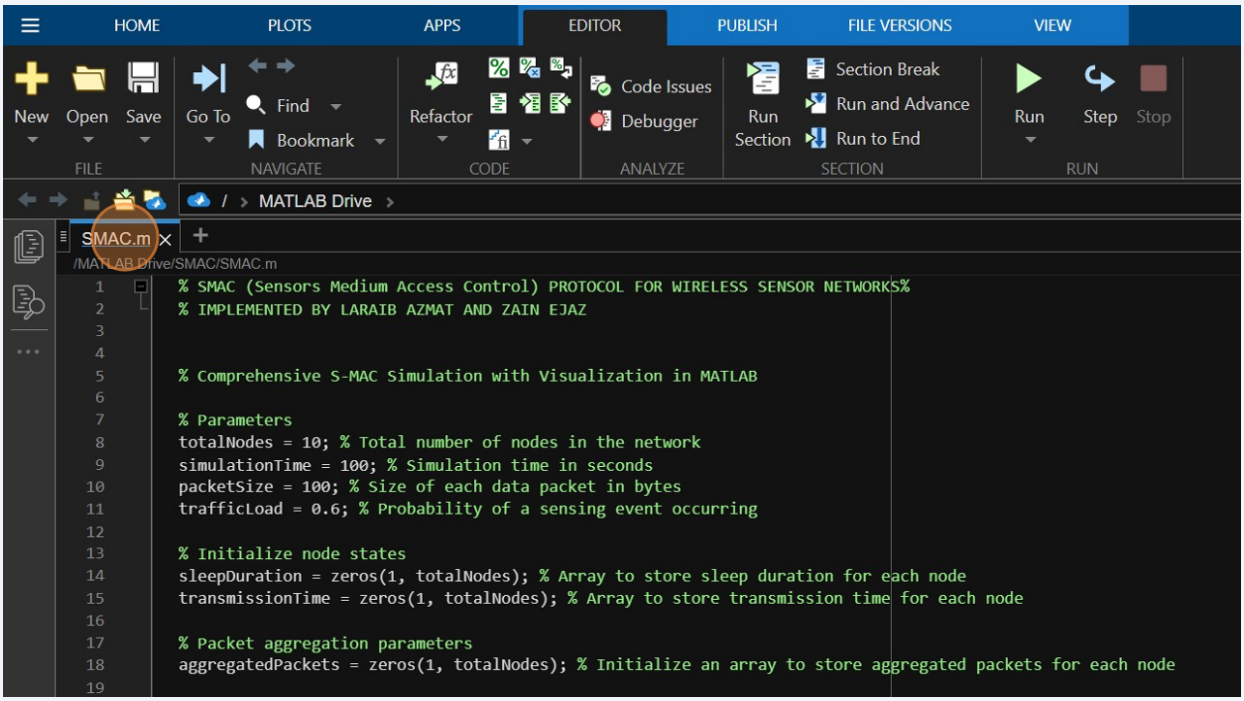
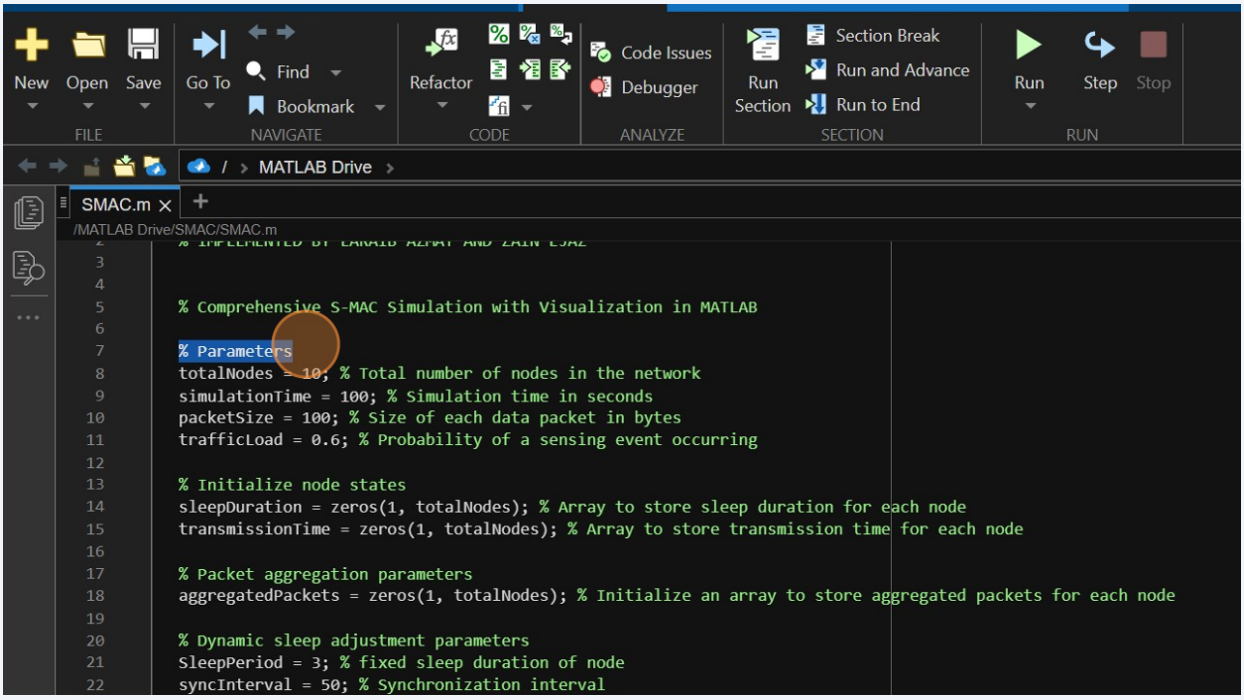
Step-by-step Implementation of SMAC for network simulation on MATLAB (with code)

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

2 We are going to Implement SMAC So First Lets know about SMAC: **The Sensor-MAC (S-MAC)** protocol is a specialized medium access control (MAC) protocol tailored for the distinct requirements of wireless sensor networks (WSNs). Prioritizing energy optimization, S-MAC introduces a strategic **duty cycling mechanism**, enabling nodes to alternate between **active and sleep states**, effectively curbing unnecessary energy consumption during periods of inactivity and **alleviating the common issue of idle listening** found in traditional MAC protocols. In the simulated environment, each sensor node adheres to a predefined duty cycle, periodically awakening for synchronization with neighboring nodes. This synchronization involves the exchange of control packets—SYNC for periodic alignment, **RTS** to signal data transmission intent, and **CTS** for confirmation. Implemented with fixed sleep and awake periods, the protocol ensures a consistent sleep-wake cycle, enhancing energy management for prolonged network longevity. The communication during wake periods unfolds in three phases: SYNC for synchronization, RTS signaling data transmission, and CTS confirming readiness. This meticulous design minimizes energy consumption and addresses hidden terminal problems through the **RTS/CTS handshake**, ensuring efficient and collision-free communication in resource-constrained wireless sensor networks



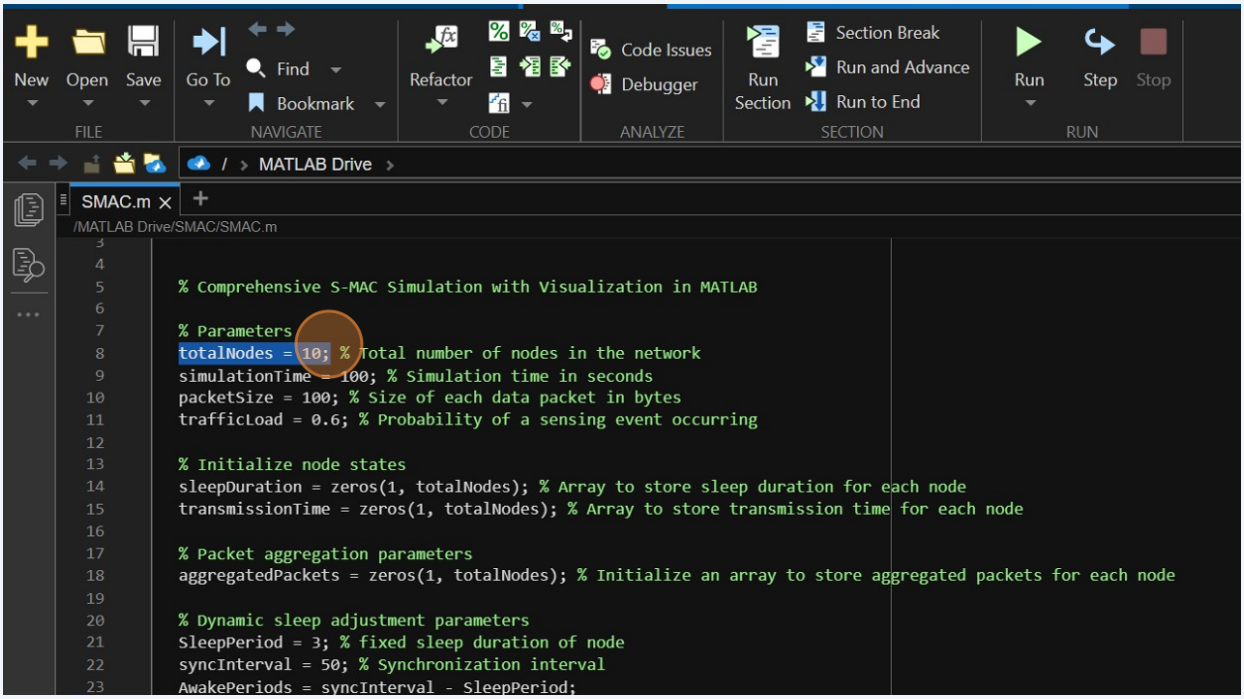
3 Now lets see the implemented code first , Starting with the initialization of Parameters



4 First parameter is the totalNodes that is selfexplanatory **totalNodes**:

• **Definition**: This parameter represents the total number of nodes within the wireless sensor network (WSN) being simulated.

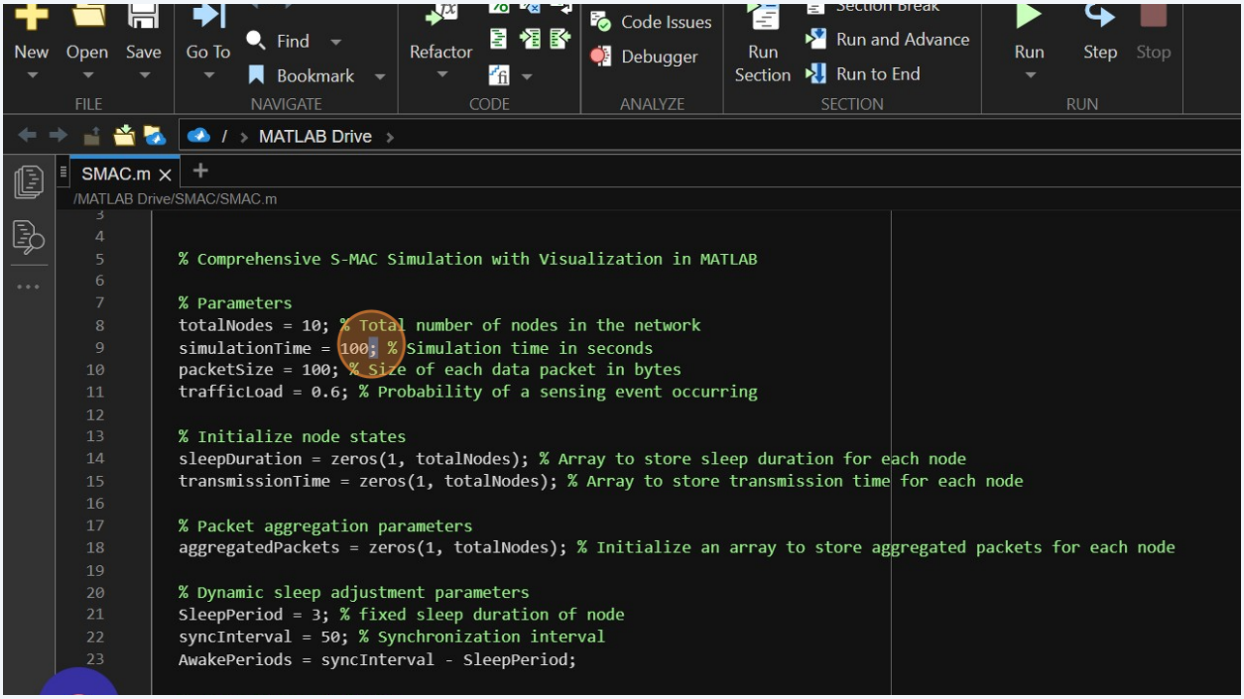
• **Usage**: It influences the size of arrays and loops in the simulation, as the script iterates over each node during the simulation time.



5 2nd is **simulationTime**:

• **Definition**: This parameter specifies the total duration of the simulation in seconds.

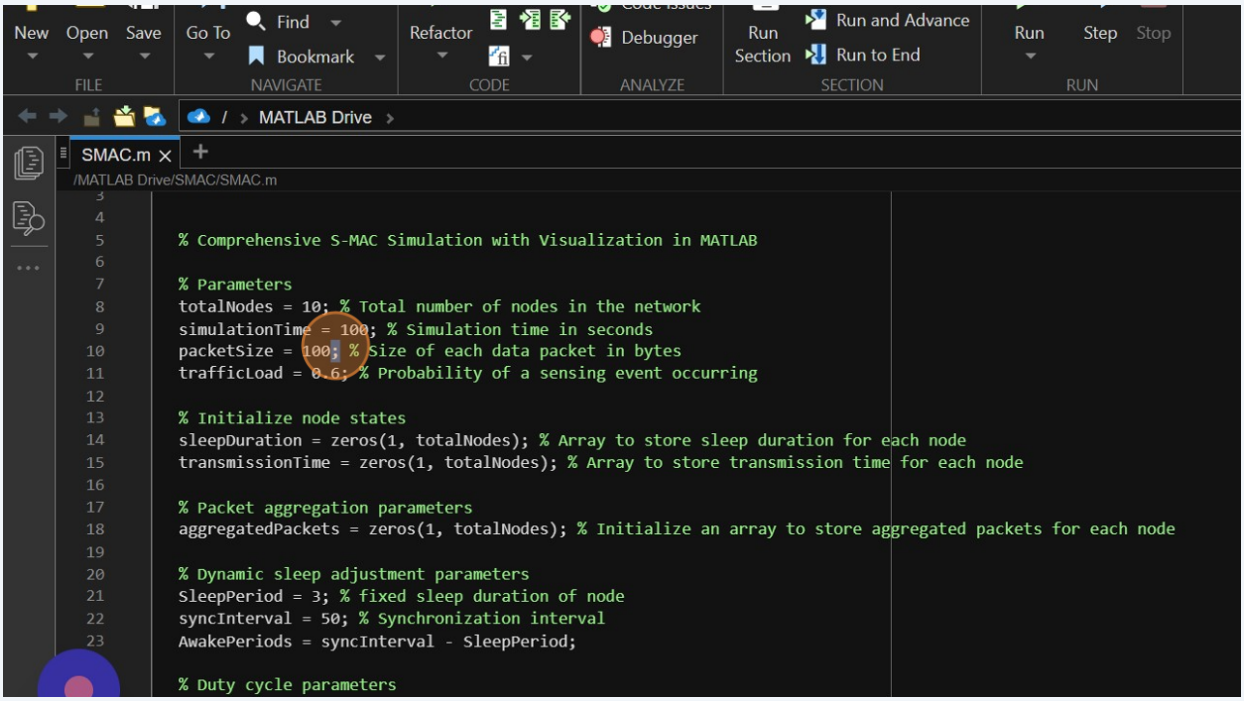
• **Usage**: It determines the overall time span for which the simulation will run, influencing the number of iterations in the simulation loop



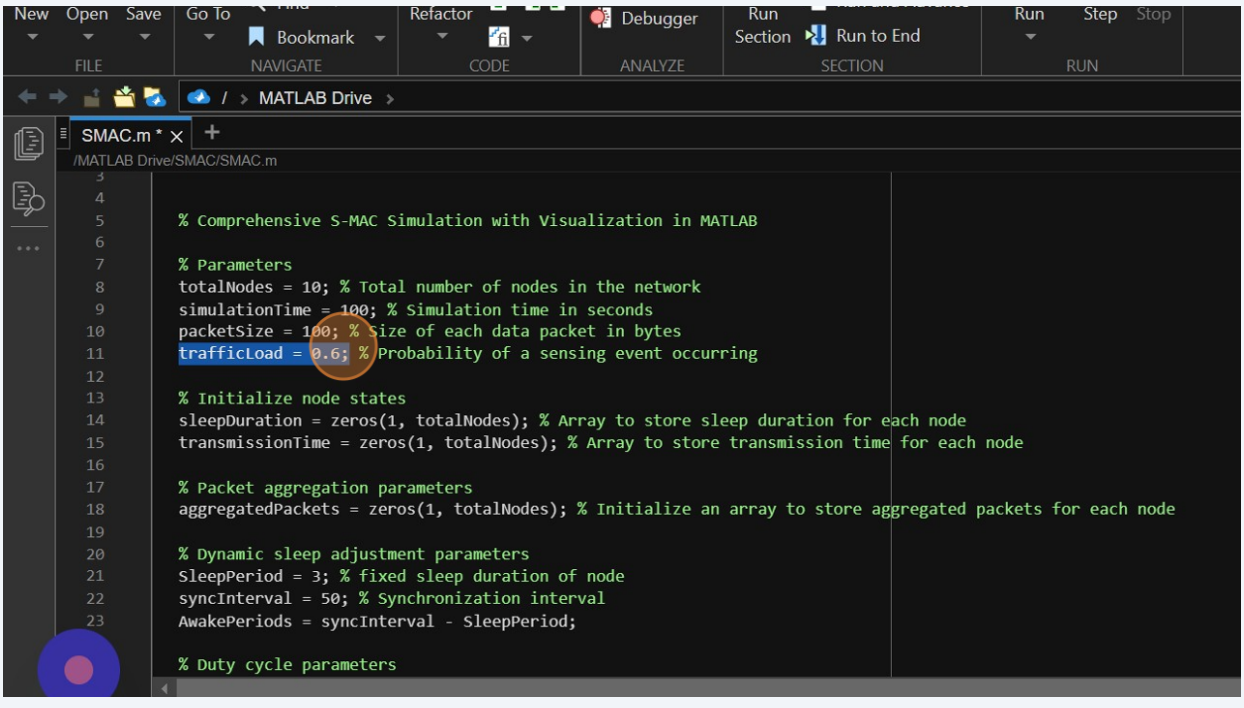
6 Then comes the third one **packetSize**:

• **Definition**: packetSize denotes the size of each data packet in bytes.

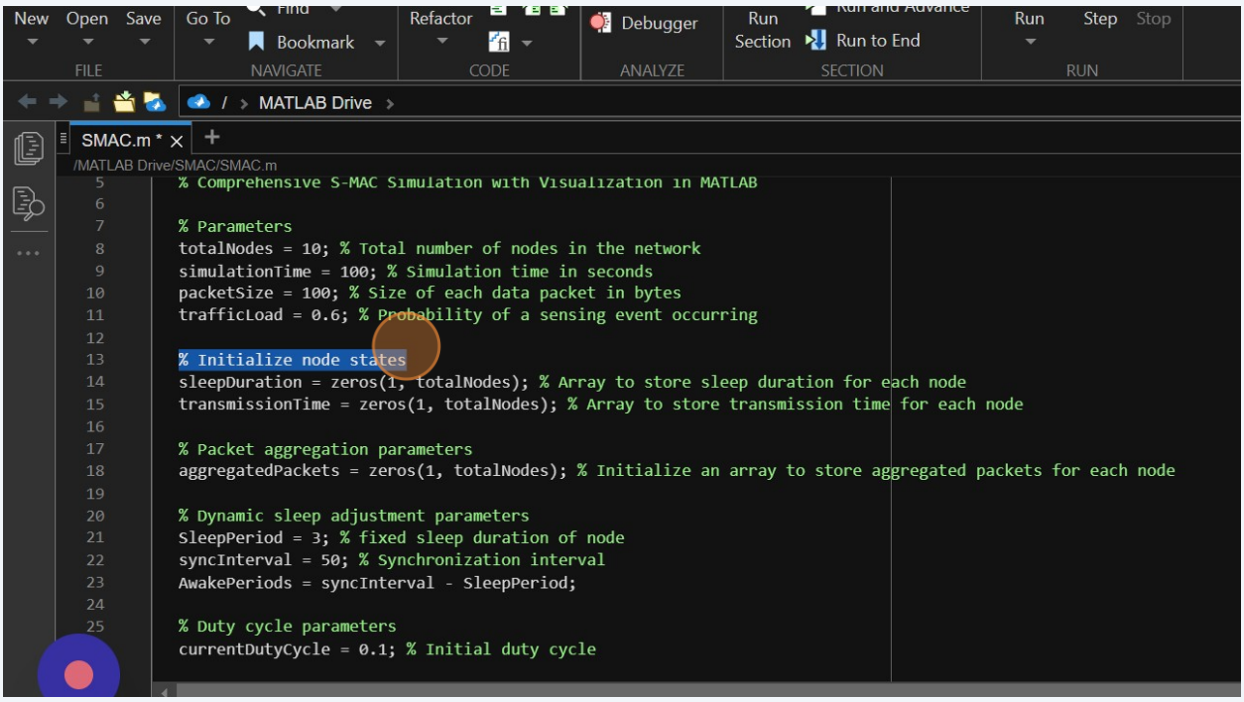
• **Usage**: It is crucial in modeling data transmission as it represents the amount of information exchanged between nodes during communication events



7 Click "trafficLoad = 0.6;"



8 Now lets have some arrays to store the states of nodes



9 The purpose is to initialize the variable **sleepDuration**.

Let me break it down:

• **zeros(1, totalNodes)**: This part creates a row vector of size totalNodes filled with zeros. The zeros function in MATLAB generates an array of zeros, and in this case, it creates a row vector with totalNodes elements.

• **sleepDuration** = ...: This part assigns the newly created row vector to the variable sleepDuration. So, sleepDuration is now a row vector containing totalNodes elements, all initialized to zero.

zeros(1, totalNodes): This part creates a row vector of size totalNodes filled with zeros. The zeros function in MATLAB generates an array of zeros, and in this case, it creates a row vector with totalNodes elements.

• **transmissionTime** = ...: This part assigns the newly created row vector to the variable transmissionTime. So, transmissionTime is now a row vector containing totalNodes elements, all initialized to zero.

This line of code sets up an array to keep track of the transmission time for each node in the network. Each element of the array corresponds to a specific node, and initially, all transmission times are set to zero. As the simulation progresses, this array will be updated to reflect the transmission time for each node based on the logic and conditions within the simulation loop

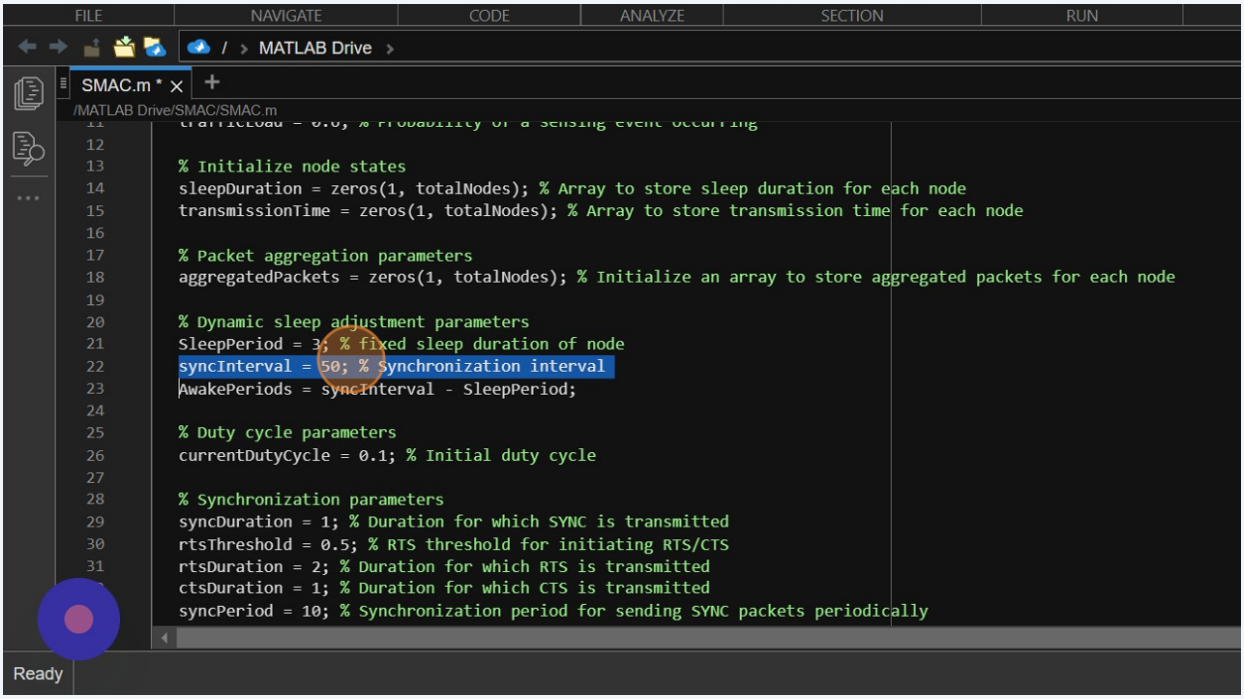
Then line **aggregatedPackets** = **zeros(1, totalNodes)**; initializes an array named aggregatedPackets with a length of totalNodes, where each element is initially set to zero. This array is intended to store the count of aggregated packets for each node in the network during the simulation.

Then some lines define **dynamic sleep adjustment** parameters for the S-MAC simulation.

Here's a breakdown:

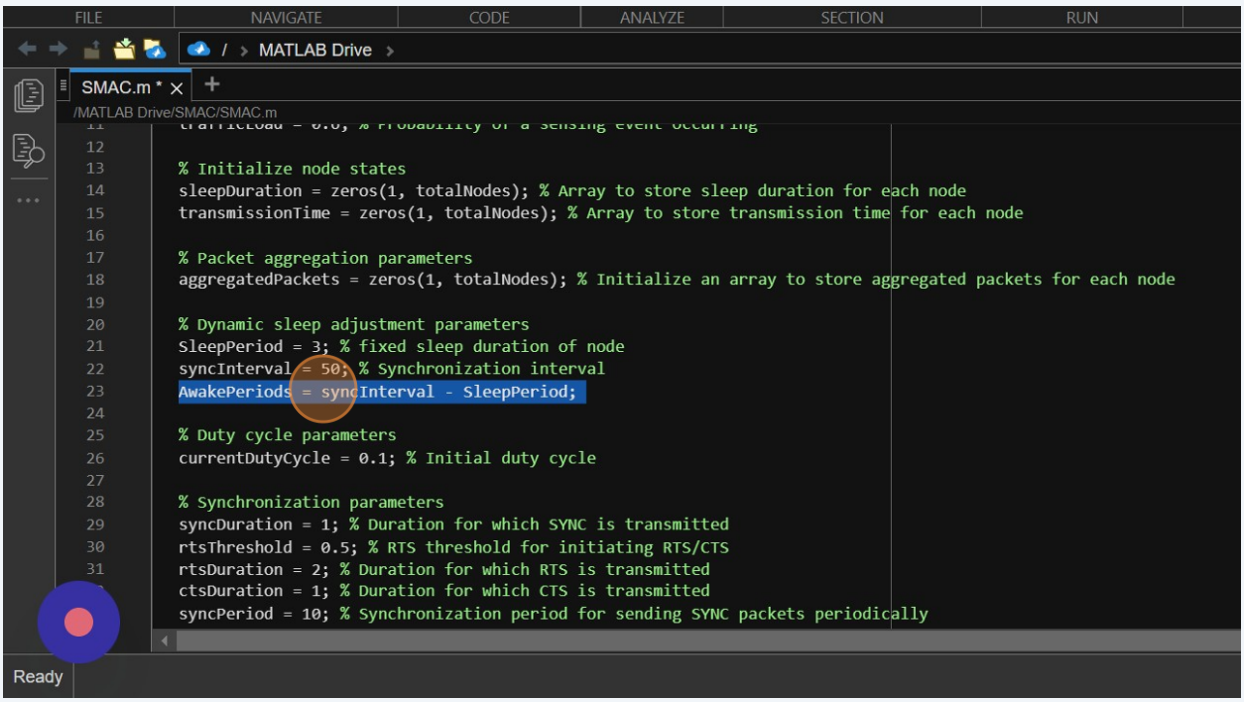
• **SleepPeriod** = 3;: This parameter represents the fixed sleep duration of a node. Each node will go to sleep for a duration of 3 simulation time units during its sleep period.

• **syncInterval** = 50;: This parameter sets the synchronization interval, indicating the total duration of the cycle (including both sleep and awake periods) for each node. In this case, it's set to 50 simulation time units.

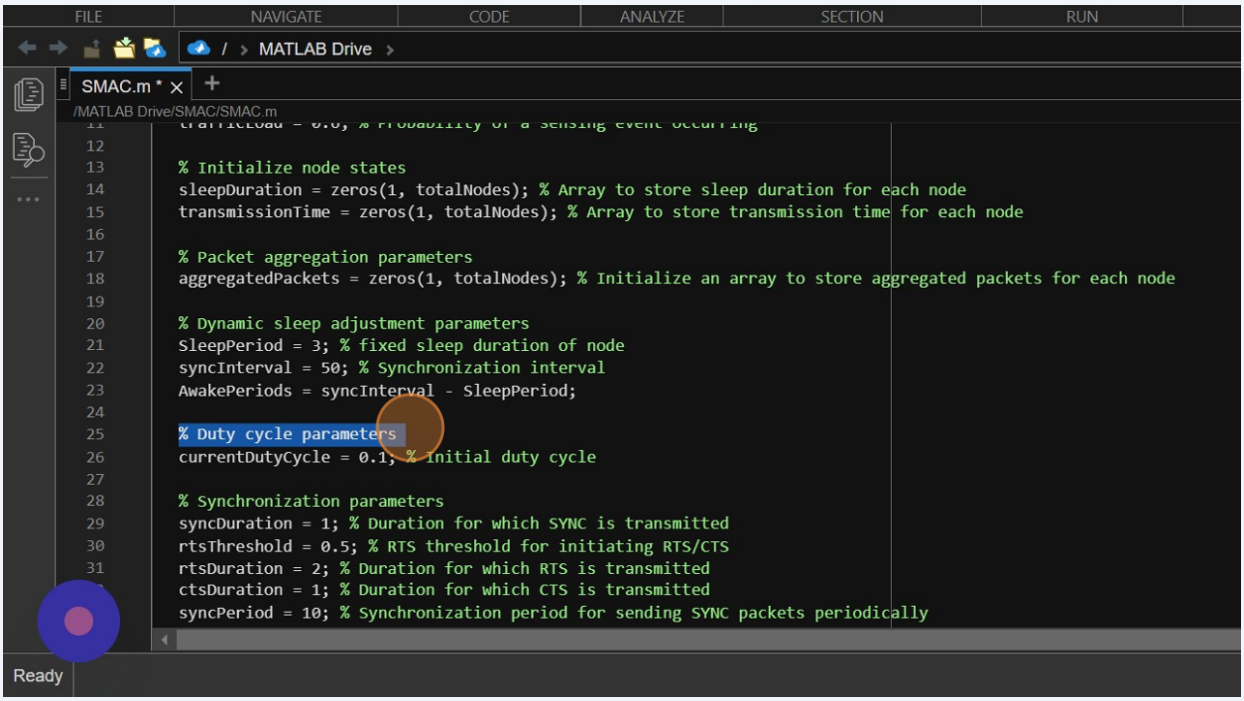


10

• **AwakePeriods** = syncInterval - SleepPeriod;: This line calculates the awake periods within the synchronization interval. It subtracts the sleep duration (SleepPeriod) from the total synchronization interval (syncInterval), resulting in the duration for which the node remains awake. The variable AwakePeriods will be used in the simulation loop to adjust the duty cycle.



11 currentDutyCycle: **This variable represents the current duty cycle of the network. It is initialized with a value of** 0.1**, indicating an initial duty cycle of 10%. This means that initially, each node in the network is expected to be active for 10% of the total synchronization interval.**



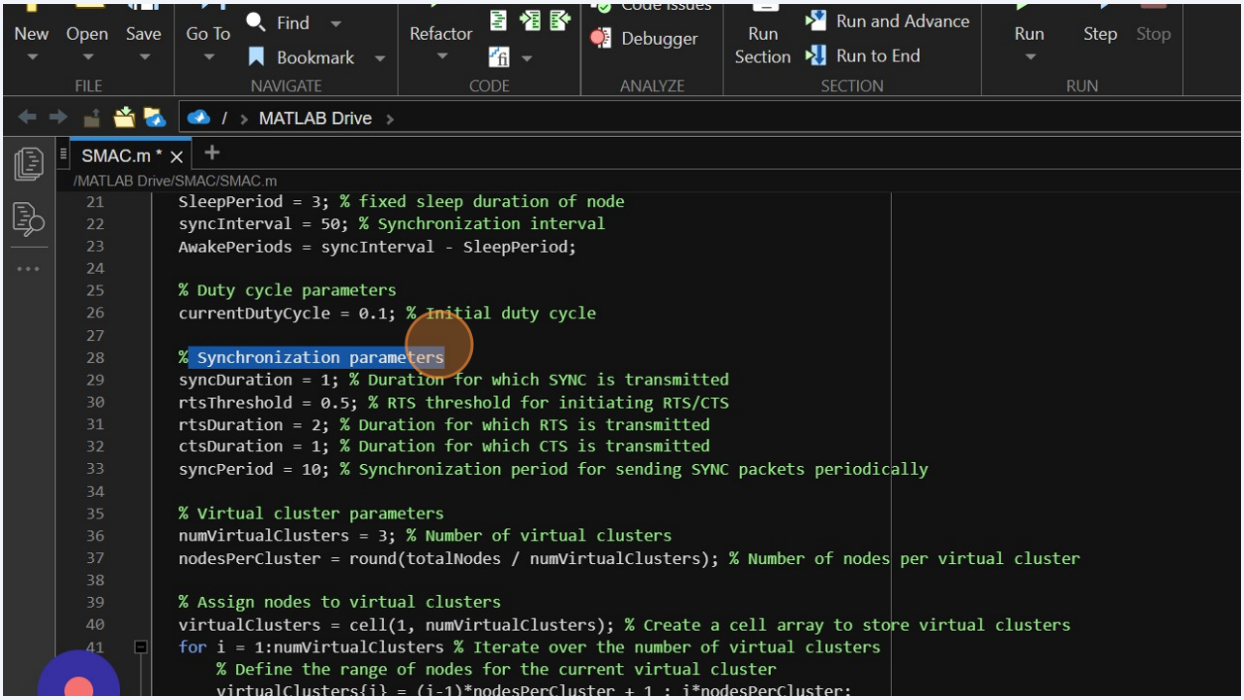
12 going to set some **synchronizationparameters** related to the synchronization process in the S-MAC simulation:

• **syncInterval** = 50;: This parameter sets the synchronization interval, representing the total duration of the cycle during which nodes synchronize and exchange control packets.

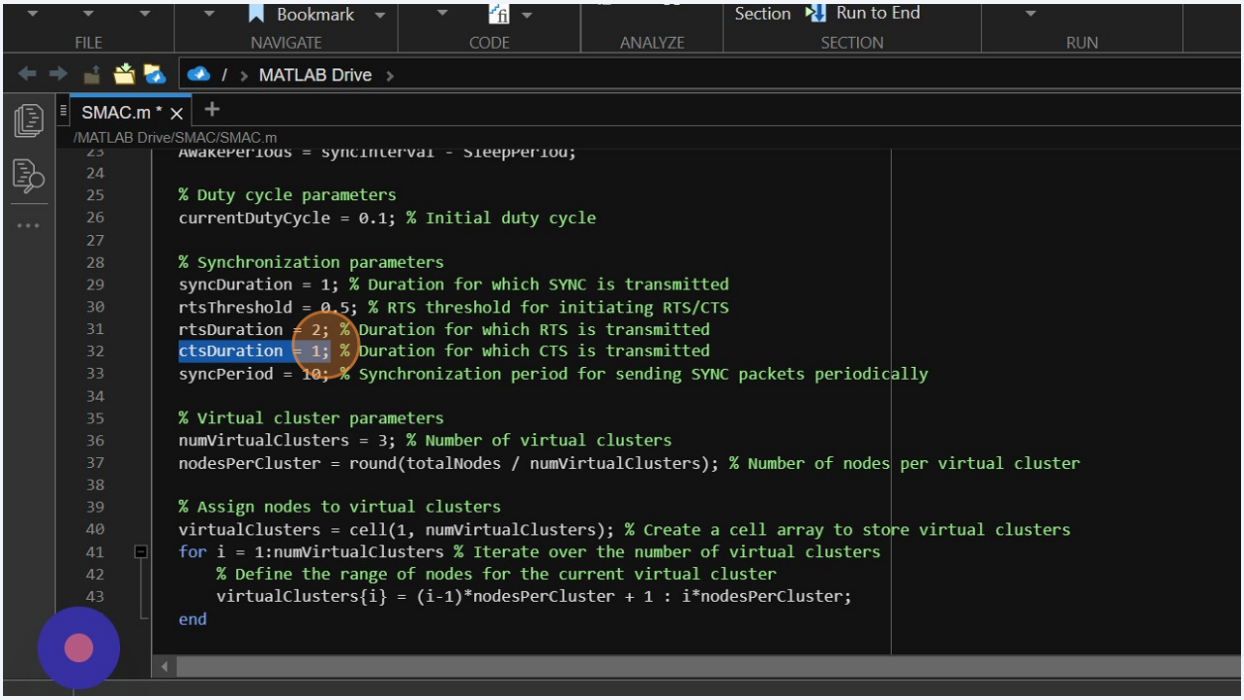
• **syncDuration** = 1;: It specifies the duration for which SYNC packets are transmitted during the synchronization phase.

• **rtsThreshold** = 0.5;: This parameter sets the RTS (Request to Send) threshold, representing the probability threshold for a neighboring node to initiate the RTS/CTS process. In the simulation, if a random number is less than this threshold, a neighbor initiates the process

• **rtsDuration** = 2;: It specifies the duration for which the RTS packet is transmitted during the RTS phase.

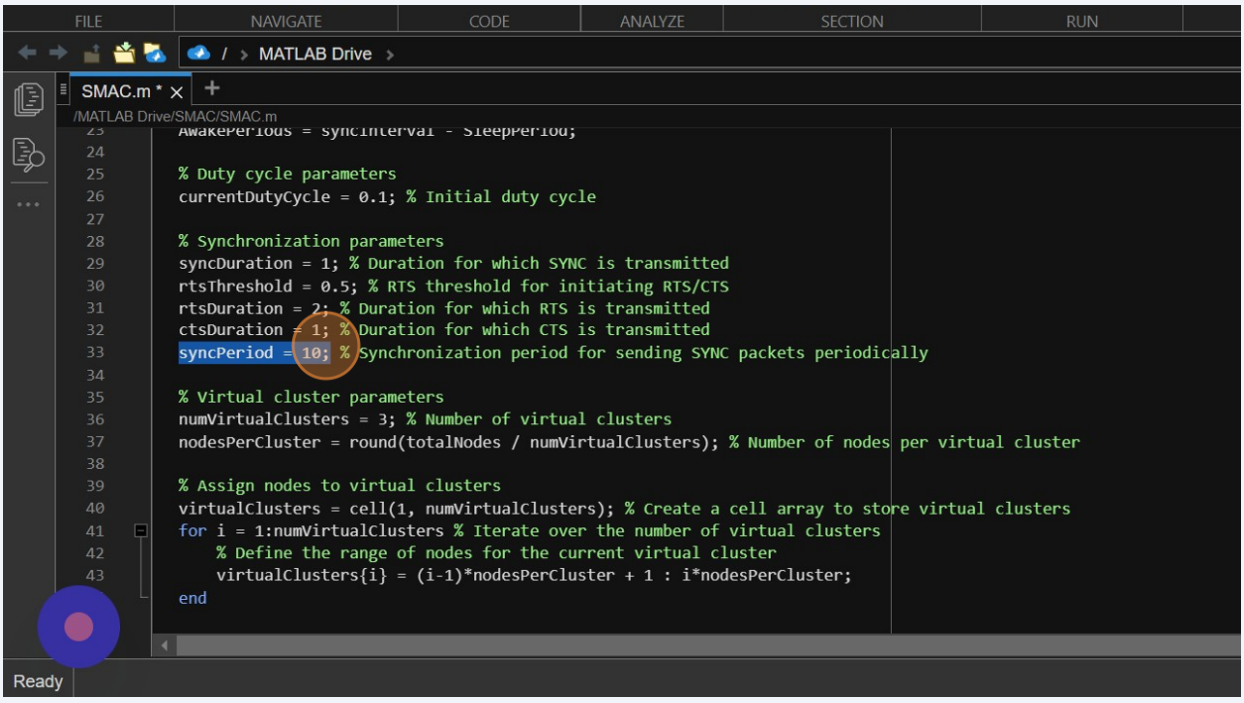


13 • **ctsDuration** = 1;: It specifies the duration for which the CTS (Clear to Send) packet is transmitted in response to an RTS packet.



14 • **syncPeriod** = 10;: This parameter sets the synchronization period, indicating how often SYNC packets are transmitted periodically during the simulation.

These parameters play a crucial role in governing the timing and duration of various synchronization-related activities, contributing to the overall coordination and communication efficiency in the S-MAC protocol simulation



15 Here's an explanation of the parameters involved:

• numVirtualClusters: This variable determines the total number of virtual clusters in the network. In the code, it's set to 3, indicating that the network will be divided into three virtual clusters.

• nodesPerCluster: This variable calculates the number of nodes that will be assigned to each virtual cluster. It is determined by dividing the total number of nodes (totalNodes) by the number of virtual clusters (numVirtualClusters). Since it's essential for this value to be an integer, round() function is used to round the result to the nearest whole number. Each virtual cluster will have approximately the same number of nodes.

These parameters are crucial for organizing the nodes within the network. By dividing the nodes into virtual clusters, it becomes easier to manage and control communication within the network.

Then

• virtualClusters: This variable is a cell array used to store the nodes assigned to each virtual cluster. It's initialized as an empty cell array with a length equal to the number of virtual clusters (numVirtualClusters).

• for i = 1:numVirtualClusters: This loop iterates over each virtual cluster index.

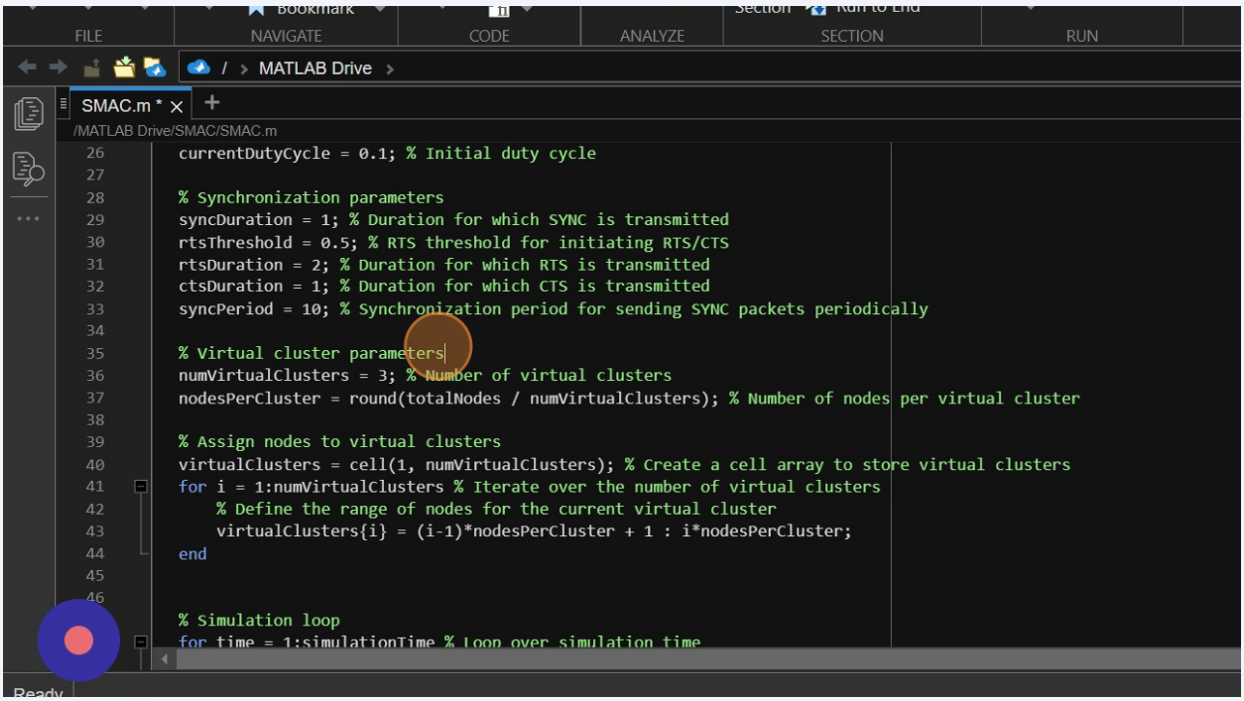
• virtualClusters{i} = (i-1)*nodesPerCluster + 1 : i*nodesPerCluster;: Within the loop, this line assigns nodes to the current virtual cluster. It uses array indexing to specify the range of nodes belonging to the current virtual cluster. The range is determined based on the number of nodes per cluster (nodesPerCluster) and the current virtual cluster index (i).

Here's how the assignment works:

• For the first virtual cluster (when i = 1), the range of nodes assigned to it starts from node 1 and ends at nodesPerCluster.

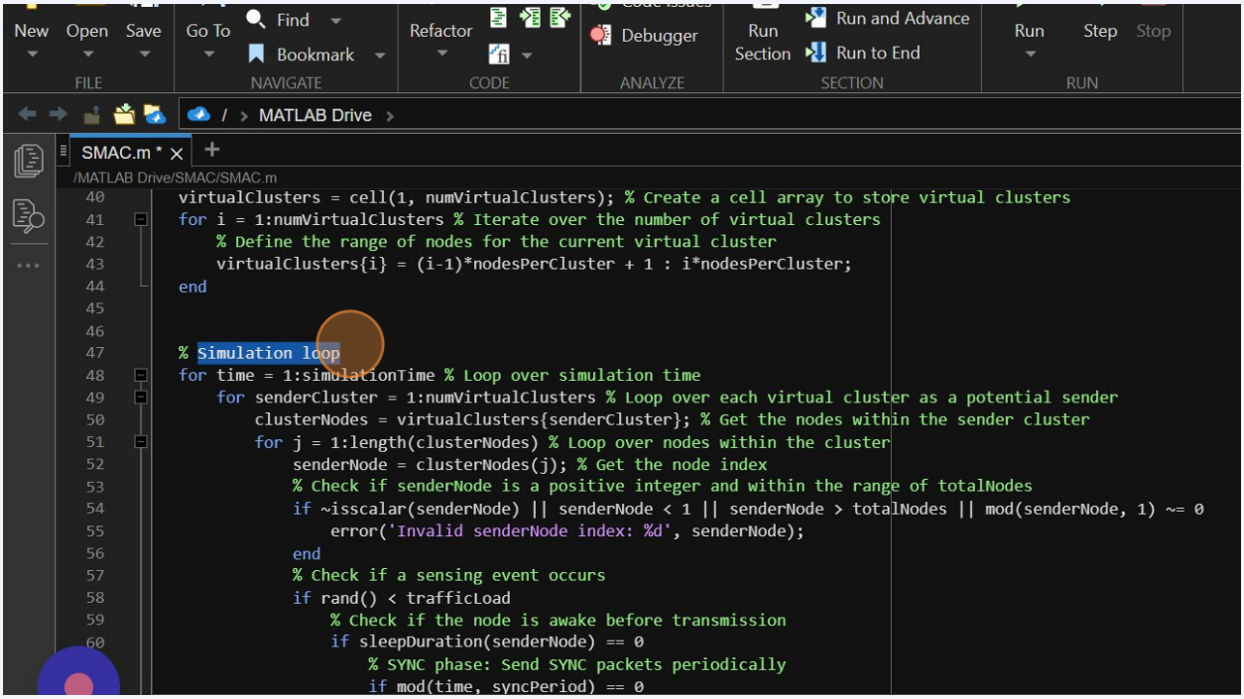
• For subsequent virtual clusters, the range is shifted by nodesPerCluster nodes.

After executing this code, the virtualClusters cell array will contain the ranges of nodes assigned to each virtual cluster.



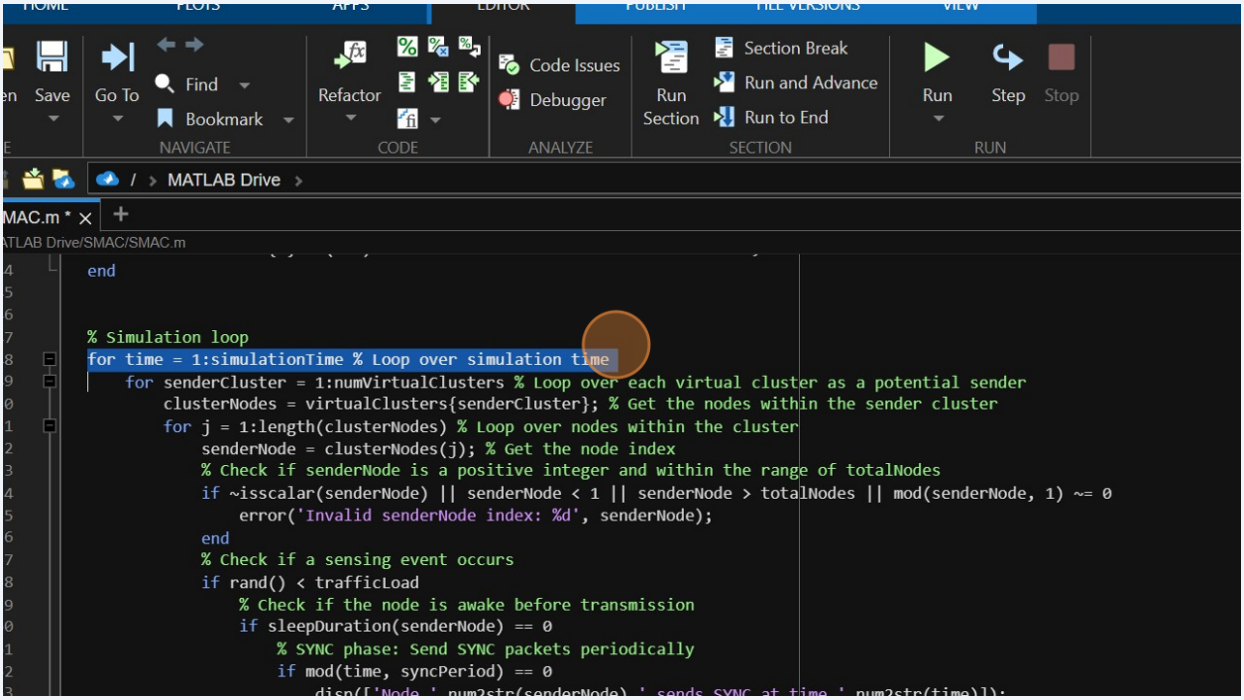
16 Now here comes the simulation Loop

Here's an overview of the key components within the loop:



17 **OuterLoop** (for time = 1:simulationTime):

• The outer loop iterates over the simulation time, progressing the simulation forward in discrete time steps.

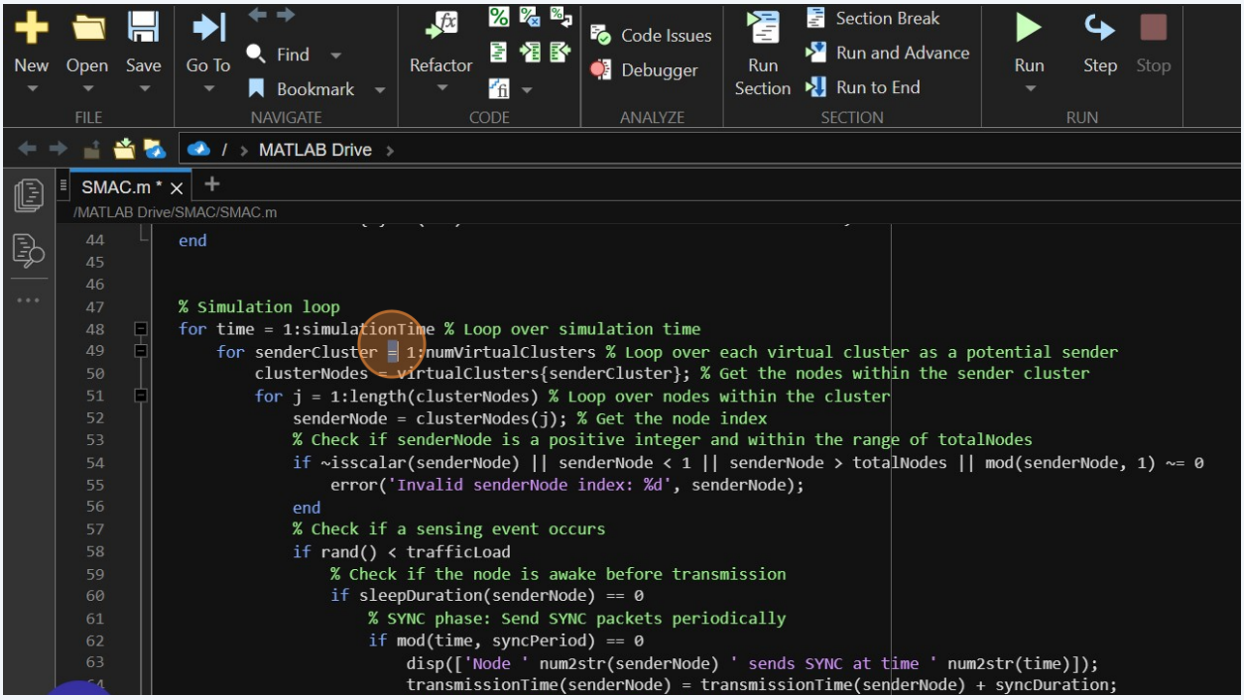


18 This section of the code iterates over each virtual cluster, treating each cluster as a potential sender of data. Let's break it down:

• for senderCluster = 1:numVirtualClusters: This loop iterates over each virtual cluster index, from 1 to the total number of virtual clusters (numVirtualClusters).

• clusterNodes = virtualClusters{senderCluster};: Within the loop, this line retrieves the nodes belonging to the current virtual cluster identified by the index senderCluster. It accesses the virtualClusters cell array using curly braces {}, where senderCluster serves as the index to access the specific virtual cluster.

After executing this code, clusterNodes will contain the indices of the nodes belonging to the current virtual cluster being considered as a potential sender.

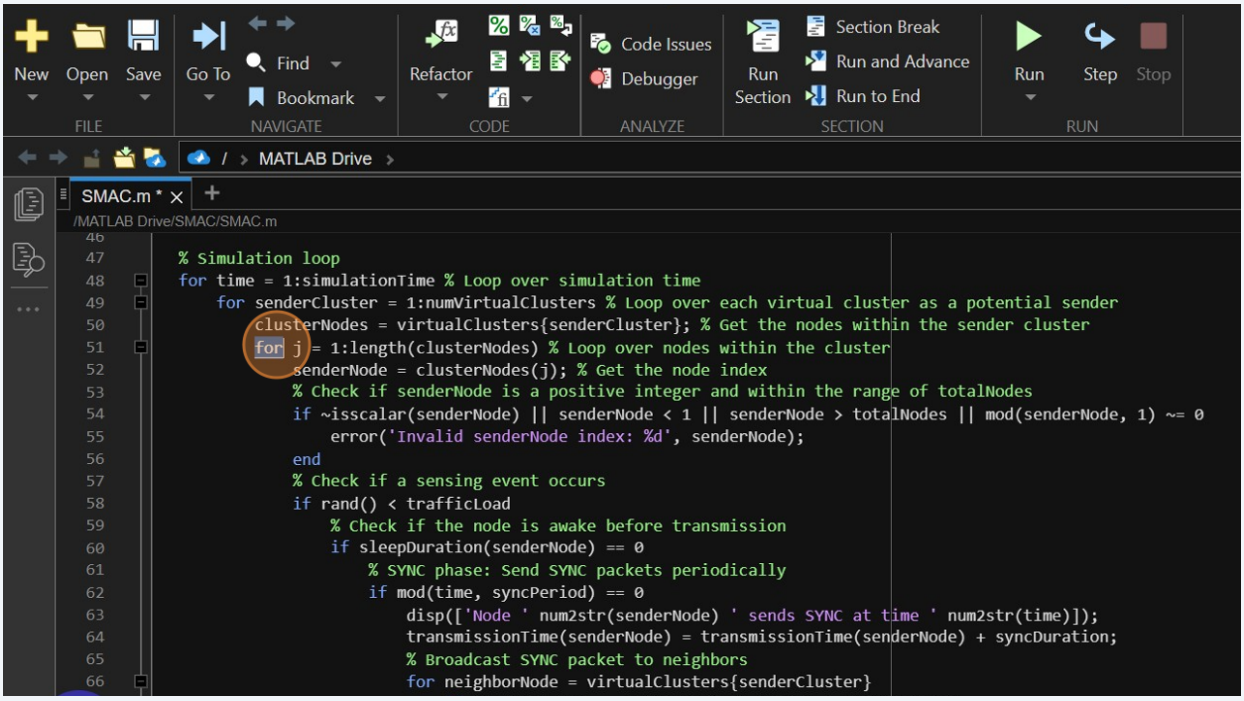


19 This section of the code iterates over each node within the current virtual cluster, allowing for individual node-level operations. Here's a breakdown:

• for j = 1:length(clusterNodes): This loop iterates over each element of the clusterNodes array, which contains the indices of the nodes within the current virtual cluster.

• senderNode = clusterNodes(j);: Within the loop, this line retrieves the index of the current node within the virtual cluster. The variable senderNode now holds the index of the node being considered for data transmission.

• The comment suggests that there should be a check to ensure that senderNode is a positive integer and within the range of totalNodes. However, the code snippet provided does not contain the actual implementation of this check.



20 This conditional statement checks whether the senderNode index meets certain criteria and raises an error if any of the conditions are not met. Let's break down each condition:

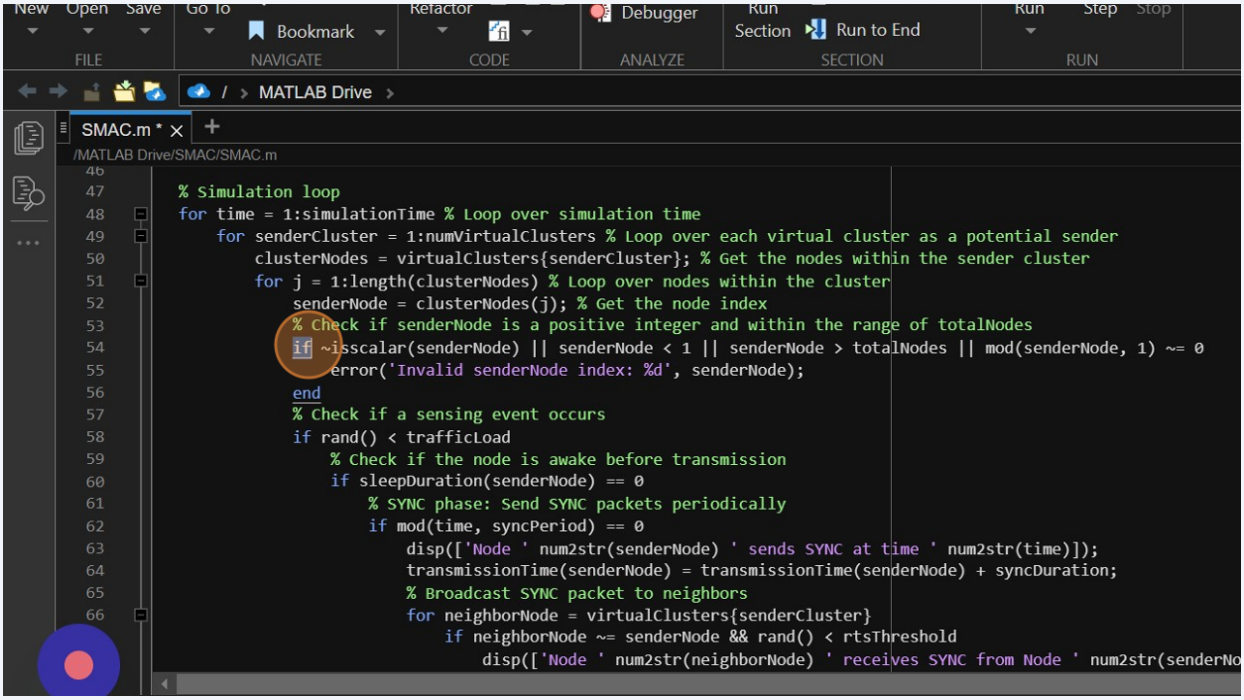
• ~isscalar(senderNode): This condition checks if senderNode is not a scalar value. In other words, it checks if senderNode is not a single integer value. If senderNode is not a scalar, it indicates that it might be an array or a non-integer value, which would be invalid for indexing nodes.

• senderNode < 1: This condition checks if senderNode is less than 1. If senderNode is less than 1, it implies that the index is out of range since node indices typically start from 1 in programming contexts.

• senderNode > totalNodes: This condition checks if senderNode is greater than the total number of nodes (totalNodes). If senderNode is greater than the total number of nodes, it indicates that the index is out of range since it exceeds the maximum valid node index.

• mod(senderNode, 1) ~= 0: This condition checks if senderNode is not an integer value. The mod(senderNode, 1) operation calculates the remainder when dividing senderNode by 1. If the remainder is not 0, it means that senderNode is not an integer.

If any of these conditions evaluate to true, the code raises an error using the error function, indicating that the senderNode index is invalid. The error message includes the value of the invalid index for debugging purposes. This helps ensure the integrity of the simulation by catching and addressing any issues related to invalid node indices.



21 This conditional statement checks whether a randomly generated number falls below a threshold defined by trafficLoad. Let's break it down:

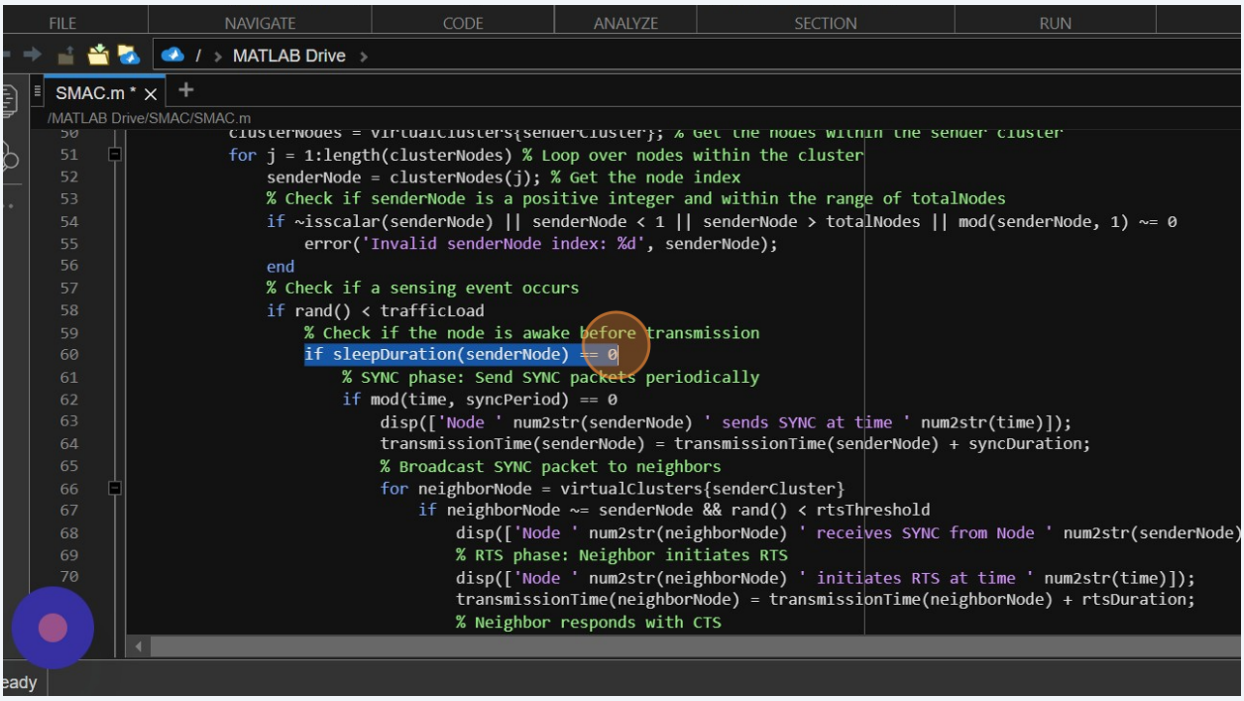
• rand(): This function generates a random number between 0 and 1. Since no arguments are passed to rand(), it generates a uniformly distributed random number in the interval [0, 1).

• trafficLoad: This variable represents the probability of a sensing event occurring. It is typically a value between 0 and 1, where 0 indicates no sensing events and 1 indicates sensing events always occur.

Therefore, rand() < trafficLoad evaluates to true with a probability equal to trafficLoad. In other words, it simulates a Bernoulli trial with success probability trafficLoad. If the randomly generated number is less than trafficLoad, the condition evaluates to true, indicating that a sensing event occurs.

• sleepDuration(senderNode): This retrieves the sleep duration of the node specified by the senderNode index. The sleep duration represents the amount of time the node has been in a sleep state.

• == 0: This checks if the retrieved sleep duration is equal to zero.

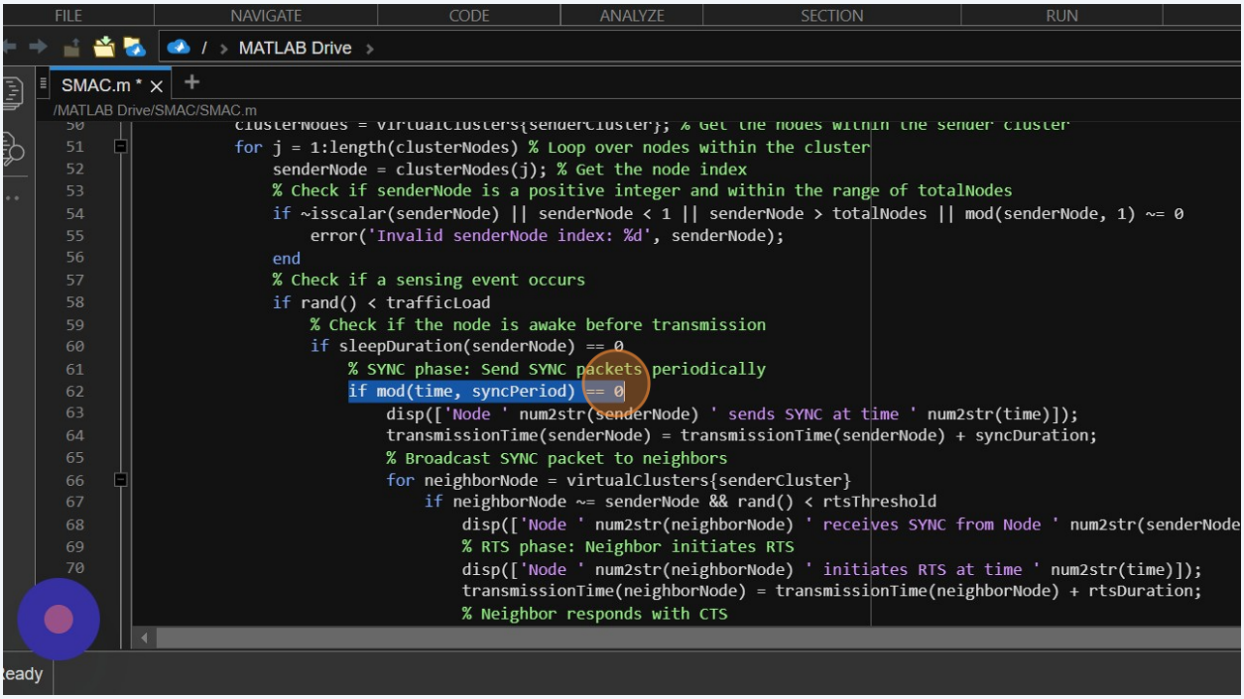


22 This conditional statement checks if the current simulation time (time) is a multiple of the synchronization period (syncPeriod). Let's break it down:

• mod(time, syncPeriod): This calculates the remainder when time is divided by syncPeriod. If the remainder is zero, it means that time is evenly divisible by syncPeriod, indicating that time is a multiple of syncPeriod.

• == 0: This checks if the remainder is zero, indicating that the current time is indeed a multiple of syncPeriod.

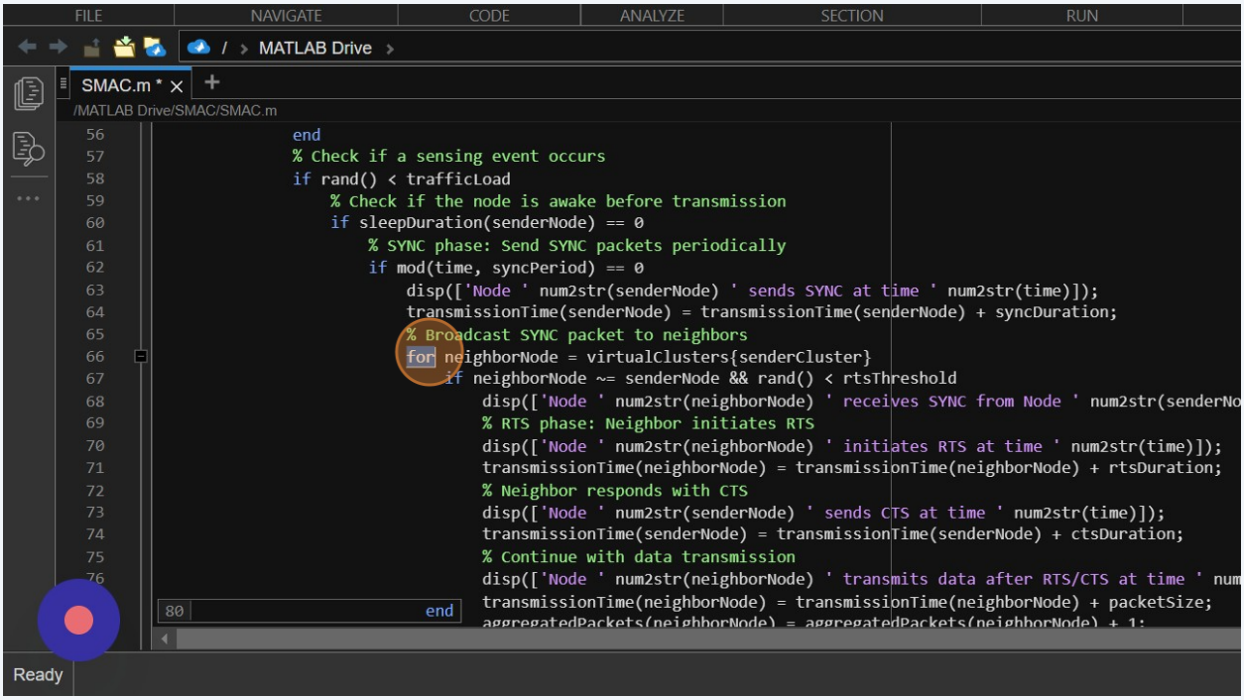
If the condition mod(time, syncPeriod) == 0 evaluates to true, it means that the current simulation time is at a synchronization point, where nodes are scheduled to perform synchronization tasks such as sending SYNC packets



23 This code block represents the process where a node receives a SYNC packet from a neighboring node, initiates a Request to Send (RTS), receives a Clear to Send (CTS), and then continues with data transmission. Let's break it down:

• for neighborNode = virtualClusters{senderCluster}: This loop iterates over each node in the virtual cluster (senderCluster) to which the current node belongs. These nodes are considered as potential neighbors.

•



24

• if neighborNode ~= senderNode && rand() < rtsThreshold: This condition checks if the neighborNode is not the same as the senderNode (to avoid initiating communication with itself) and if a randomly generated number is less than the RTS threshold (rtsThreshold). This randomization introduces a stochastic element into the simulation, simulating the probability of initiating RTS based on the given threshold.

• If the condition is met, it means that the neighbor node is selected for communication and the RTS process begins:

• disp(['Node ' num2str(neighborNode) ' receives SYNC from Node ' num2str(senderNode)]);: This line displays a message indicating that the neighbor node has received a SYNC packet from the sender node.

• disp(['Node ' num2str(neighborNode) ' initiates RTS at time ' num2str(time)]);: This line displays a message indicating that the neighbor node initiates the RTS phase at the current simulation time.

• transmissionTime(neighborNode) = transmissionTime(neighborNode) + rtsDuration;: This increments the transmission time of the neighbor node by the duration of the RTS packet (rtsDuration).

• disp(['Node ' num2str(senderNode) ' sends CTS at time ' num2str(time)]);: This line displays a message indicating that the sender node sends a Clear to Send (CTS) packet in response to the RTS from the neighbor node.

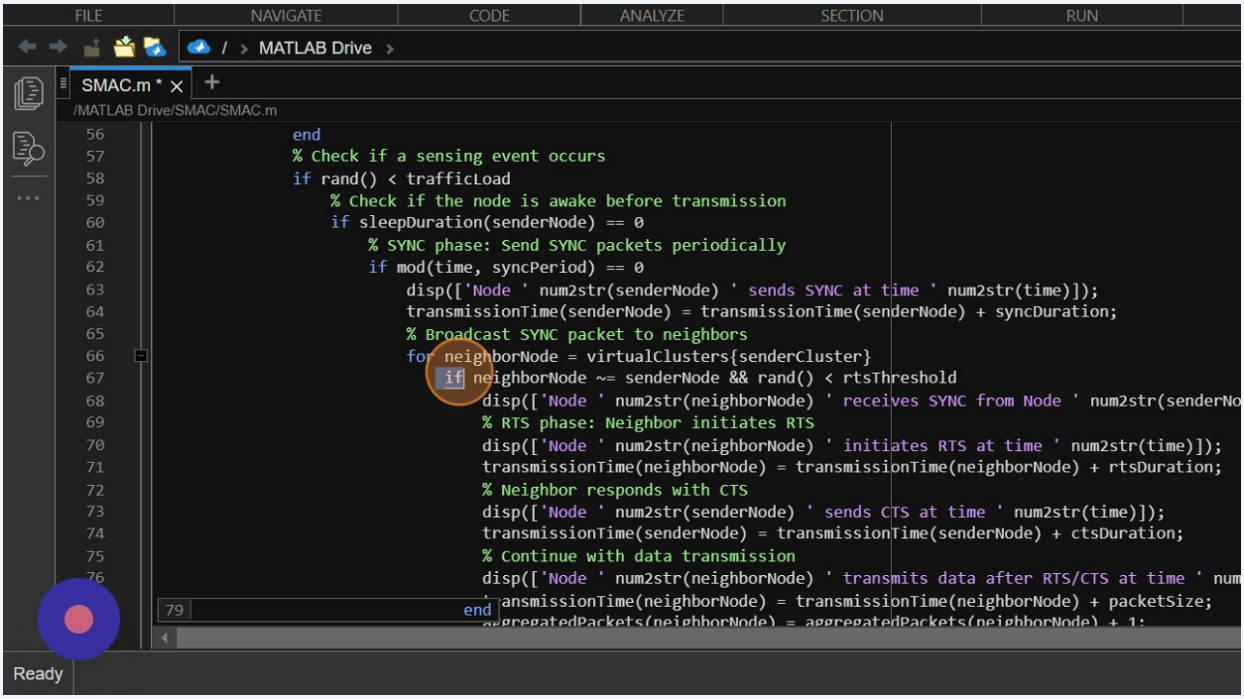
• transmissionTime(senderNode) = transmissionTime(senderNode) + ctsDuration;: This increments the transmission time of the sender node by the duration of the CTS packet (ctsDuration).

• disp(['Node ' num2str(neighborNode) ' transmits data after RTS/CTS at time ' num2str(time)]);: This line displays a message indicating that the neighbor node continues with data transmission after receiving the CTS from the sender node.

• transmissionTime(neighborNode) = transmissionTime(neighborNode) + packetSize;: This increments the transmission time of the neighbor node by the size of the data packet (packetSize).

• aggregatedPackets(neighborNode) = aggregatedPackets(neighborNode) + 1;: This increments the count of aggregated packets for the neighbor node.

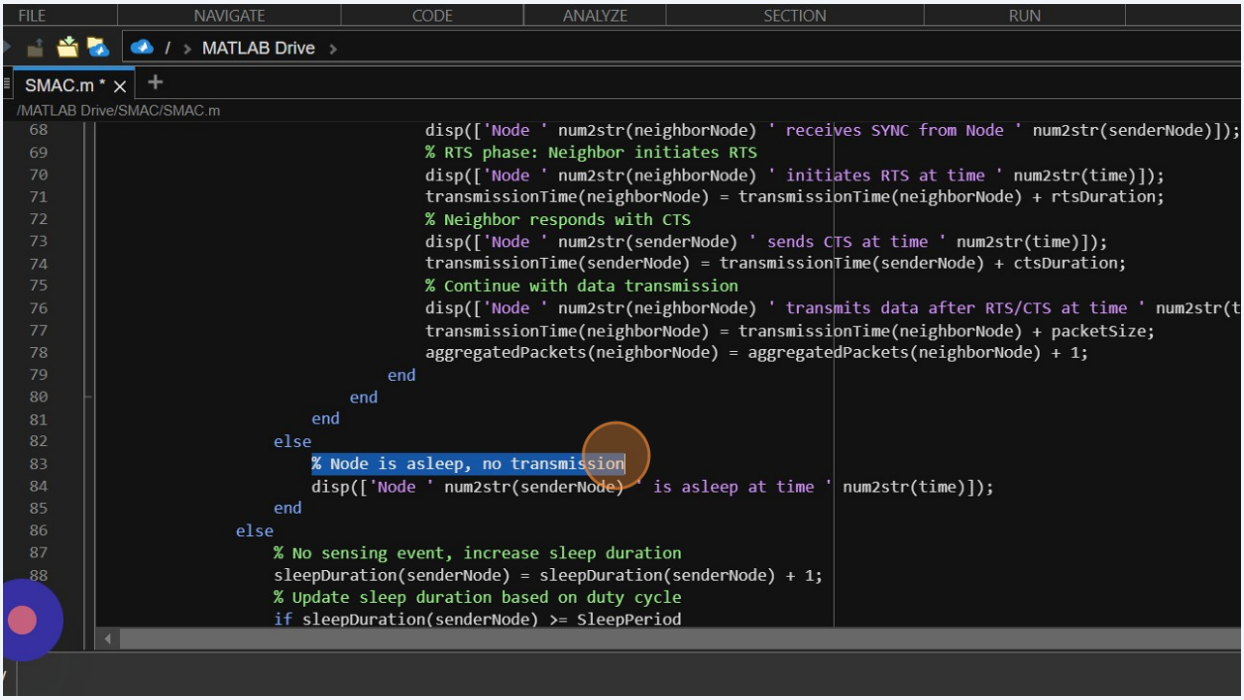
Overall, this block simulates the process of initiating and completing a data transmission session between neighboring nodes within the network



25 This part of the code handles the scenario where a node is currently in a sleep state and therefore cannot transmit data. Let's break it down:

• disp(['Node ' num2str(senderNode) ' is asleep at time ' num2str(time)]);: This line displays a message indicating that the node represented by senderNode is currently in a sleep state at the current simulation time time.

The purpose of this message is to provide insight into the simulation process, informing the user or developer that the node is currently unable to transmit data due to being in a sleep state.



26 This part of the code handles the situation where no sensing event occurs, leading to an increase in the sleep duration of the node. Let's break it down:

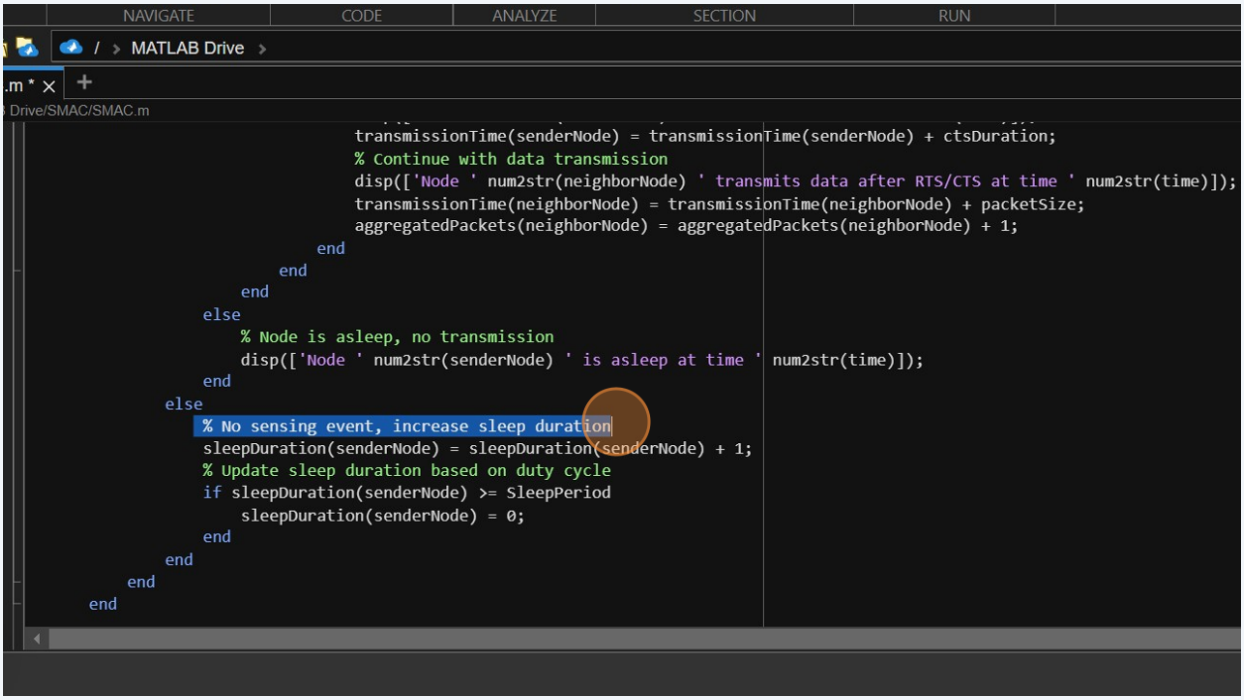
• sleepDuration(senderNode) = sleepDuration(senderNode) + 1;: This line increments the sleep duration of the node represented by senderNode by 1. It indicates that the node has spent an additional unit of time in the sleep state.

• if sleepDuration(senderNode) >= SleepPeriod: This conditional statement checks if the sleep duration of the node has reached or exceeded the predefined sleep period (SleepPeriod).

• If the condition is met (sleepDuration(senderNode) >= SleepPeriod), it means that the node has been in a sleep state for a duration equal to or longer than the specified sleep period. In this case:

• sleepDuration(senderNode) = 0;: This line resets the sleep duration of the node to zero, effectively indicating that the node has completed its sleep cycle and is ready to transition to an active state for the next cycle.

This mechanism simulates the behavior of nodes in a wireless sensor network where nodes alternate between sleep and active states according to a predefined duty cycle. By incrementing the sleep duration and resetting it after reaching the sleep period, the simulation captures the periodic sleep-wake cycle of nodes, which is essential for energy management and network performance optimization



27

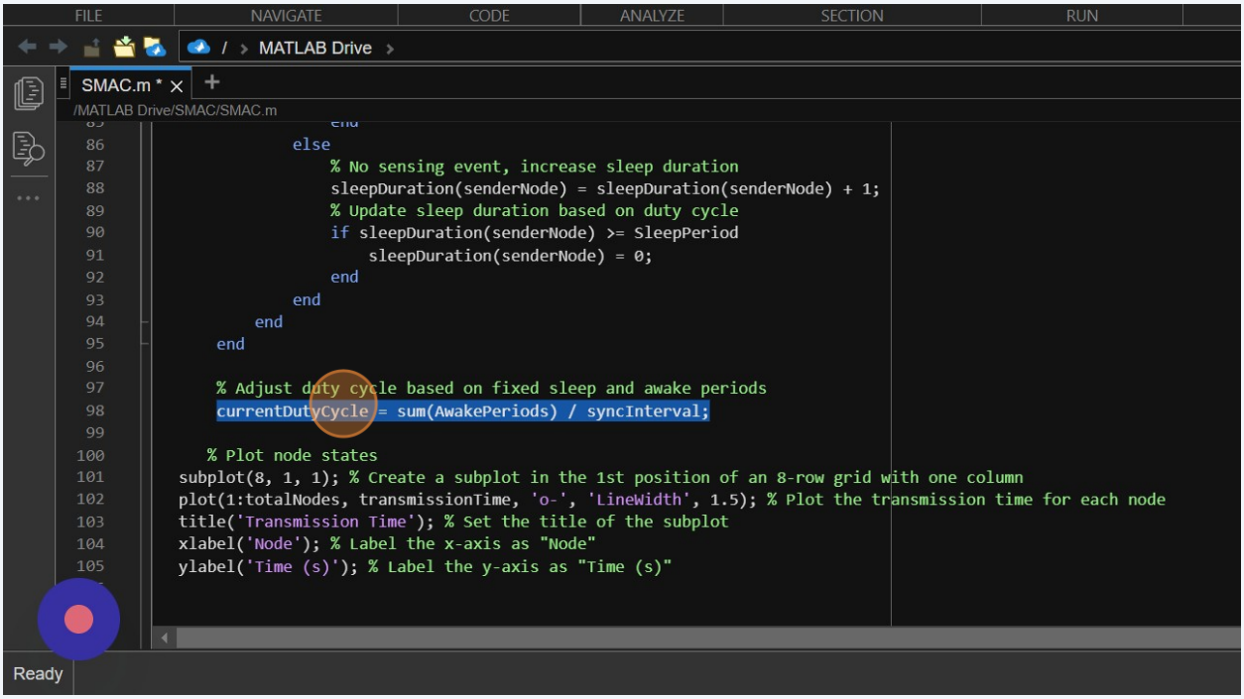
This section of the code calculates the current duty cycle based on fixed sleep and awake periods. Let's break it down:

• sum(AwakePeriods): This calculates the total duration of awake periods across all nodes in the network. The AwakePeriods variable likely represents an array or vector containing the duration of awake periods for each node.

• / syncInterval: This divides the total duration of awake periods by the synchronization interval (syncInterval). The synchronization interval typically represents the duration over which the duty cycle is evaluated or synchronized.

• currentDutyCycle = ...: This assigns the result of the division to the variable currentDutyCycle, representing the current duty cycle of the network.

In wireless sensor networks, the duty cycle is often adjusted dynamically based on the sleep and awake periods of nodes to balance energy consumption with communication requirements.



28 This section of the code is responsible for plotting the transmission time of each node in the network. Here's a breakdown:

• subplot(8, 1, 1): This function call creates a subplot grid with 8 rows and 1 column, and it selects the first subplot for the current plot.

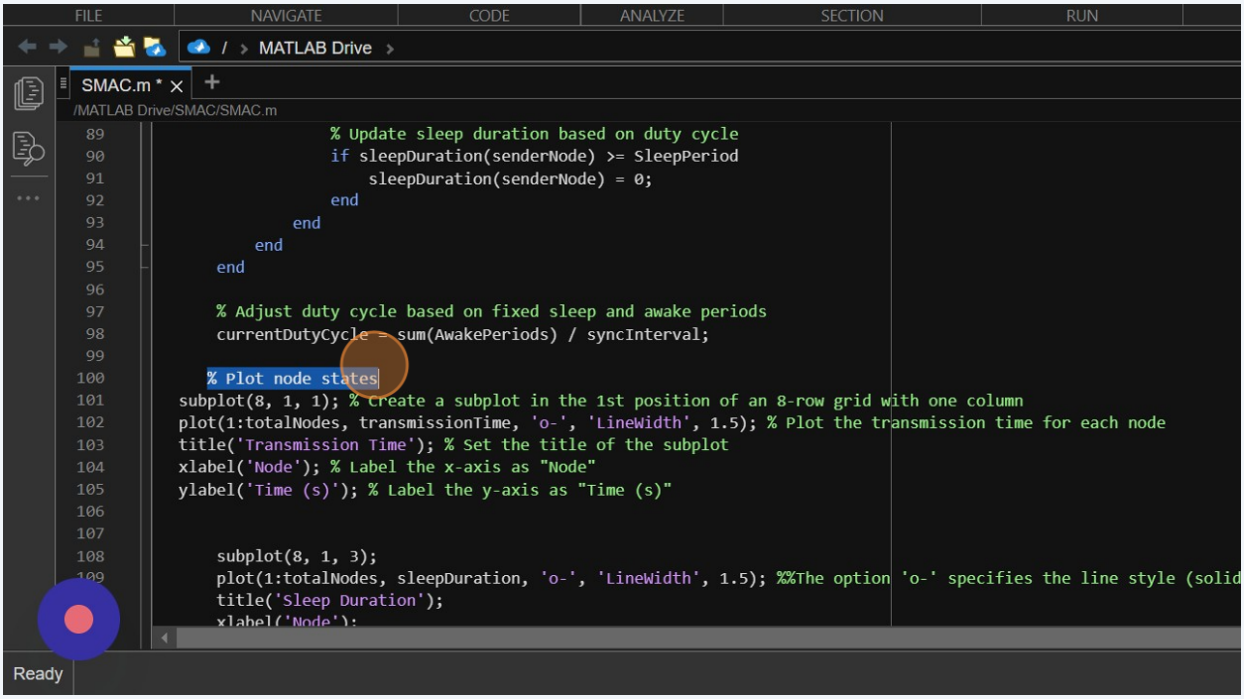
• plot(1:totalNodes, transmissionTime, 'o-', 'LineWidth', 1.5): This line plots the transmission time for each node. The x-axis represents the nodes in the network (from 1 to the total number of nodes), and the y-axis represents the transmission time. The data is plotted as a line with markers ('o') and a solid line connecting the markers ('-'). The line width is set to 1.5 for better visibility.

• title('Transmission Time'): This sets the title of the subplot to "Transmission Time".

• xlabel('Node'): This labels the x-axis as "Node", indicating the node indices.

• ylabel('Time (s)'): This labels the y-axis as "Time (s)", indicating the transmission time in seconds.

Overall, this code segment generates a plot showing the transmission time of each node in the network. This visualization can help analyze the distribution of transmission times across nodes and identify any patterns or outliers in the data



29 This section of the code is responsible for plotting the sleep duration of each node in the network. Let's break it down:

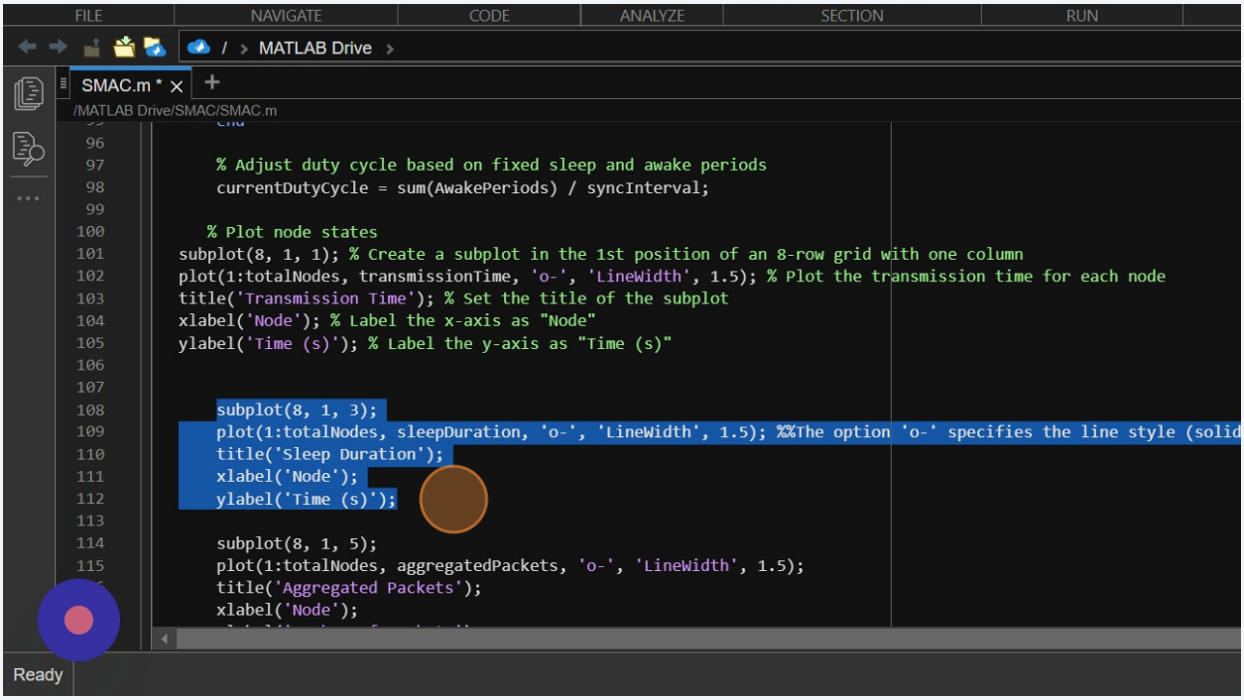
• subplot(8, 1, 3): This function call creates a subplot grid with 8 rows and 1 column, and it selects the third subplot for the current plot.

• plot(1:totalNodes, sleepDuration, 'o-', 'LineWidth', 1.5): This line plots the sleep duration for each node. The x-axis represents the nodes in the network (from 1 to the total number of nodes), and the y-axis represents the sleep duration. The data is plotted as a line with markers ('o') and a solid line connecting the markers ('-'). The line width is set to 1.5 for better visibility.

• title('Sleep Duration'): This sets the title of the subplot to "Sleep Duration".

• xlabel('Node'): This labels the x-axis as "Node", indicating the node indices.

• ylabel('Time (s)'): This labels the y-axis as "Time (s)", indicating the sleep duration in seconds.



30 This part of the code creates a subplot to plot the number of aggregated packets for each node in the network. Let's break it down:

• subplot(8, 1, 5): This function creates a subplot grid with 8 rows and 1 column, and it selects the fifth subplot for the current plot.

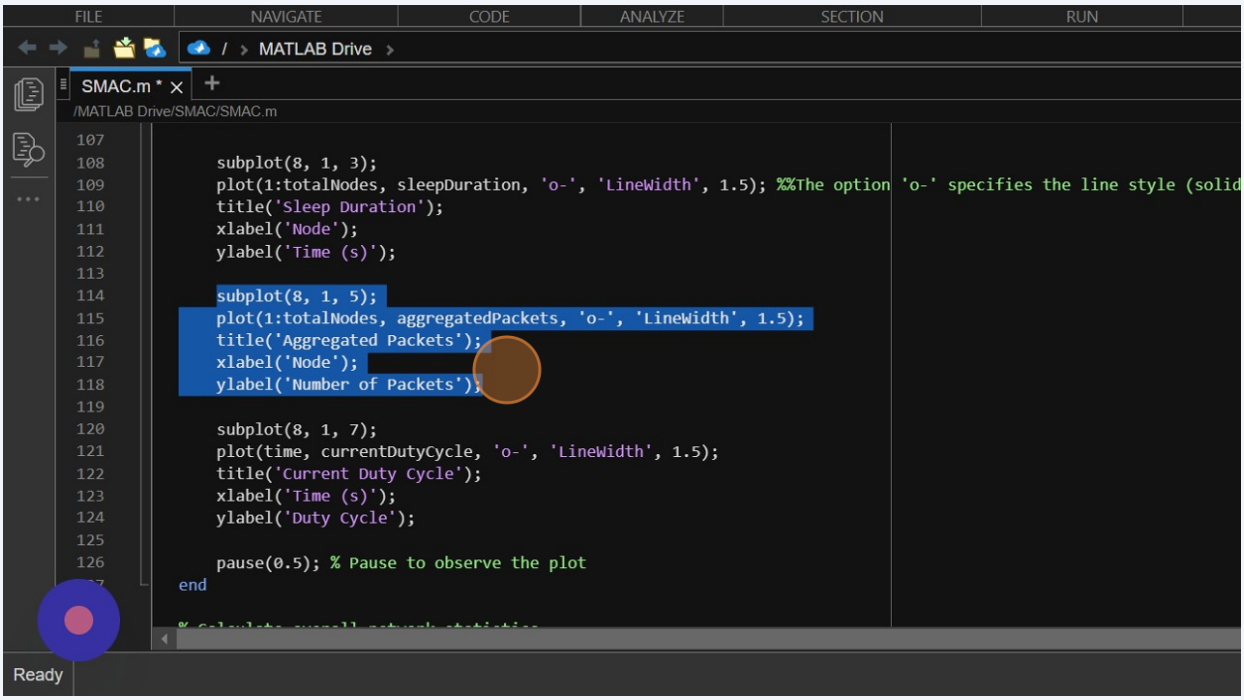
• plot(1:totalNodes, aggregatedPackets, 'o-', 'LineWidth', 1.5): This line plots the number of aggregated packets for each node. The x-axis represents the nodes in the network (from 1 to the total number of nodes), and the y-axis represents the number of aggregated packets. The data is plotted as a line with markers ('o') and a solid line connecting the markers ('-'). The line width is set to 1.5 for better visibility.

• title('Aggregated Packets'): This sets the title of the subplot to "Aggregated Packets".

• xlabel('Node'): This labels the x-axis as "Node", indicating the node indices.

• ylabel('Number of Packets'): This labels the y-axis as "Number of Packets", indicating the count of aggregated packets for each node.

Overall, this code segment generates a plot showing the number of aggregated packets for each node in the network



31 This part of the code creates a subplot to plot the current duty cycle of the network over time. Here's the breakdown:

• subplot(8, 1, 7): This function creates a subplot grid with 8 rows and 1 column, and it selects the seventh subplot for the current plot.

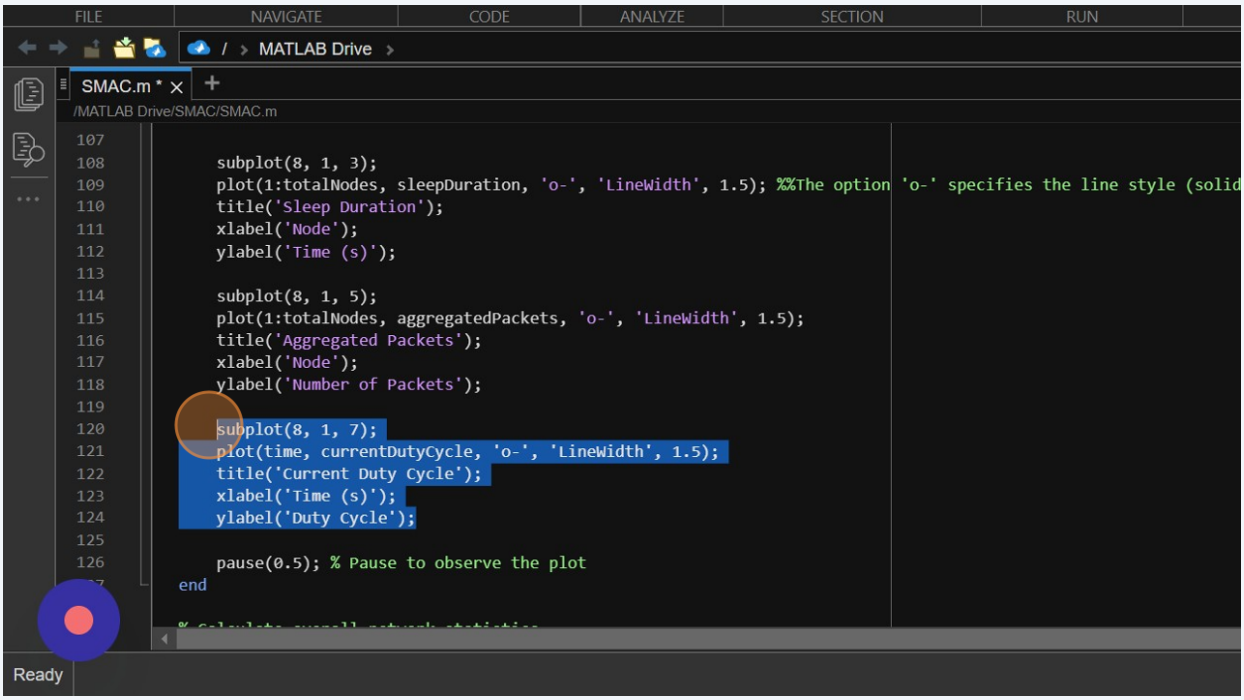
• plot(time, currentDutyCycle, 'o-', 'LineWidth', 1.5): This line plots the duty cycle over time. The x-axis represents time in seconds, and the y-axis represents the duty cycle. The data is plotted as a line with markers ('o') and a solid line connecting the markers ('-'). The line width is set to 1.5 for better visibility.

• title('Current Duty Cycle'): This sets the title of the subplot to "Current Duty Cycle".

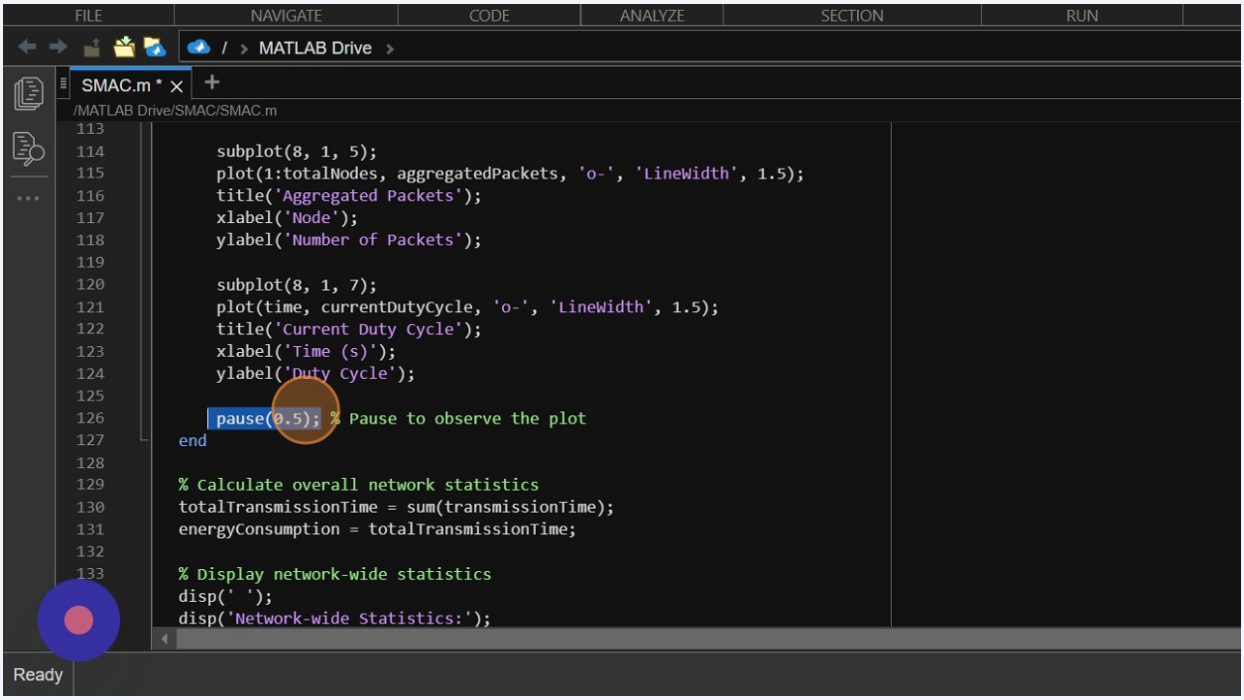
• xlabel('Time (s)'): This labels the x-axis as "Time (s)", indicating the time in seconds.

• ylabel('Duty Cycle'): This labels the y-axis as "Duty Cycle", indicating the proportion of time nodes spend in the active state relative to the synchronization interval.

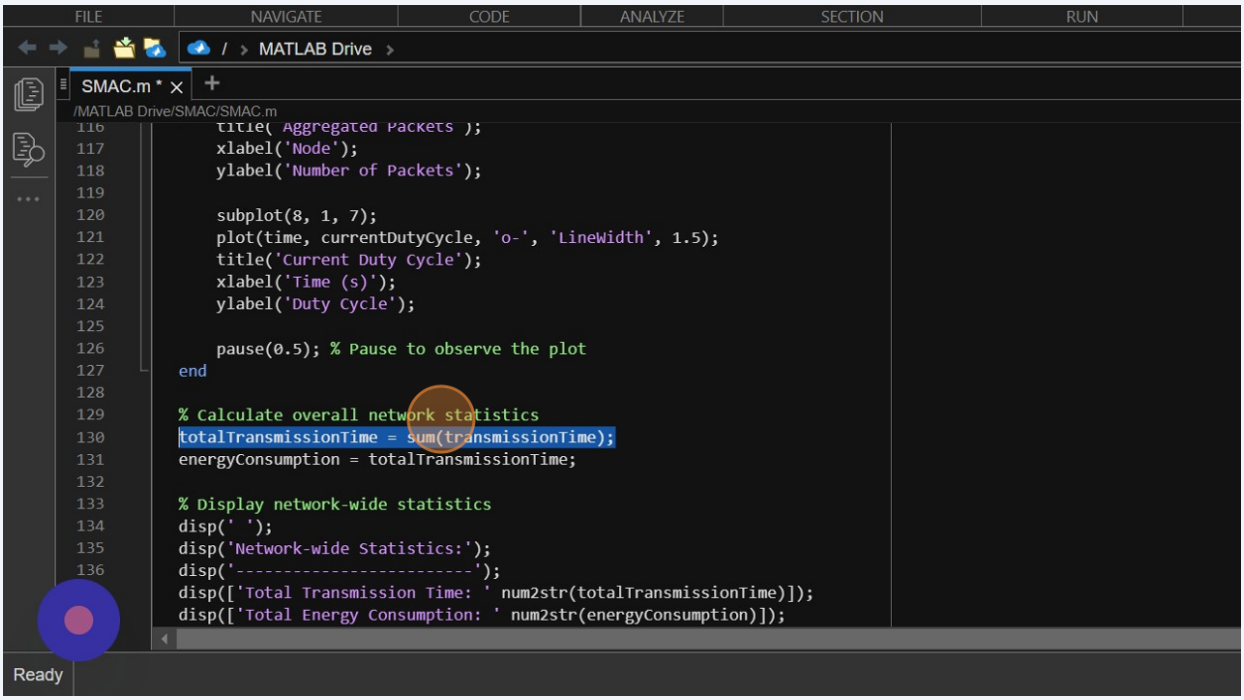
This subplot allows visualization of how the duty cycle of the network changes over time.



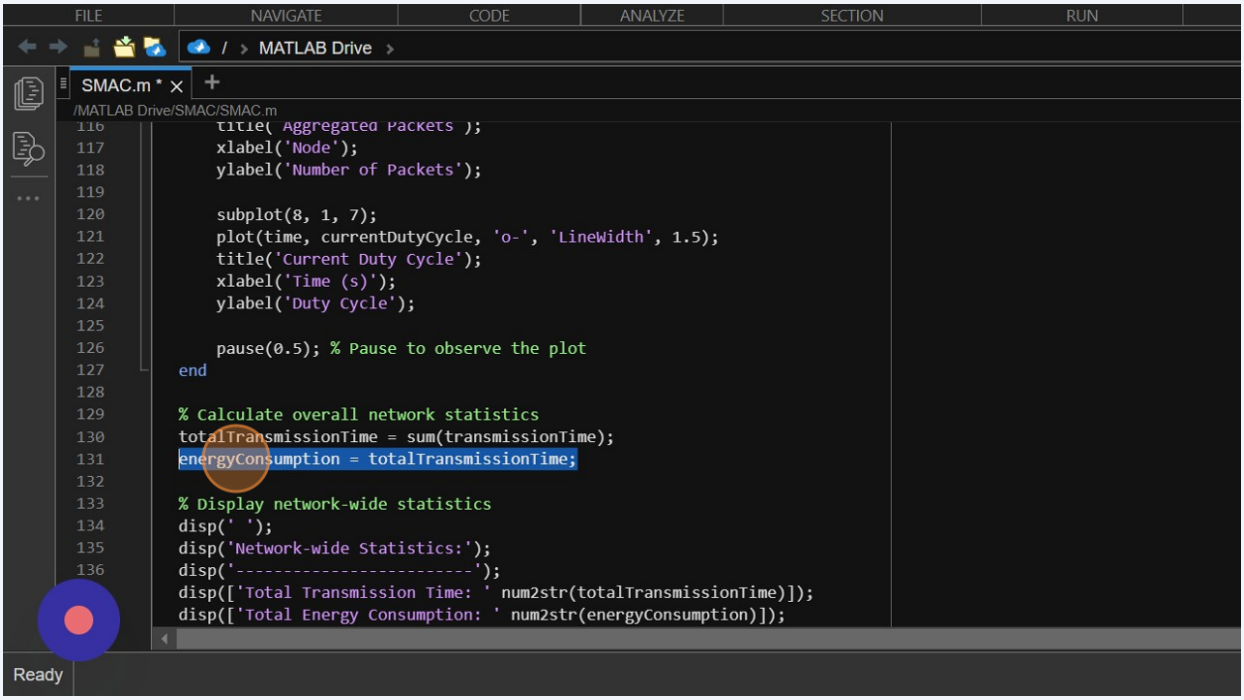
32 Pauses the execution for 0.5 seconds, allowing time to observe the plot.



33 The total transmission time here is calculated by the sum of all the transmission Time involved in the simulation

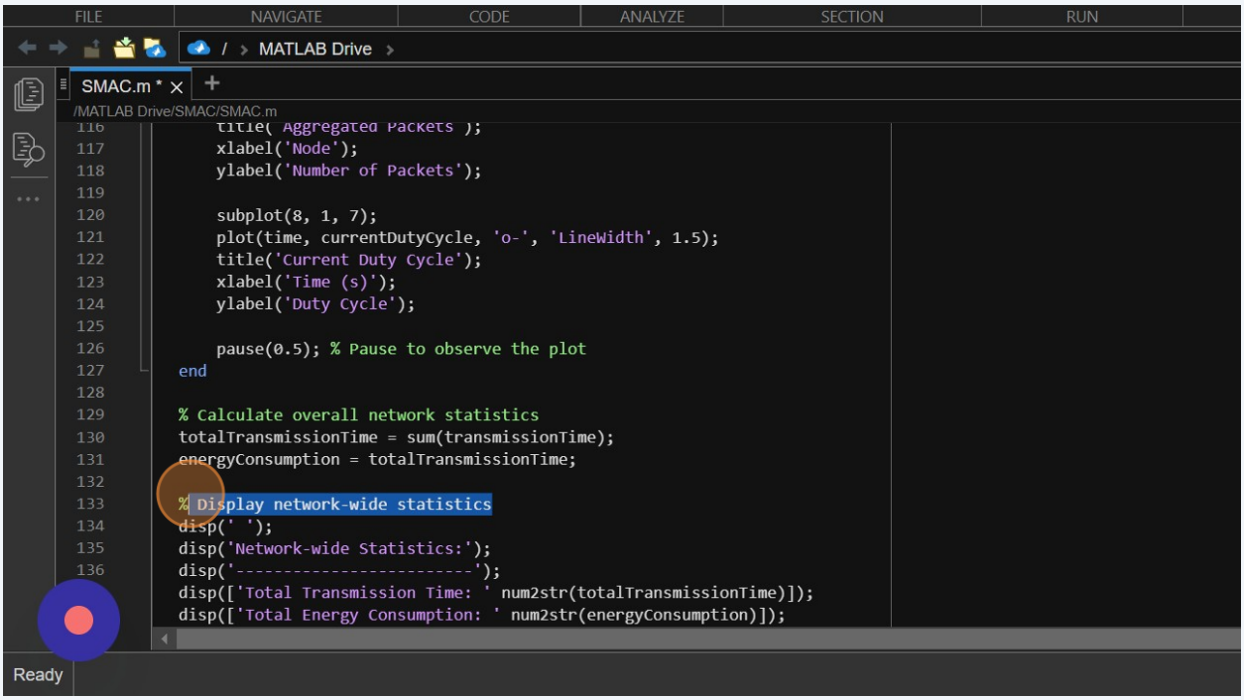


34 Then energy consumption will be obtained by the totalTransmissionTime

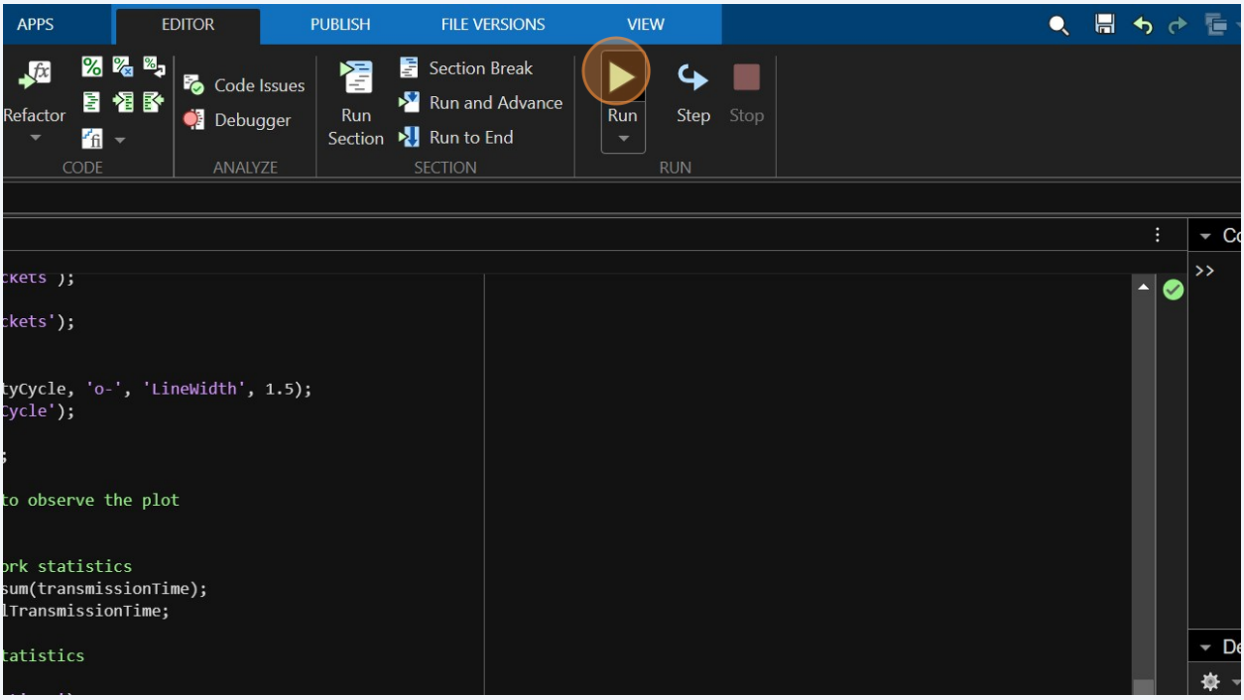


35 Now here is the code to display the overall statistics at the end of simulation

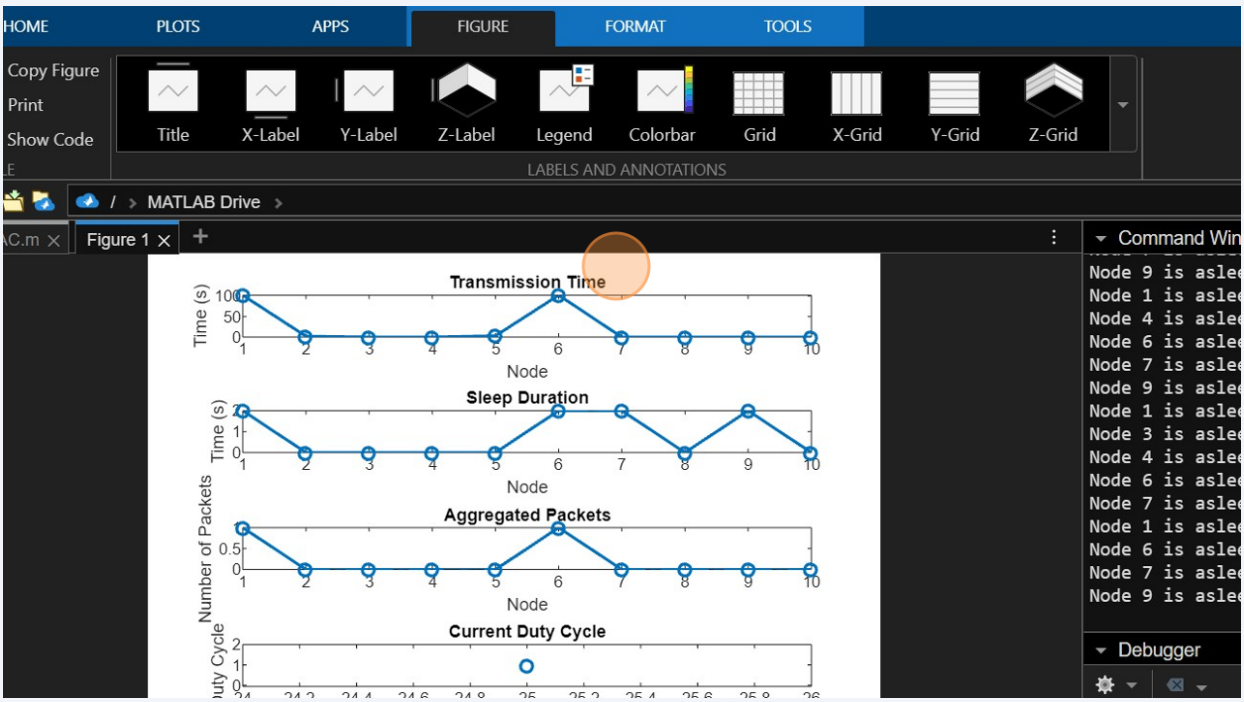
the header before printing the stats after that the two values previously calculated in 130 and 131 number line is displayed



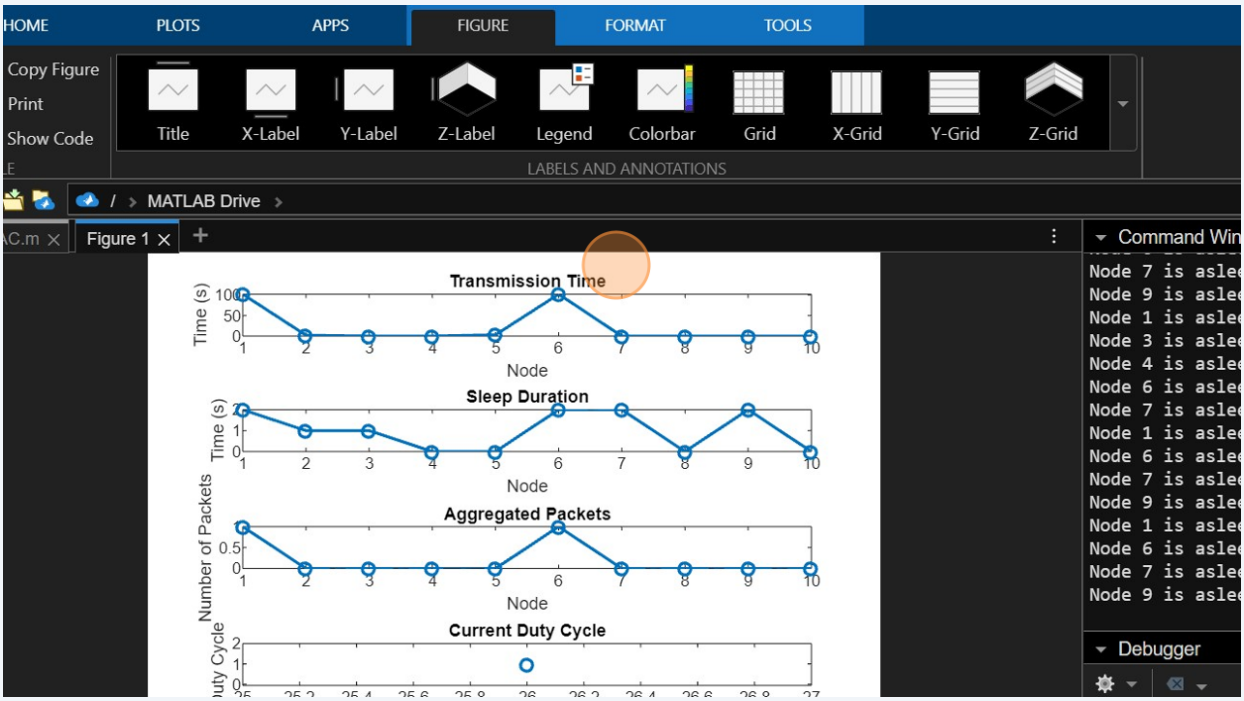
36 Now lets run it to see the simulation and output



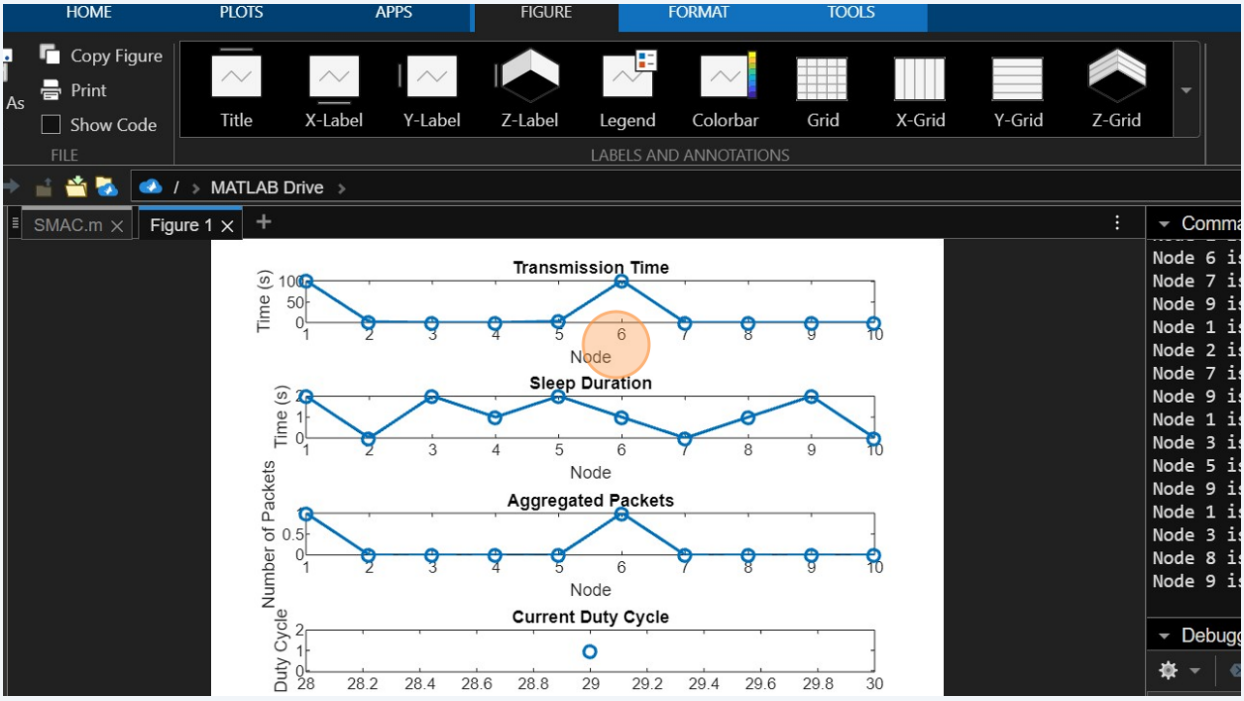
37 On the top of figure we plotted the first subplot as Transmission time and title was set to "Transmission Time"



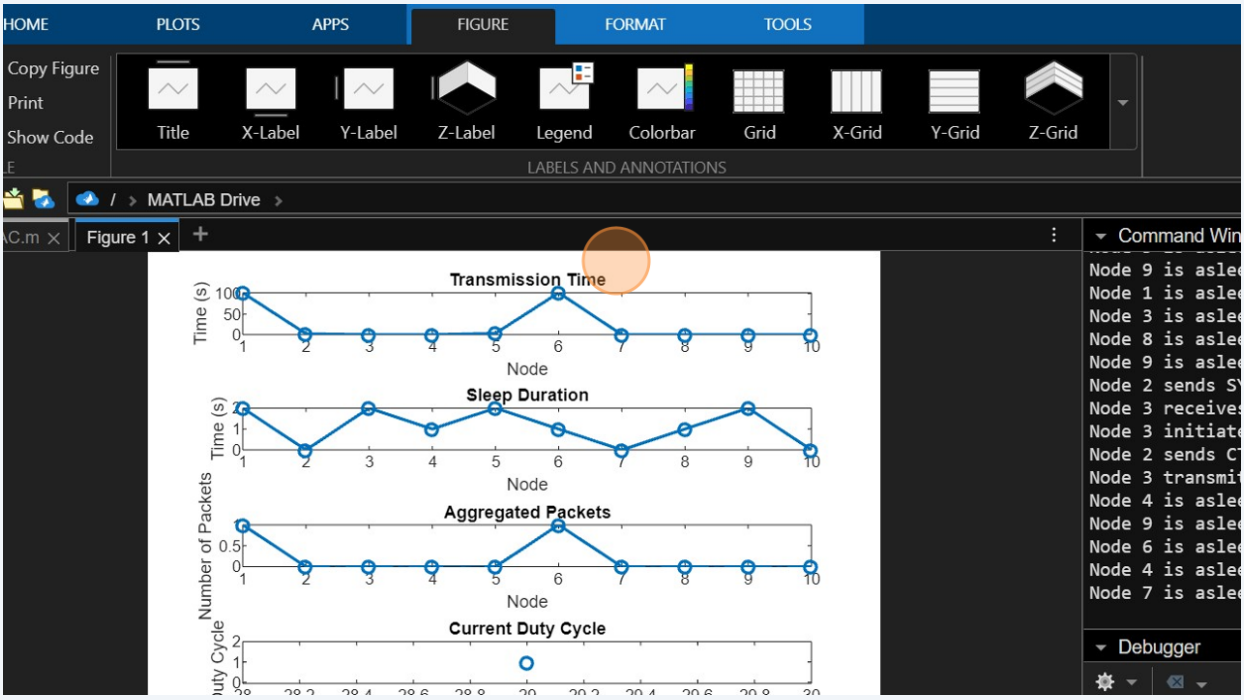
38 In the trasmission time graph (subplot) on y axis there is time displayed in seconds and in this figure circles are showing the nodes In this simulation there are 10 nodes We can take it on random positions as wel



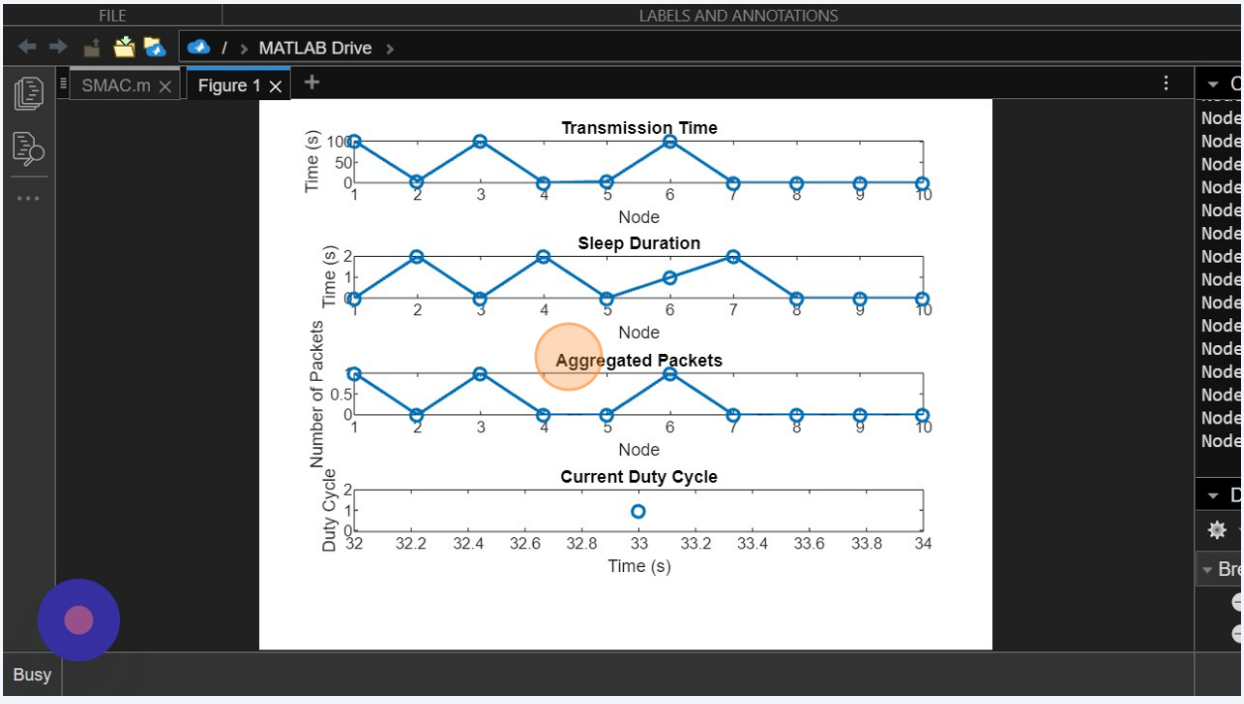
39 Next thing is sleep duration that is displayed here in the Subplot titled Sleep Duration and showing the sleep duration of each node in seconds that how much time the node is sleeping



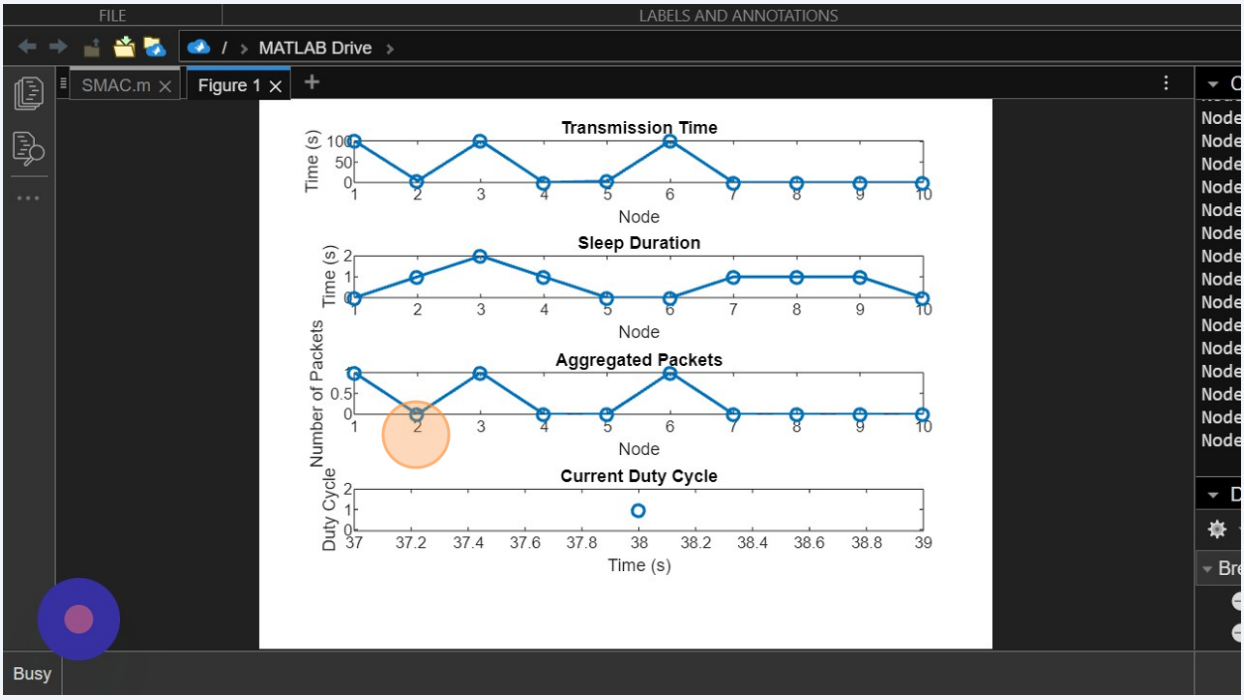
40 You can observe that for the first 3 graphs nodes are diaplayed on x axis



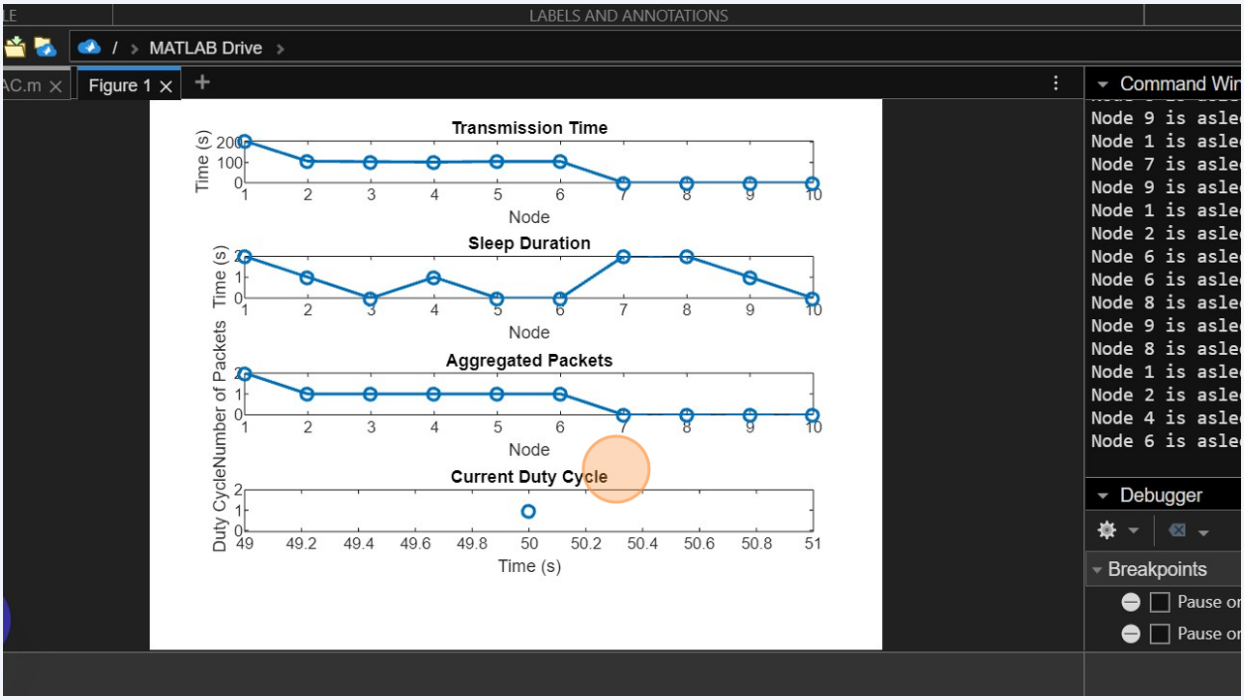
41 Then here comes the aggregated packets and in y axis here the number of packets are shown that each of 10 nodes is having how many aggregated packets



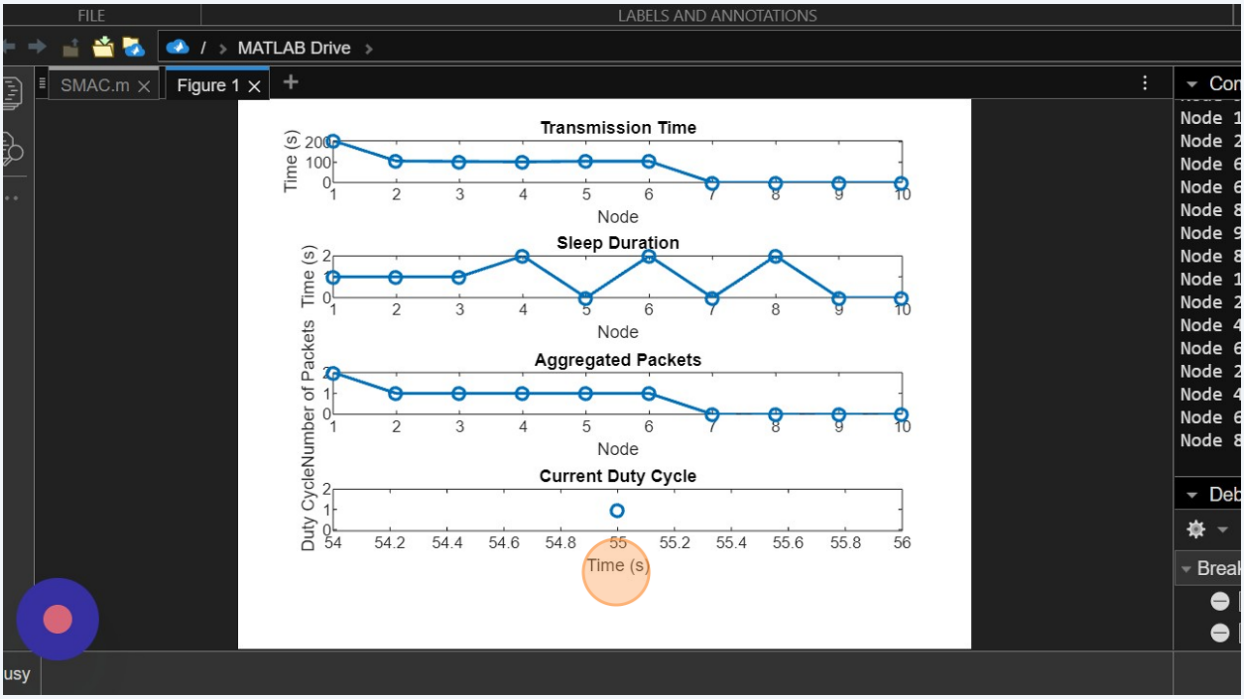
42 Here each node and its number of aggregated packets are shown like this one



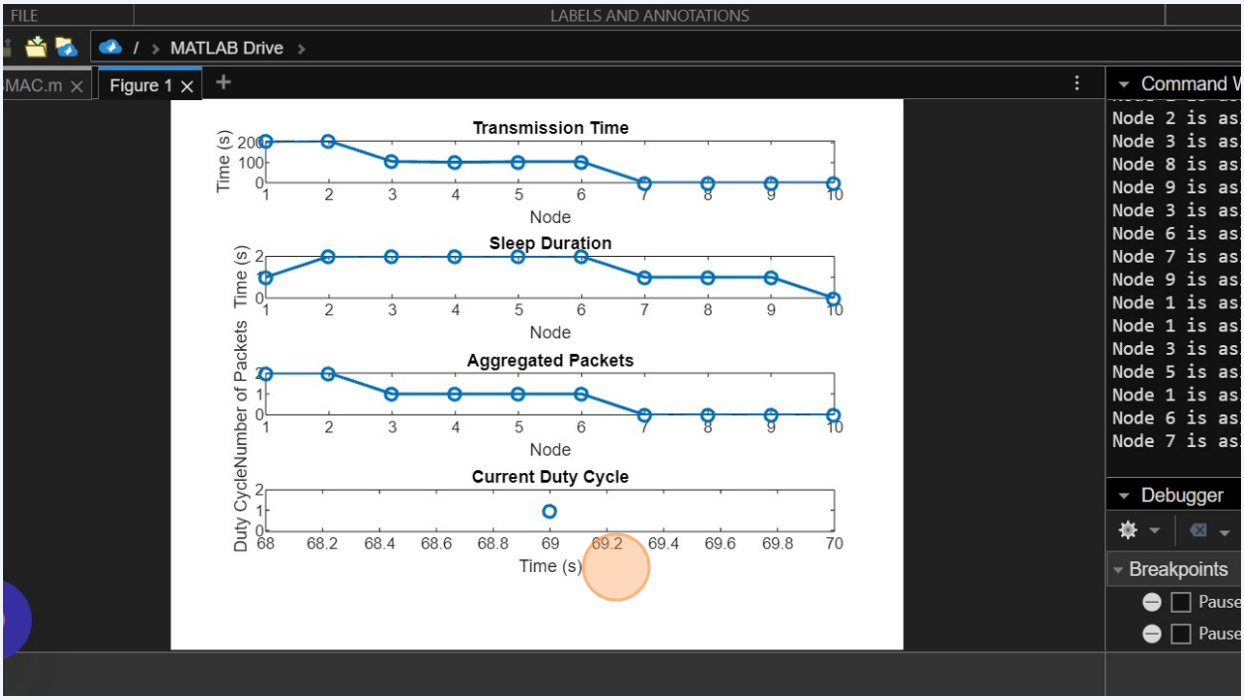
43 Then comes the Current Duty Cycle that is what we have set in the perimeters and initial was 0.5 and total dutycycles are 100



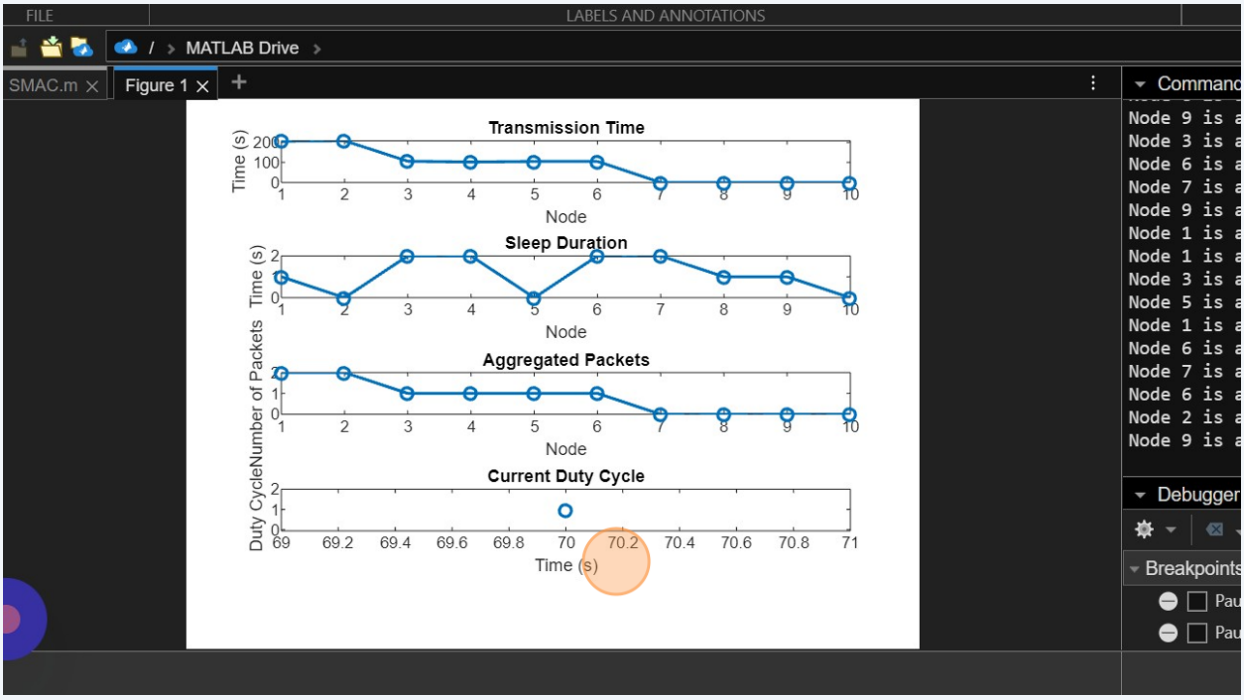
44 On the x axis of duty cycles time is shown bcz with time transmission occurs and duty cycles are changed by adding new to previous one



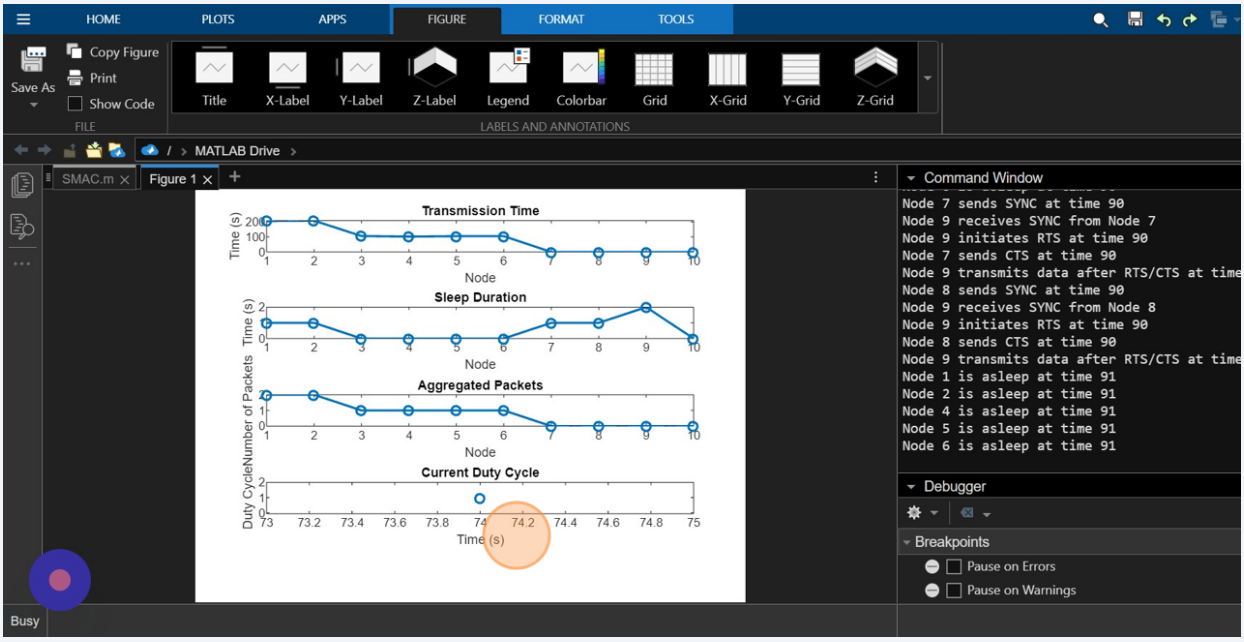
45 Again lets observe it at 69 duty cycles



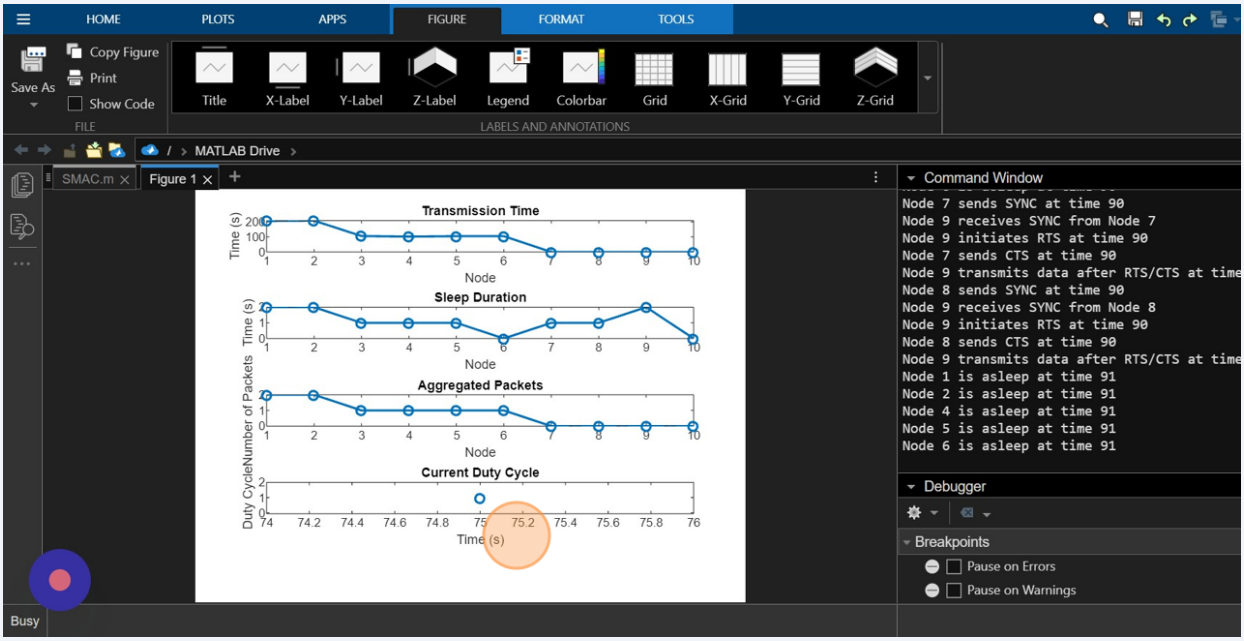
46 At 70th duty cycle



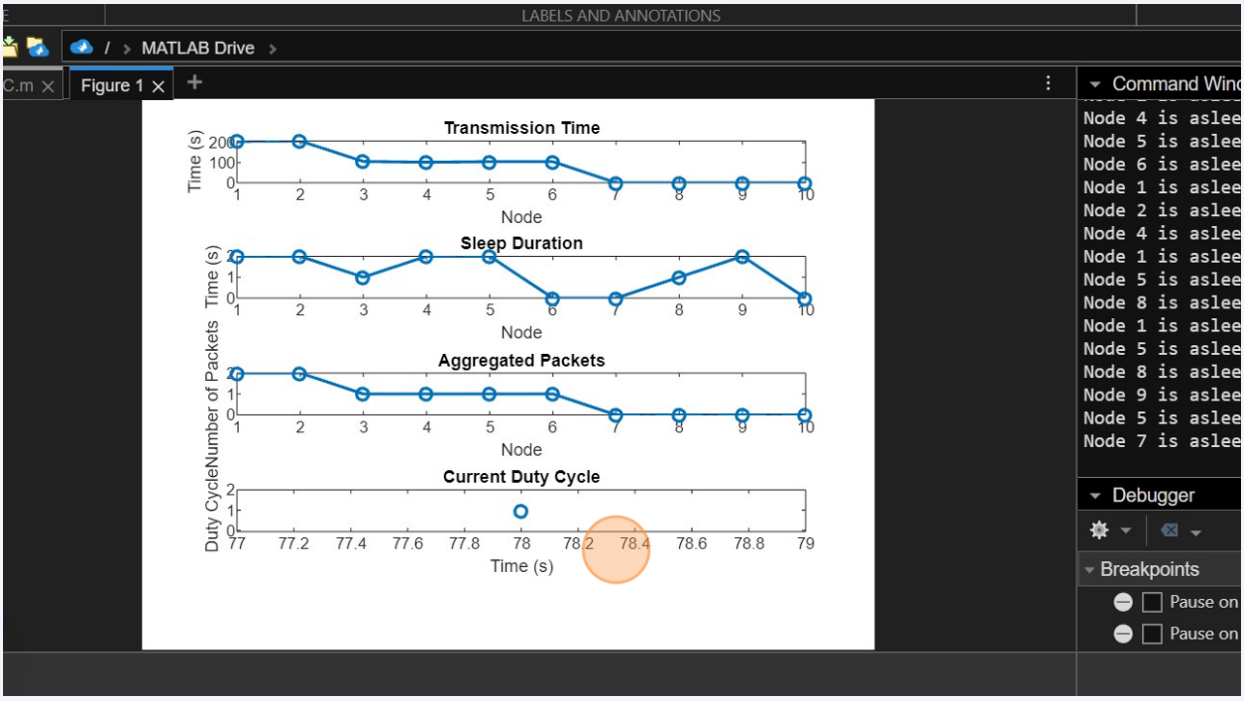
47 Now observing some other commands that were previously discussed in code and theory that node transmitting data is also displayed after the RTS CTS handshake and also the Sync packet recieved by nodes numbers



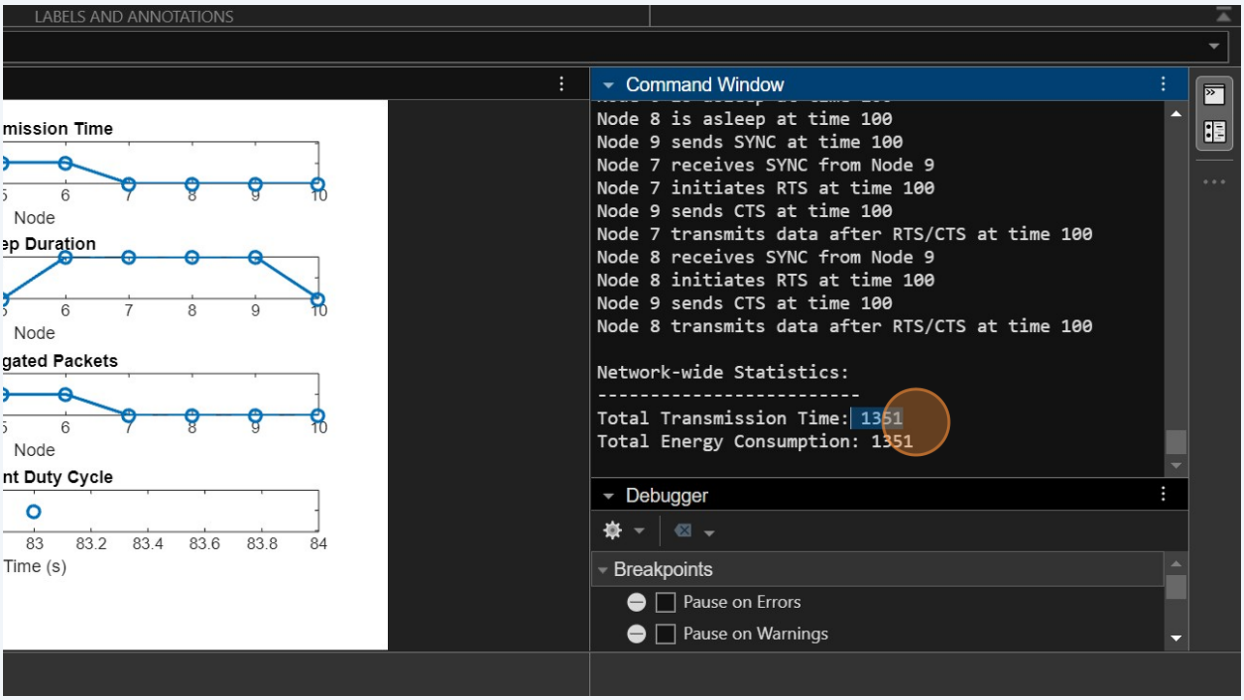
48 In the command window from start we can see that the nodes at sleep mode are displayed and their time to sleep is also shown



49 Click here.

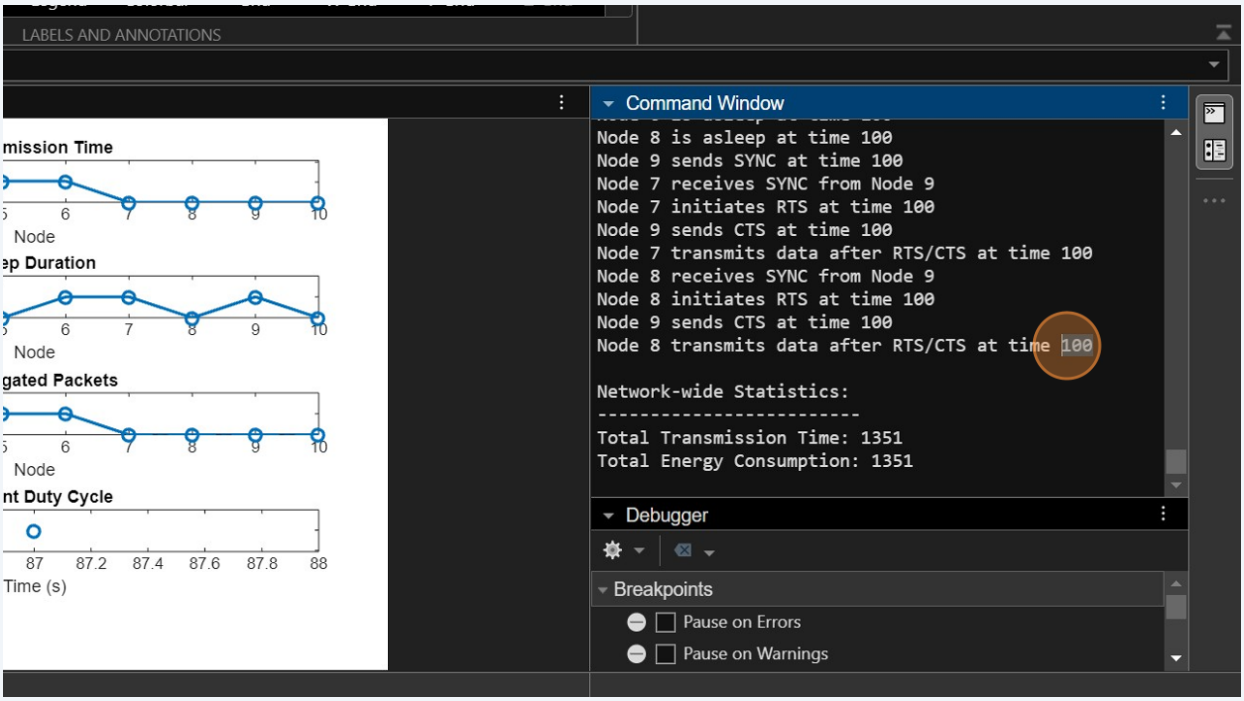


50 And after 100 duty cycles the total Network Wide Statistics are displayed as we have seen at the end of our code Both Total Transmission time as well as total energy consumption is calculated and displayed



51 You can observe the execution like

"Node 1 is asleep at time 100
Node 2 is asleep at time 100
Node 5 sends SYNC at time 100
Node 6 receives SYNC from Node 5
Node 6 initiates RTS at..."



52 Here you can see the completed 100 duty cycles as defined in the code and the execution is stopped here

53 Here is the code:

```
% SMAC (Sensors Medium Access Control) PROTOCOL FOR WIRELESS SENSOR NETWORKS%
% IMPLEMENTED BY LARAIB AZMAT AND ZAIN EJAZ

% Comprehensive S-MAC Simulation with Visualization in MATLAB

% Parameters
totalNodes = 10; % Total number of nodes in the network
simulationTime = 100; % Simulation time in seconds
packetSize = 100; % Size of each data packet in bytes
trafficLoad = 0.6; % Probability of a sensing event occurring

% Initialize node states
sleepDuration = zeros(1, totalNodes); % Array to store sleep duration for each node
transmissionTime = zeros(1, totalNodes); % Array to store transmission time for each node

% Packet aggregation parameters
aggregatedPackets = zeros(1, totalNodes); % Initialize an array to store aggregated packets for each node

% Dynamic sleep adjustment parameters
SleepPeriod = 3; % fixed sleep duration of node
syncInterval = 50; % Synchronization interval
AwakePeriods = syncInterval - SleepPeriod;

% Duty cycle parameters
currentDutyCycle = 0.1; % Initial duty cycle

% Synchronization parameters
syncDuration = 1; % Duration for which SYNC is transmitted
rtsThreshold = 0.5; % RTS threshold for initiating RTS/CTS
rtsDuration = 2; % Duration for which RTS is transmitted
ctsDuration = 1; % Duration for which CTS is transmitted
syncPeriod = 10; % Synchronization period for sending SYNC packets periodically

% Virtual cluster parameters
numVirtualClusters = 3; % Number of virtual clusters
nodesPerCluster = round(totalNodes / numVirtualClusters); % Number of nodes per virtual cluster

% Assign nodes to virtual clusters
virtualClusters = cell(1, numVirtualClusters); % Create a cell array to store virtual clusters
for i = 1:numVirtualClusters % Iterate over the number of virtual clusters
% Define the range of nodes for the current virtual cluster
virtualClusters{i} = (i-1)*nodesPerCluster + 1 : i*nodesPerCluster;
end
```

```
% Simulation loop
for time = 1:simulationTime % Loop over simulation time
for senderCluster = 1:numVirtualClusters % Loop over each virtual cluster as a
potential sender
clusterNodes = virtualClusters{senderCluster}; % Get the nodes within the sender
cluster
for j = 1:length(clusterNodes) % Loop over nodes within the cluster
senderNode = clusterNodes(j); % Get the node index
% Check if senderNode is a positive integer and within the range of totalNodes
if ~isscalar(senderNode) || senderNode < 1 || senderNode > totalNodes ||
mod(senderNode, 1) ~= 0
error('Invalid senderNode index: %d', senderNode);
end
% Check if a sensing event occurs
if rand() < trafficLoad
% Check if the node is awake before transmission
if sleepDuration(senderNode) == 0
% SYNC phase: Send SYNC packets periodically
if mod(time, syncPeriod) == 0
disp(['Node ' num2str(senderNode) ' sends SYNC at time ' num2str(time)]);
transmissionTime(senderNode) = transmissionTime(senderNode) + syncDuration;
% Broadcast SYNC packet to neighbors
for neighborNode = virtualClusters{senderCluster}
if neighborNode ~= senderNode && rand() < rtsThreshold
disp(['Node ' num2str(neighborNode) ' receives SYNC from Node '
num2str(senderNode)]);
% RTS phase: Neighbor initiates RTS
disp(['Node ' num2str(neighborNode) ' initiates RTS at time ' num2str(time)]);
transmissionTime(neighborNode) = transmissionTime(neighborNode) +
rtsDuration;
% Neighbor responds with CTS
disp(['Node ' num2str(senderNode) ' sends CTS at time ' num2str(time)]);
transmissionTime(senderNode) = transmissionTime(senderNode) + ctsDuration;
% Continue with data transmission
disp(['Node ' num2str(neighborNode) ' transmits data after RTS/CTS at time '
num2str(time)]);
transmissionTime(neighborNode) = transmissionTime(neighborNode) +
packetSize;
aggregatedPackets(neighborNode) = aggregatedPackets(neighborNode) + 1;
end
end
end
else
% Node is asleep, no transmission
disp(['Node ' num2str(senderNode) ' is asleep at time ' num2str(time)]);
end
else
% No sensing event, increase sleep duration
sleepDuration(senderNode) = sleepDuration(senderNode) + 1;
% Update sleep duration based
```