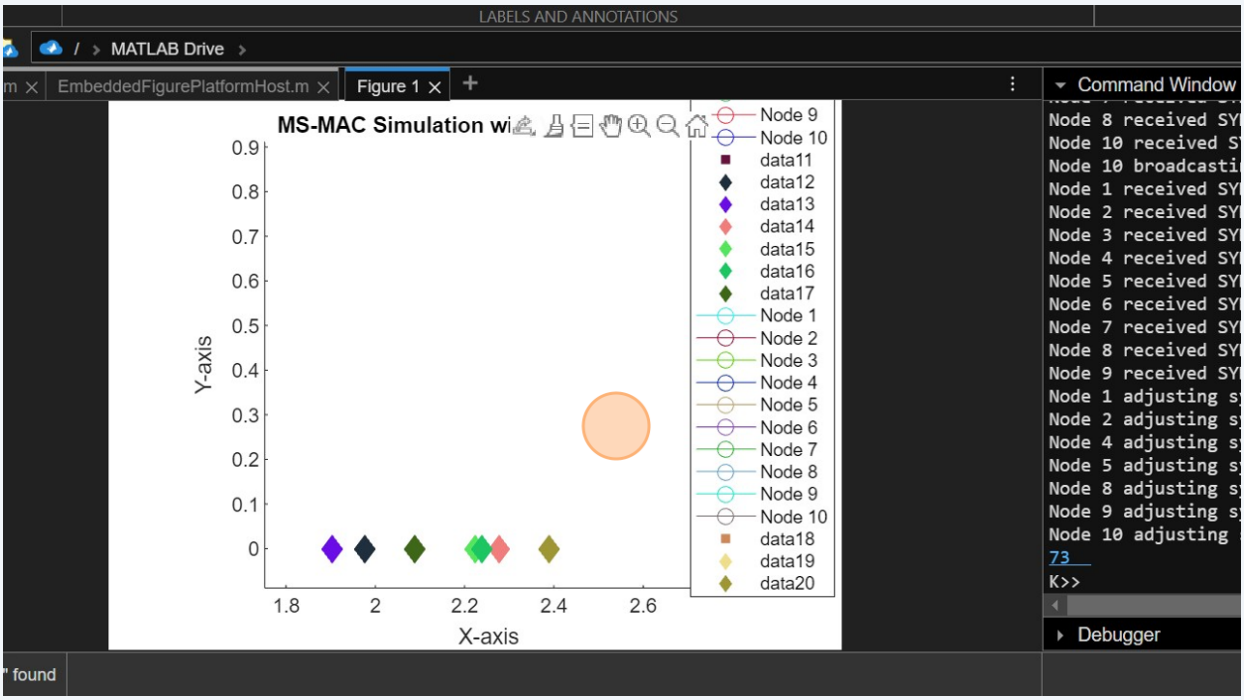
32 Here in the figure you can see each node with its unique color to represent the node and also the data in each node is uniquely displayed



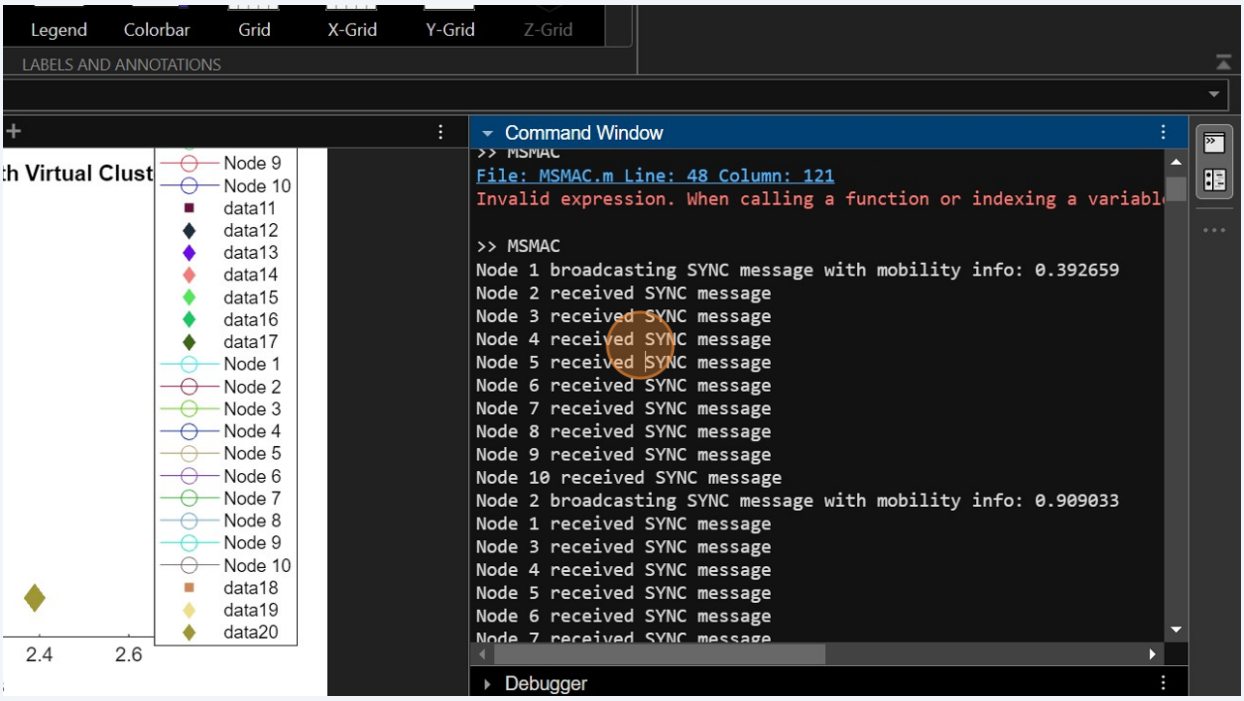
33 You can observe the different color of nodes and data of nodes



34 The mobile nodes keep moving with time and make new virtual clusters and share the data between the nodes of same cluster that we will also see in the command window in next steps

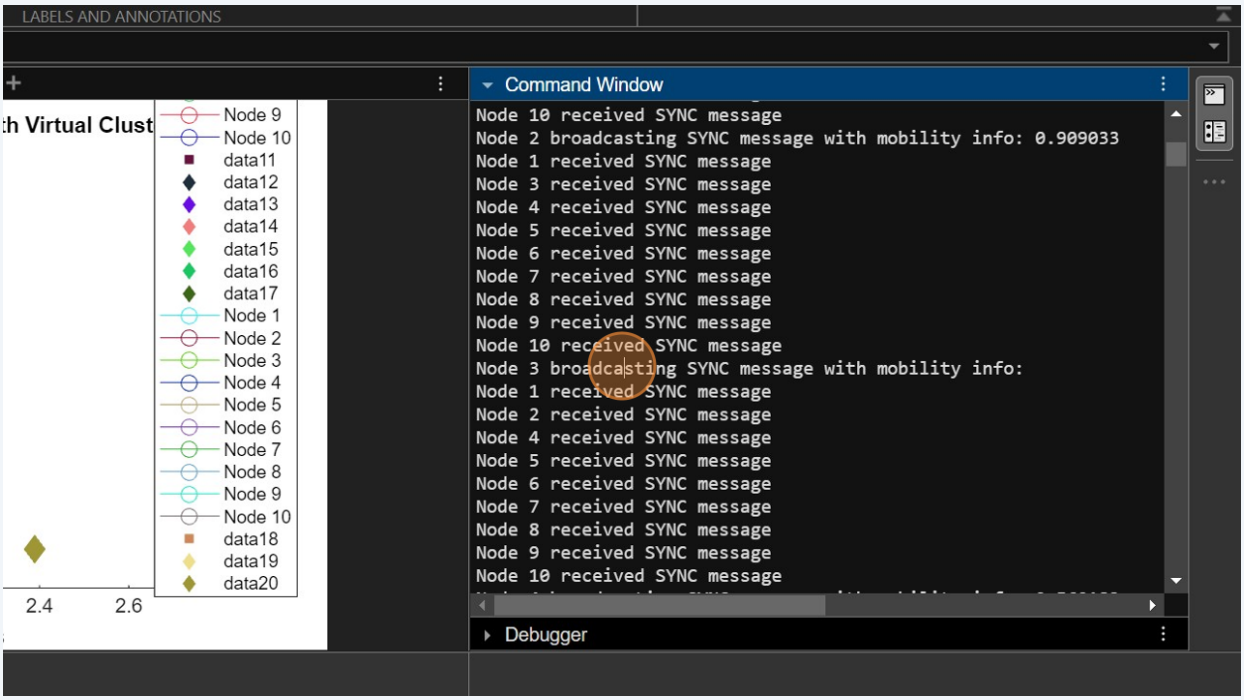


35 Here at the start you can see Node 1 broadcasting SYNC message with mobility info: 0.392659 and other nodes are receiving the msg
Node 2 received SYNC message
Node 3 received SYNC message
Node 4 received SYNC mes..."

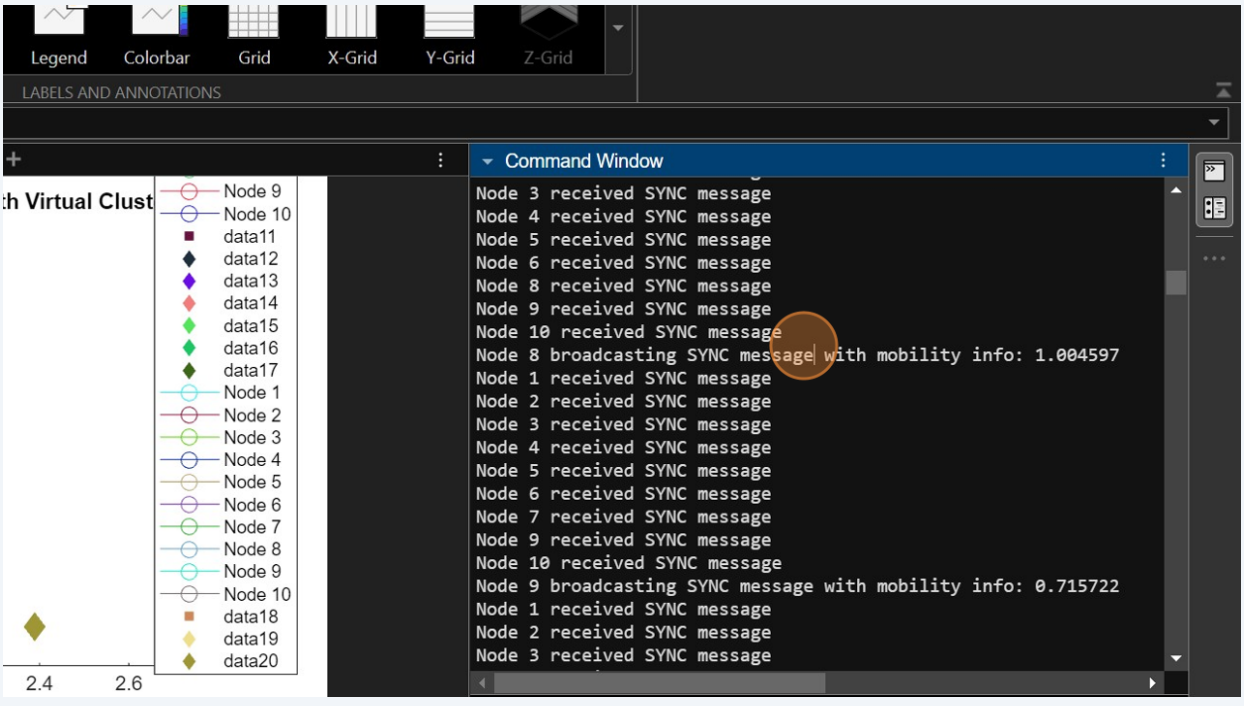


36 Similarly for the other nodes as well

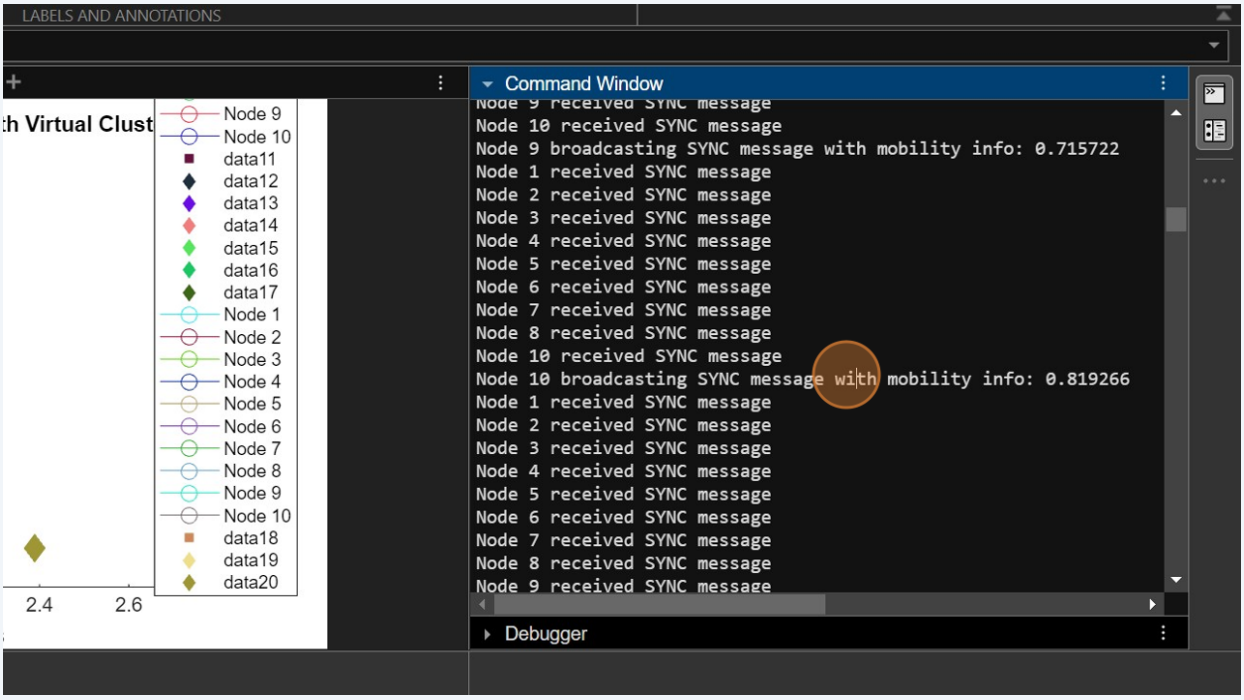
and the node that is static that dont display any mobility info



37 Then the nodes also broadcasts the SYNC message with the mobility info in case of mobile nodes

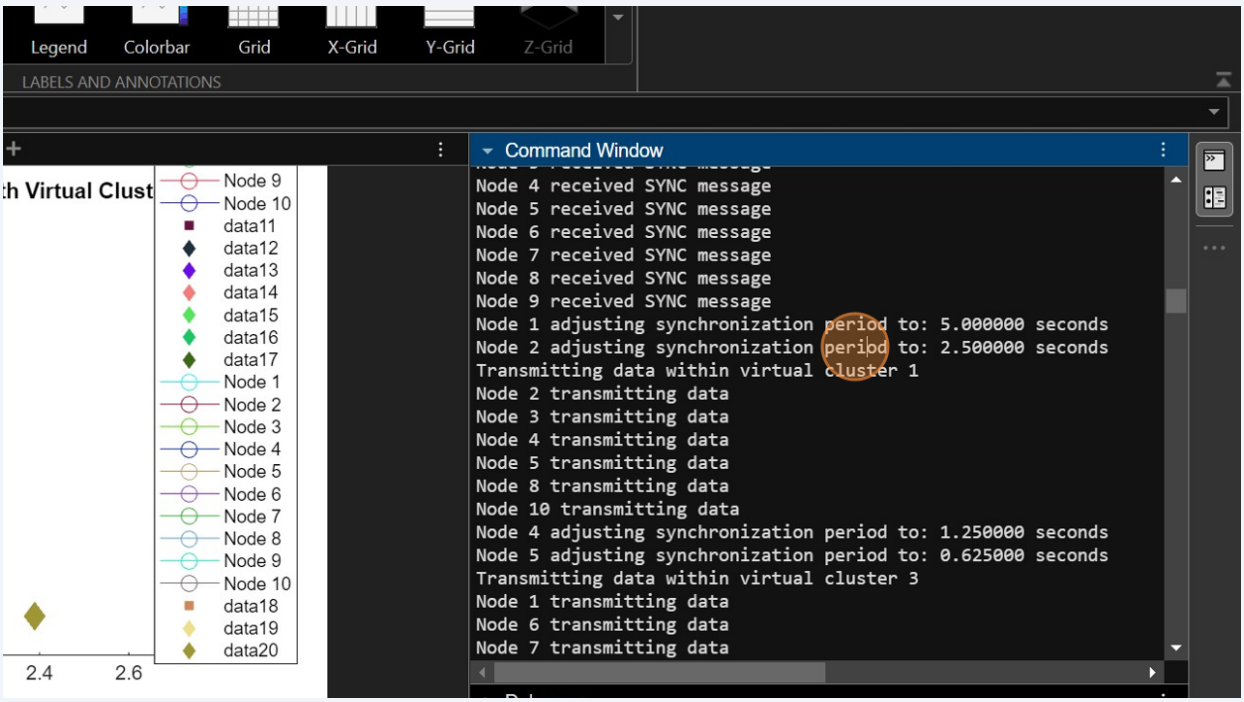


38 And remaining nodes receive the message

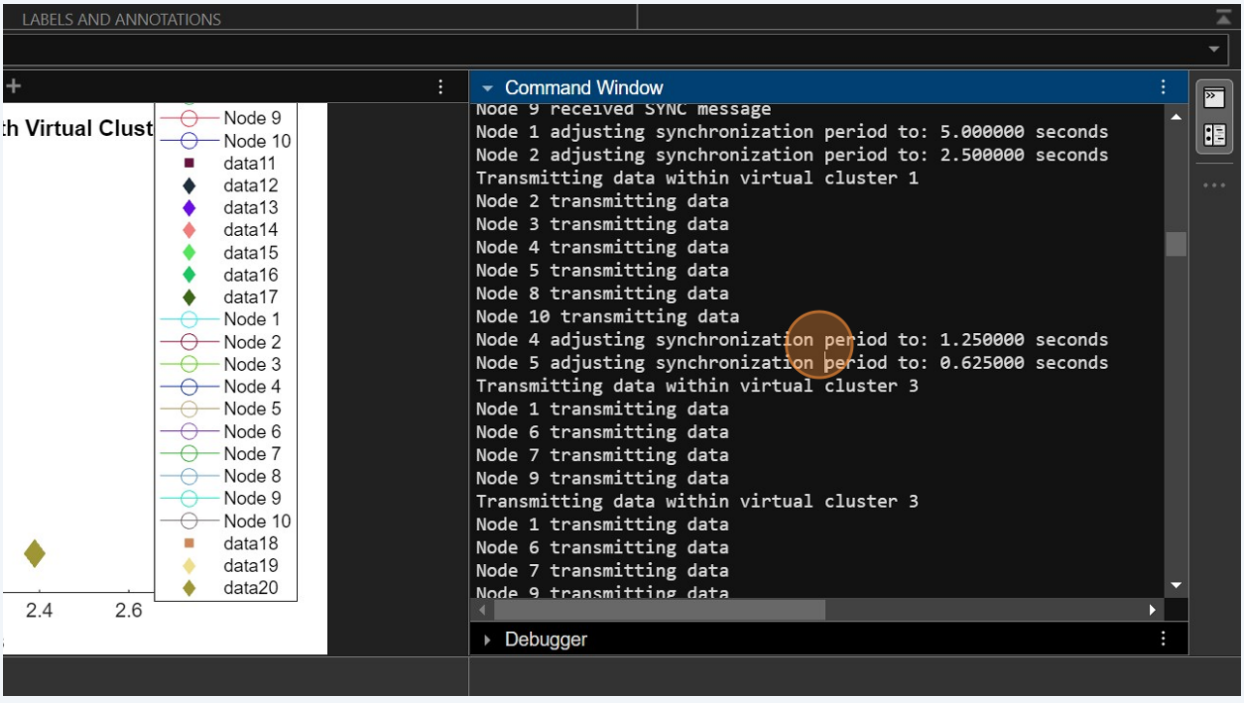


39 Here you can see the Node 2 adjusting synchronization period to: 2.500000 seconds and the info that which node is transmitting data and in which cluster is shown:

Transmitting data within virtual cluster 1
Node 2 transmitting data
Node 3 transmit..."



40 Here in the virtual cluster 3 the data is transmitted by nodes



41

Here is the code :

```
% MS-MAC IMPLEMENTATION AND SIMULATION WITH VIRTUAL CLUSTERS

% Constants
synchronizationPeriod = 10; % Synchronization period in seconds
activeZoneThreshold = 1/4; % Example threshold for active zone, adjust as needed
virtualClusterDuration = 120; % Duration of virtual clusters in seconds
connectionSetupTime = 10; % Connection setup time in seconds

% Simulation parameters
simulationTime = 600; % Total simulation time in seconds
numNodes = 10; % Number of nodes in the network
numClusters = 3; % Number of virtual clusters

% Initialize nodes
nodes = struct('id', num2cell(1:numNodes), ... % Assign unique IDs to each node in the network
'isMobile', num2cell(rand(1, numNodes) > 0.5), ... % Generate random values (0 or 1) to indicate if a node is mobile
'speed', num2cell(rand(1, numNodes)), ... % Generate random speeds for each node (for mobile nodes)
'position', num2cell(rand(1, numNodes)), ... % Generate random positions for each node
'mobilityInfo', cell(1, numNodes), ... % Initialize empty cell array to store mobility information for each node
'cluster', num2cell(randi(numClusters, 1, numNodes))); % Assign nodes to random initial virtual clusters

% Plotting setup
figure; % Create a new figure for plotting
hold on; % Enable hold on to allow multiple plots on the same figure
title('MS-MAC Simulation with Virtual Clusters'); % Set the title of the plot
xlabel('X-axis'); % Label the x-axis
ylabel('Y-axis'); % Label the y-axis
axis equal; % Ensure equal aspect ratio to prevent distortion in the plot
xlim([0 1]); % Set the limits of the x-axis from 0 to 1
ylim([0 1]); % Set the limits of the y-axis from 0 to 1

% Simulation loop
for currentTime = 0:synchronizationPeriod:simulationTime
% Loop through the simulation time in steps of synchronization period
```

% Update the array with the new X-positions xPositions = [nodes.position]; % Plot nodes' trajectories with different colors based on mobility for i = 1:numNodes if nodes(i).isMobile % Update positions of moving nodes based on detectMobility nodes(i).position = nodes(i).position + nodes(i).speed; % Plotting with only node numbers and unique colors plot(nodes(i).position, 0, 'o-', 'MarkerSize', 8, 'DisplayName', ['Node ', num2str(i)], 'Color', rand(1, 3)); else % For stationary nodes, update plot without moving them plot(nodes(i).position, 0, 'o-', 'MarkerSize', 8, 'DisplayName', ['Node ', num2str(i)], 'Color', rand(1, 3))); end end legend('show'); % Show legend drawnow; % Pause for synchronization period pause(synchronizationPeriod); % Broadcast SYNC message with mobility information for i = 1:numNodes nodes(i).mobilityInfo = detectMobility(nodes, i); broadcastSyncMessage(nodes(i).id, nodes(i).mobilityInfo, numNodes); end % Check each node's scenario for i = 1:numNodes if ~isempty(nodes(i).mobilityInfo) % Node is in an active zone synchronizationPeriod = adjustSynchronizationPeriod(nodes(i).id, nodes(i).mobilityInfo, synchronizationPeriod); handleActiveZone(nodes, i); else % Node is stationary handleStationaryNode(nodes, i, currentTime, virtualClusterDuration, numClusters, synchronizationPeriod); end end

end

% Updated detectMobility function
function mobilityInfo = detectMobility(nodes, currentNode)
if nodes(currentNode).isMobile
% Simple physics-based model for node movement
acceleration = 0.1 * randn(); % Random acceleration
nodes(currentNode).speed = nodes(currentNode).speed + acceleration;
nodes(currentNode).position = nodes(currentNode).position +
nodes(currentNode).speed;
mobilityInfo = nodes(currentNode).speed;
else
% For stationary nodes, return empty mobilityInfo
mobilityInfo = [];
end
end

% Function to broadcast SYNC message
function broadcastSyncMessage(nodeId, mobilityInfo, numNodes)
fprintf('Node %d broadcasting SYNC message with mobility info: %f\n', nodeId, mobilityInfo);

`% Simulate sending SYNC message to other nodes in the network % For demon