COMPUTER NETWORKS LAB
SEMESTER PROJECT

A PROJECT REPORT

submitted by

Vinayak Singhal (2021BITE032)

MD Kaishar (2021BITE036)

D.V.S.Lakshman (2021BITE034)

to

Prof. Iqra Altaf Gillani

(Instructor : Computer Networks Lab)

Department of Information Technology & Engineering

National Institute of Technology, Srinagar

19 July 2024

CONTENTS

Chapter 1

HOW TO RUN THE PROJECT

To run the project, you can use the `Prompt` class or the `NetworkSimulatorGUI` class.

1.1 USING PROMPT CLASS

```
javac Prompt.java
```

```
java Prompt
```

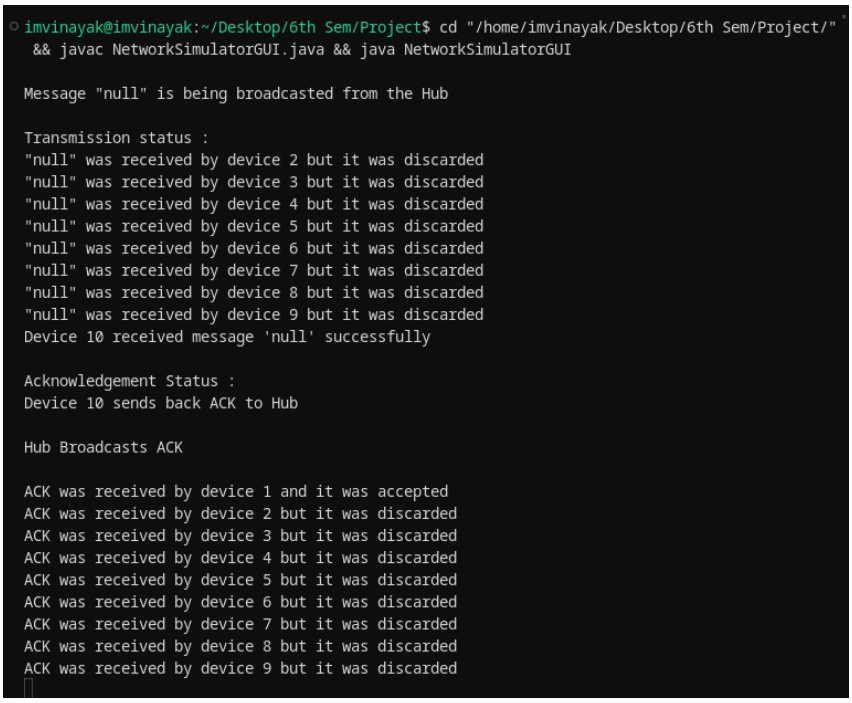


Figure 1.1: Program execution using terminal

1.2 USING NETWORKSIMULATORGUI CLASS

```
javac NetworkSimulatorGUI.java
```

```
java NetworkSimulatorGUI
```

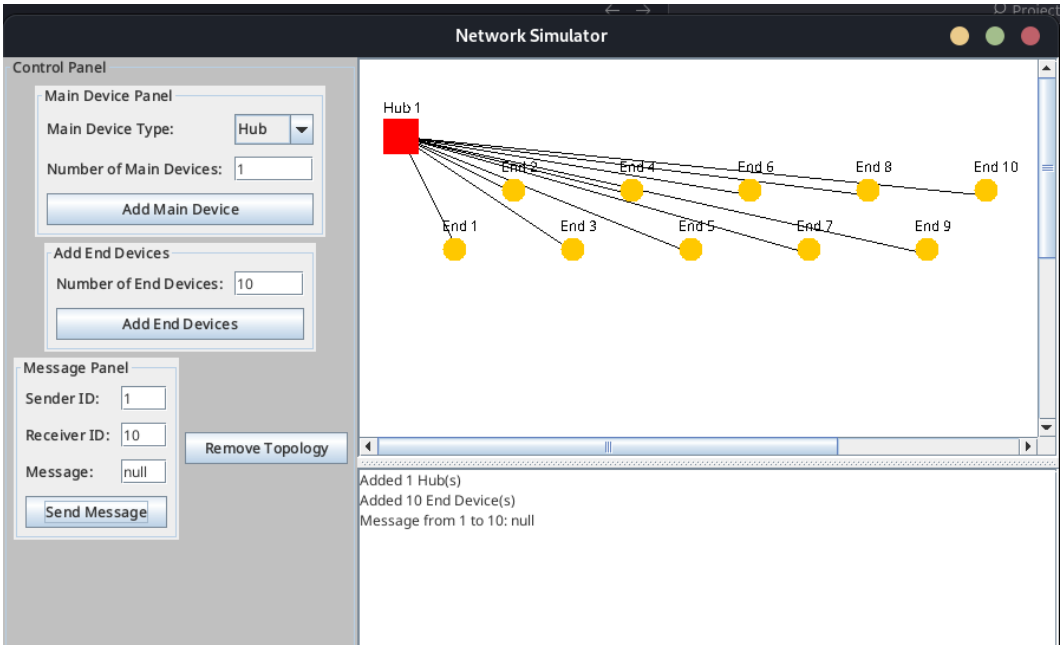


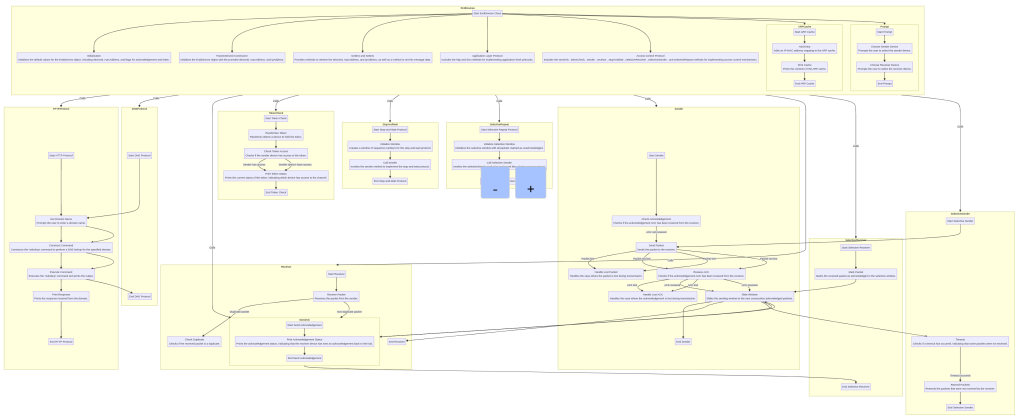
Figure 1.2: Program execution using terminal

Chapter 2

FILE DESCRIPTIONS

2.1 ENDDEVICES

EndDevices.java : This class represents end devices in the network. It includes attributes for device ID, MAC address, IP address, and methods for data handling, such as sending and receiving messages.



Here's a detailed Mermaid flow diagram with explanations for the provided code:Explanation:

1. The `EndDevices` class is the main entry point, which contains various methods and protocols implemented within the class.

2. The `HTTPProtocol` and `DNSProtocol` subgraphs represent the implementation of the HTTP and DNS protocols, respectively. They handle the user input, construct the necessary commands, execute them, and print the responses.

3. The `SendAck` subgraph represents the process of sending an acknowledgement (ACK) from the receiver device to the hub.

4. The `TokenCheck` subgraph represents the implementation of the token-based access control protocol, where the token is randomly assigned to different devices, and the sender device waits for its turn to access the channel.

5. The `StopAndWait` subgraph represents the implementation of the stop-and-wait protocol, where the sender sends a packet and waits for the acknowledgement before sending the next packet.

6. The `SelectiveRepeat` subgraph represents the implementation of the selective repeat protocol, where the sender sends multiple packets and the receiver selectively acknowledges the received packets.

7. The `Sender` and `Receiver` subgraphs represent the common functionality shared between the stop-and-wait and selective repeat protocols, such as sending packets, handling lost packets, and receiving acknowledgements.

8. The `SelectiveReceiver` subgraph represents the specific implementation of the receiver logic in the selective repeat protocol, including marking received packets, sliding the receiving window, and sending acknowledgements.

9. The `SelectiveSender` subgraph represents the specific implementation of the sender logic in the selective repeat protocol, including sending packets, handling lost packets and acknowledgements, sliding the sending window, and resending lost packets.

10. The `ARPCache` subgraph represents the functionality for managing the ARP cache, including adding new entries and printing the cache contents.

11. The `Prompt` subgraph represents the functionality for prompting the user to select the sender and receiver devices.

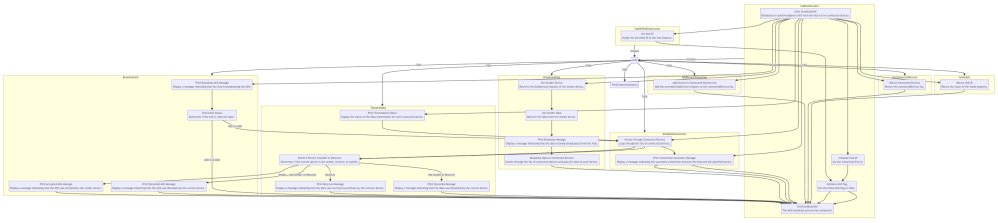
The flow diagram provides a comprehensive overview of the code's structure, functionality, and interactions between the different components. It can be used to understand the overall logic and flow of the `EndDevices` class and its associated protocols and mechanisms.

Created using codetoflow.com

Figure 2.1: Program execution using Graphical Interface

2.2 HUB

Hub.java : This class represents a network hub, which connects multiple end devices and facilitates data transmission between them. It includes methods for establishing connections, broadcasting messages, and handling acknowledgments.



Here's a detailed Mermaid flow diagram with explanations for the provided Hub class code:Explanation:

1. **HubConstructor**: This subgraph represents the default constructor for the Hub class. It initializes the hubId to 0 and the ack flag to false.

2. **HubWithIdConstructor**: This subgraph represents the constructor for the Hub class that takes an ID as a parameter. It sets the provided ID to the hubId property and initializes the ack flag to false.

3. **GetHubId**: This subgraph represents the getId() method, which returns the current value of the hubId property.

4. **GetConnectedDevices**: This subgraph represents the getDevices() method, which returns the list of connected EndDevices instances.

5. **AddDeviceToTopology**: This subgraph represents the topology() method, which adds a new EndDevices instance to the list of connected devices.

6. **EstablishConnection**: This subgraph represents the connection() method, which prints a message for each connected device indicating the established connection between the Hub and the device.

7. **PrintConnectionStatus**: This subgraph represents the printConnection() method, which prints a message indicating the successful connection between the Hub and a specific connected device.

8. **BroadcastData**: This subgraph represents the broadcast() method, which retrieves the data from the sender device, prints a message indicating the data broadcast, and then passes the data to all connected devices.

9. **TransmitData**: This subgraph represents the transmission() method, which prints the transmission status for each connected device, indicating whether the data was received, discarded, or accepted.

10. **BroadcastAck**: This subgraph represents the broadcastAck() method, which prints a message indicating that the Hub is broadcasting an ACK, checks the validity of the ACK, and then prints messages for each connected device indicating whether the ACK was discarded or accepted.

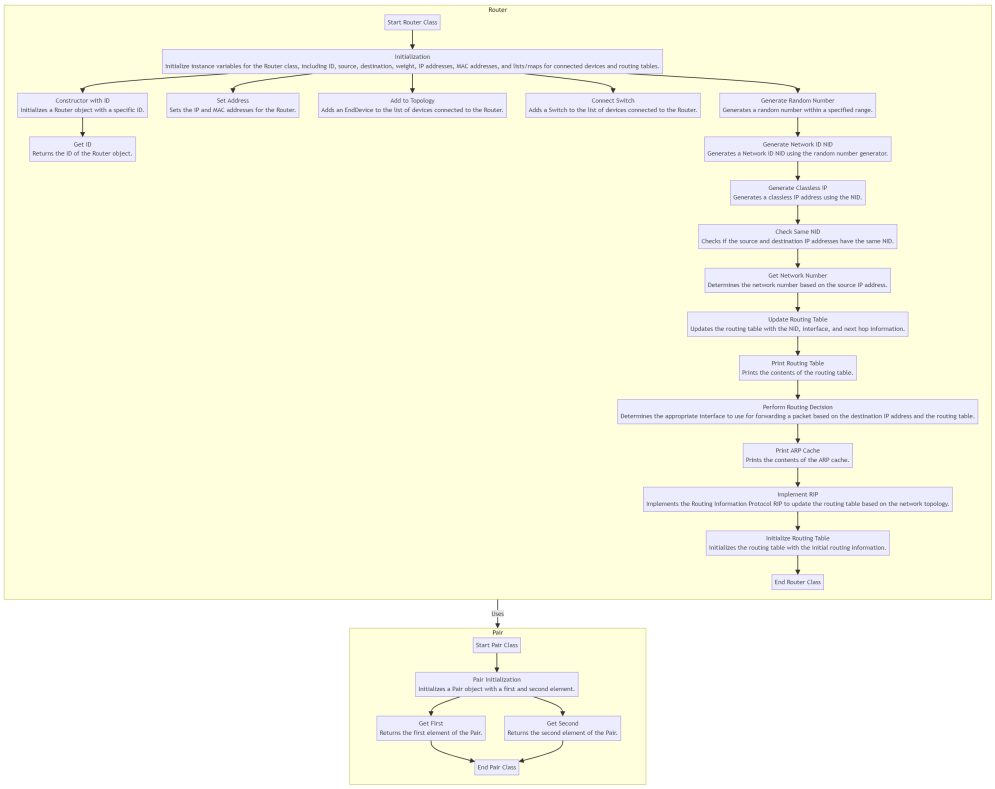
The flow diagram shows the relationships and interactions between the different methods of the Hub class, providing a comprehensive visual representation of the code's logic and functionality.

Created using codetoflow.com

Figure 2.2: Flowchart of Hub Class

2.3 ROUTER

Router.java : This class simulates a network router, which routes data packets between different networks. It includes methods for setting IP and MAC addresses, connecting to switches, generating NIDs, and maintaining routing tables.



Here's a detailed Mermaid flow diagram with explanations for the provided Router class code:Explanation of the Mermaid flow diagram:

1. **Router Class**:
- The diagram starts with the "Start Router Class" node, which represents the entry point of the Router class.
- The "Initialization" node sets up the initial state of the Router object, including its ID, source, destination, weight, IP addresses, MAC addresses, and various lists and maps for connected devices and routing tables.
- The "Constructor with ID" node initializes a Router object with a specific ID.
- The "Get ID" node returns the ID of the Router object.
- The "Set Address" node sets the IP and MAC addresses for the Router.
- The "Add to Topology" node adds an EndDevice to the list of devices connected to the Router.
- The "Connect Switch" node adds a Switch to the list of devices connected to the Router.
- The "Generate Random Number" node generates a random number within a specified range.
- The "Generate NID" node generates a Network ID (NID) using the random number generator.
- The "Generate Classless IP" node generates a classless IP address using the NID.
- The "Check Same NID" node checks if the source and destination IP addresses have the same NID.
- The "Get Network Number" node determines the network number based on the source IP address.
- The "Update Routing Table" node updates the routing table with the NID, interface, and next hop information.
- The "Print Routing Table" node prints the contents of the routing table.
- The "Perform Routing Decision" node determines the appropriate interface to use for forwarding a packet based on the destination IP address and the routing table.
- The "Print ARP Cache" node prints the contents of the ARP cache.
- The "Implement RIP" node implements the Routing Information Protocol (RIP) to update the routing table based on the network topology.
- The "Initialize Routing Table" node initializes the routing table with the initial routing information.
- The diagram ends with the "End Router Class" node.

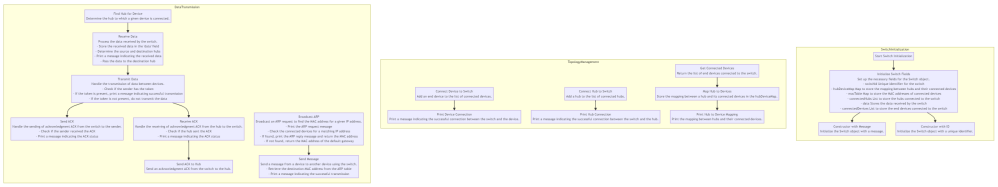
2. **Pair Class**:
- The diagram includes a subgraph for the Pair class, which is used within the Router class.
- The "Start Pair Class" node represents the entry point of the Pair class.
- The "Pair Initialization" node initializes a Pair object with a first and second element.
- The "Get First" node returns the first element of the Pair.
- The "Get Second" node returns the second element of the Pair.
- The diagram ends with the "End Pair Class" node.

The flow diagram provides a comprehensive visual representation of the Router class, including its key components, functionality, and the relationships between different parts of the code. The detailed explanations within the nodes help the user understand the purpose and behavior of each element in the code.

Figure 2.3: Flowchart of Router Class

2.4 SWITCH

Switch.java : This class represents a network switch that connects multiple devices within a local area network (LAN). It includes methods for topology management, device mapping, and handling MAC address tables.



Here is a detailed Mermaid flow diagram with explanations for the provided Switch class code:Explanation of the Mermaid flow diagram:

1. **SwitchInitialization**:
- This subgraph represents the initialization of the Switch object.
- It includes the initialization of the various fields, such as `switchId`, `hubDeviceMap`, `macTable`, `connectedHubs`, `data`, and `connectedDevices`.
- The subgraph also covers the two constructors: one with a message and one with an ID.

2. **TopologyManagement**:
- This subgraph handles the management of the switch's topology, including connecting devices and hubs to the switch.
- The `ConnectDevice` and `ConnectHub` functions add the end devices and hubs to the respective lists.
- The `PrintDeviceConnection` and `PrintHubConnection` functions print messages indicating the successful connection between the switch and the device/hub.
- The `GetConnectedDevices` function returns the list of end devices connected to the switch.
- The `MapHubToDevices` function stores the mapping between hubs and their connected devices in the `hubDeviceMap`.
- The `PrintHubToDeviceMap` function prints the mapping between hubs and their connected devices.

3. **DataTransmission**:
- This subgraph covers the data transmission functionality of the switch.
- The `FindHubForDevice` function determines the hub to which a given device is connected.
- The `ReceiveData` function processes the data received by the switch, including storing the data, determining the source and destination hubs, and passing the data to the destination hub.
- The `TransmitData` function handles the transmission of data between devices, checking the token status and printing the transmission status.
- The `SendAck` function sends an acknowledgment (ACK) from the switch to the sender, printing the ACK status.
- The `ReceiveAck` function receives an acknowledgment (ACK) from the hub to the switch, printing the ACK status.
- The `SendAckToHub` function sends an acknowledgment (ACK) from the switch to the hub.
- The `BroadcastARP` function broadcasts an ARP request to find the MAC address for a given IP address, returning the MAC address.
- The `SendMessage` function sends a message from a device to another device using the switch, printing the successful transmission.

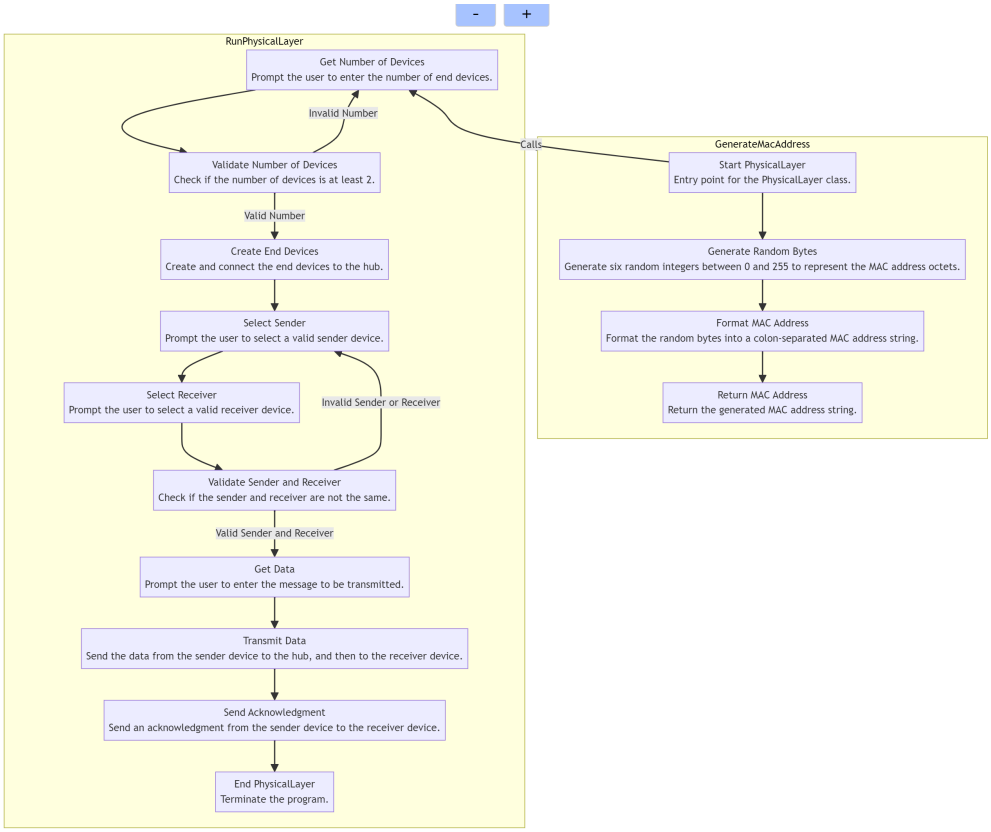
The Mermaid flow diagram provides a comprehensive visual representation of the Switch class's functionality, including the initialization, topology management, and data transmission processes. The detailed explanations within the subgraphs help to understand the purpose and behavior of each component of the code.

Created using codetoflow.com

Figure 2.4: Flowchart of Switch Class

2.5 PHYSICALLAYER

PhysicalLayer.java : This class simulates the physical layer of the network, handling the basic transmission of data between end devices. It includes methods for initializing end devices, selecting senders and receivers, and managing message transmission.



Here is a detailed Mermaid flow diagram with explanations for the provided code:Explanation of the Mermaid flow diagram:

1. **GenerateMacAddress Subgraph**:
- This subgraph represents the functionality of the `generateMacAddress()` method in the `PhysicalLayer` class.
- It starts with the `Start` node, which is the entry point for the method.
- The `GenerateRandomBytes` step generates six random integers between 0 and 255 to represent the MAC address octets.
- The `FormatMacAddress` step formats the random bytes into a colon-separated MAC address string.
- Finally, the `Return` step returns the generated MAC address string.

2. **RunPhysicalLayer Subgraph**:
- This subgraph represents the main functionality of the `run()` method in the `PhysicalLayer` class.
- It starts with the `Start` node, which is the entry point for the method.
- The `GetNumDevices` step prompts the user to enter the number of end devices.
- The `ValidateNumDevices` step checks if the number of devices is at least 2. If the number is invalid, it goes back to the `GetNumDevices` step.
- The `CreateEndDevices` step creates the end devices and connects them to the hub.
- The `SelectSender` and `SelectReceiver` steps prompt the user to select a valid sender and receiver device, respectively.
- The `ValidateSenderReceiver` step checks if the selected sender and receiver are not the same. If they are the same, it goes back to the `SelectSender` step.
- The `GetData` step prompts the user to enter the message to be transmitted.
- The `TransmitData` step sends the data from the sender device to the hub, and then to the receiver device.
- The `SendAcknowledgment` step sends an acknowledgment from the sender device to the receiver device.
- Finally, the `End` step terminates the program.

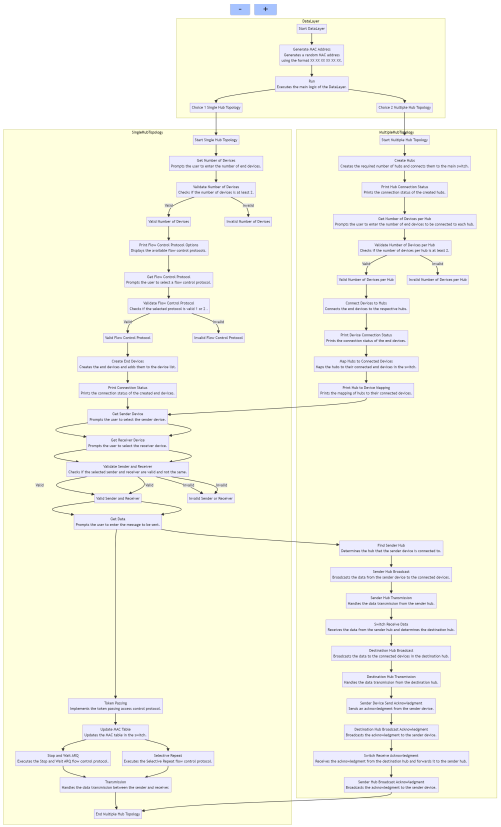
3. **Connections between Subgraphs**:
- The `RunPhysicalLayer` subgraph calls the `GenerateMacAddress` subgraph to generate a random MAC address.

The Mermaid flow diagram provides a detailed representation of the code's logic and structure, with explanations for each step, decision point, and function call. This diagram can help students understand the flow of the program and the purpose of each component.

Figure 2.5: Flowchart of Physical Layer Class

2.6 DATALAYER

DataLayer.java : This class simulates the data link layer of the network, handling the logical link control and media access control. It includes methods for generating MAC addresses, managing flow control protocols, and connecting devices through switches.



Here is a detailed Mermaid flow diagram with explanations for the provided code:The provided code implements a data layer that handles the communication between end devices, hubs, and switches in a network topology. The code supports two main scenarios: a single hub topology and a multiple hub topology.

1. **Single Hub Topology**:
- The user is prompted to enter the number of end devices to be created.
- The code validates the number of devices to ensure there are at least two.
- The available flow control protocols (Stop and Wait ARQ and Selective Repeat) are displayed, and the user selects one.
- The end devices are created, and their connection status is printed.
- The user selects the sender and receiver devices, and the message to be sent is obtained.
- The token passing access control protocol is implemented, and the MAC table in the switch is updated.
- Depending on the selected flow control protocol, either the Stop and Wait ARQ or Selective Repeat protocol is executed, and the data transmission between the sender and receiver is handled.

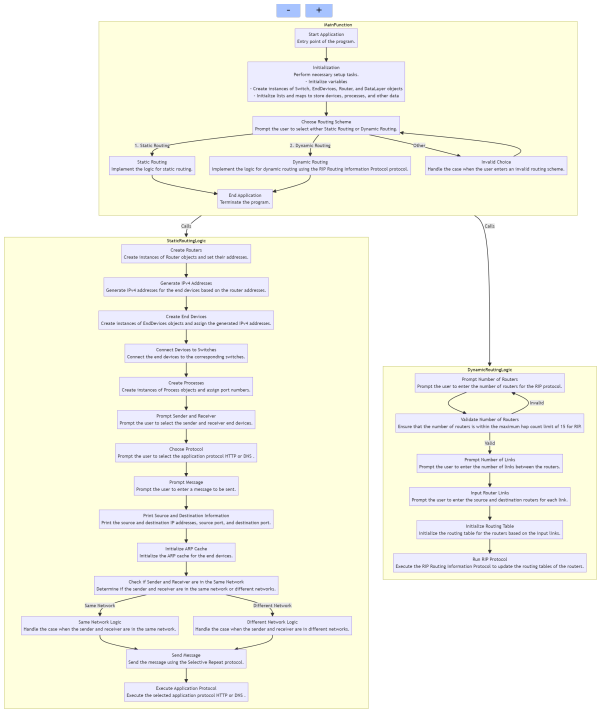
2. **Multiple Hub Topology**:
- The required number of hubs are created and connected to the main switch.
- The connection status of the hubs is printed.
- The user is prompted to enter the number of end devices to be connected to each hub.
- The end devices are connected to the respective hubs, and their connection status is printed.
- The mapping of hubs to their connected end devices is established in the switch, and this mapping is printed.
- The user selects the sender and receiver devices, and the message to be sent is obtained.
- The hub that the sender device is connected to is determined.
- The sender hub broadcasts the data to the connected devices, and the data transmission from the sender hub is handled.
- The switch receives the data from the sender hub and determines the destination hub.
- The destination hub broadcasts the data to the connected devices, and the data transmission from the destination hub is handled.
- The sender device sends an acknowledgment, which is broadcasted by the destination hub and received by the switch.
- The switch forwards the acknowledgment to the sender hub, which then broadcasts it to the sender device.

The Mermaid flow diagram provides a detailed representation of the code's logic and structure, with explanations for each key component and step. The subgraphs encapsulate the functionality of the single hub and multiple hub topologies, making it easier to understand the flow and interactions between the different entities (end devices, hubs, and switch) in the network.

Figure 2.6: Flowchart of Data Layer Class

2.7 NETWORKLAYER

NetworkLayer.java: This class simulates the network layer of the network, responsible for routing data packets between devices. It includes methods for selecting routing schemes, initializing routers, and managing ARP and routing tables.



Here's a detailed Mermaid flow diagram with explanations for the provided code:Explanation of the Mermaid flow diagram:

1. **MainFunction**:
- The `Start` node represents the entry point of the program.
- The `Initialization` node performs the necessary setup tasks, such as initializing variables, creating instances of various objects, and initializing lists and maps.
- The `ChooseRoutingScheme` node prompts the user to select either Static Routing or Dynamic Routing.
- Based on the user's choice, the flow goes to either the `StaticRouting` or `DynamicRouting` node.
- If the user enters an invalid choice, the flow goes to the `InvalidChoice` node, which redirects back to the `ChooseRoutingScheme` node.
- The `End` node represents the termination of the program.

2. **StaticRoutingLogic**:
- The `CreateRouters` node creates instances of Router objects and sets their addresses.
- The `GenerateIPv4Addresses` node generates IPv4 addresses for the end devices based on the router addresses.
- The `CreateEndDevices` node creates instances of EndDevices objects and assigns the generated IPv4 addresses.
- The `ConnectDevicesToSwitches` node connects the end devices to the corresponding switches.
- The `CreateProcesses` node creates instances of Process objects and assigns port numbers.
- The `PromptSenderReceiver` node prompts the user to select the sender and receiver end devices.
- The `ChooseProtocol` node prompts the user to select the application protocol (HTTP or DNS).
- The `PromptMessage` node prompts the user to enter a message to be sent.
- The `PrintSourceDestinationInfo` node prints the source and destination IP addresses, source port, and destination port.
- The `InitializeARPCache` node initializes the ARP cache for the end devices.
- The `CheckSameNetwork` node determines if the sender and receiver are in the same network or different networks.
- The `SameNetworkLogic` node handles the case when the sender and receiver are in the same network.
- The `DifferentNetworkLogic` node handles the case when the sender and receiver are in different networks.
- The `SendMessage` node sends the message using the Selective Repeat protocol.
- The `ExecuteApplicationProtocol` node executes the selected application protocol (HTTP or DNS).

3. **DynamicRoutingLogic**:
- The `PromptNumRouters` node prompts the user to enter the number of routers for the RIP protocol.
- The `ValidateNumRouters` node ensures that the number of routers is within the maximum hop count limit of 15 for RIP.
- The `PromptNumLinks` node prompts the user to enter the number of links between the routers.
- The `InputRouterLinks` node prompts the user to enter the source and destination routers for each link.
- The `InitializeRoutingTable` node initializes the routing table for the routers based on the input links.
- The `RunRIPProtocol` node executes the RIP (Routing Information Protocol) to update the routing tables of the routers.

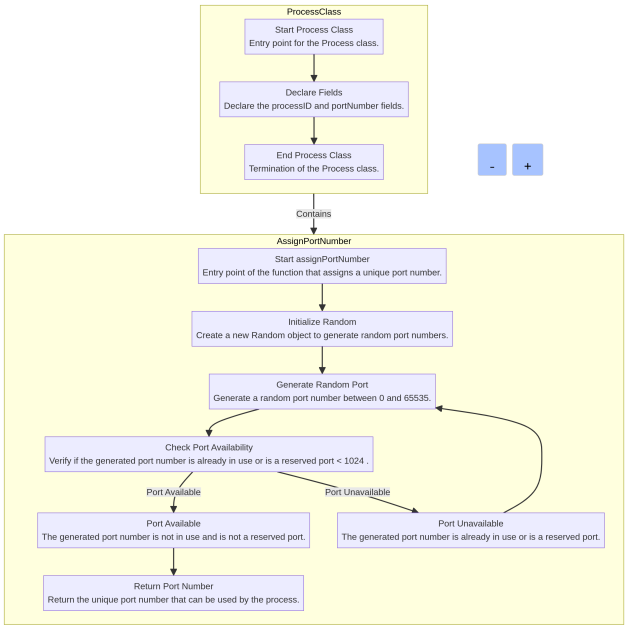
The Mermaid flow diagram provides a comprehensive visual representation of the code's logic and structure, with detailed explanations for each key component and decision point. The subgraphs and annotations help to clearly delineate the different functionalities and interactions within the NetworkLayer class.

Created using codetoflow.com

Figure 2.7: Flowchart of Network Layer Class

2.8 PROCESS

Process.java : This class represents a network process that assigns random port numbers to network connections. It includes a method for generating unique port numbers and ensuring they are not duplicated.



Here is the Mermaid flow diagram with detailed explanations for the provided code:Explanation of the Mermaid flow diagram:

1. **Process Class**:
- The Process class is the main entry point for the code.
- It declares two fields: `processID` and `portNumber`.

2. **assignPortNumber() Function**:
- The `assignPortNumber()` function is responsible for generating a unique port number for a process.
- It starts by initializing a new `Random` object to generate random port numbers.
- The function then enters a loop to generate a random port number between 0 and 65535.
- Inside the loop, the function checks if the generated port number is already in use (i.e., present in the `processMap`) or if it is a reserved port (less than 1024).
- If the port is available (not in use and not a reserved port), the function returns the generated port number.
- If the port is unavailable, the function generates a new random port number and continues the loop until a unique port number is found.

The Mermaid flow diagram provides a detailed representation of the code's logic and structure. It includes the following key elements:

1. **Subgraph for `assignPortNumber()` Function**:
- This subgraph represents the flow and logic of the `assignPortNumber()` function.
- It includes the individual steps, such as initializing the `Random` object, generating a random port number, checking the port availability, and returning the unique port number.
- Each step is accompanied by a detailed explanation of its purpose and functionality.

2. **Subgraph for the `Process` Class**:
- This subgraph represents the structure and declaration of the `Process` class.
- It includes the entry point, the declaration of the `processID` and `portNumber` fields, and the termination of the class.

3. **Connections between Subgraphs**:
- The diagram shows the relationship between the `Process` class and the `assignPortNumber()` function, indicating that the `assignPortNumber()` function is contained within the `Process` class.

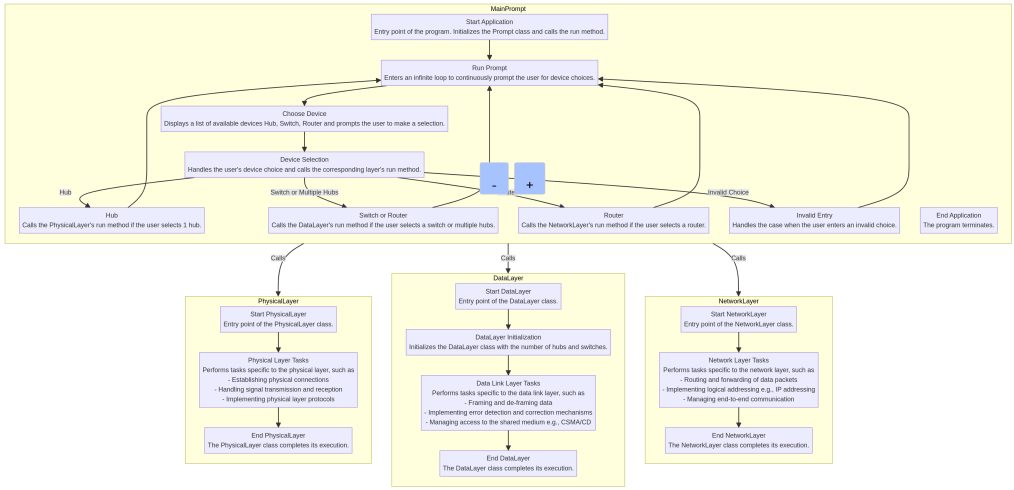
The Mermaid flow diagram provides a comprehensive and detailed representation of the code's logic and structure, making it easier for the user to understand the functionality and flow of the provided code.

Created using codetoflow.com

Figure 2.8: Flowchart of Process Class

2.9 PROMPT

Prompt.java : This class provides a command-line interface for interacting with the network simulator. It includes methods for running the simulation, selecting devices, and managing user inputs.



Here is a detailed Mermaid flow diagram with explanations for the provided Java code:Explanation of the Mermaid flow diagram:

1. The main entry point of the application is the `Prompt` class, which contains the `run()` method. This method is responsible for the overall flow of the program.

2. The `run()` method starts by initializing a `Scanner` object to receive user input. It then enters an infinite loop to continuously prompt the user for device choices.

3. Inside the loop, the code displays a list of available devices (Hub, Switch, Router) and prompts the user to make a selection.

4. Based on the user's choice, the code branches to the corresponding layer's `run()` method:
- If the user selects a Hub, the `PhysicalLayer` class is called.
- If the user selects a Switch or multiple Hubs, the `DataLayer` class is called.
- If the user selects a Router, the `NetworkLayer` class is called.
- If the user enters an invalid choice, the code handles the error and prompts the user again.

5. Each layer class (`PhysicalLayer`, `DataLayer`, and `NetworkLayer`) has its own subgraph in the Mermaid flow diagram, representing the tasks and functionality specific to that layer.

6. The `PhysicalLayer` class is responsible for establishing physical connections, handling signal transmission and reception, and implementing physical layer protocols.

7. The `DataLayer` class is responsible for framing and de-framing data, implementing error detection and correction mechanisms, and managing access to the shared medium (e.g., CSMA/CD).

8. The `NetworkLayer` class is responsible for routing and forwarding data packets, implementing logical addressing (e.g., IP addressing), and managing end-to-end communication.

9. The flow diagram shows the interactions between the `Prompt` class and the various layer classes, with the `Prompt` class calling the appropriate layer's `run()` method based on the user's device choice.

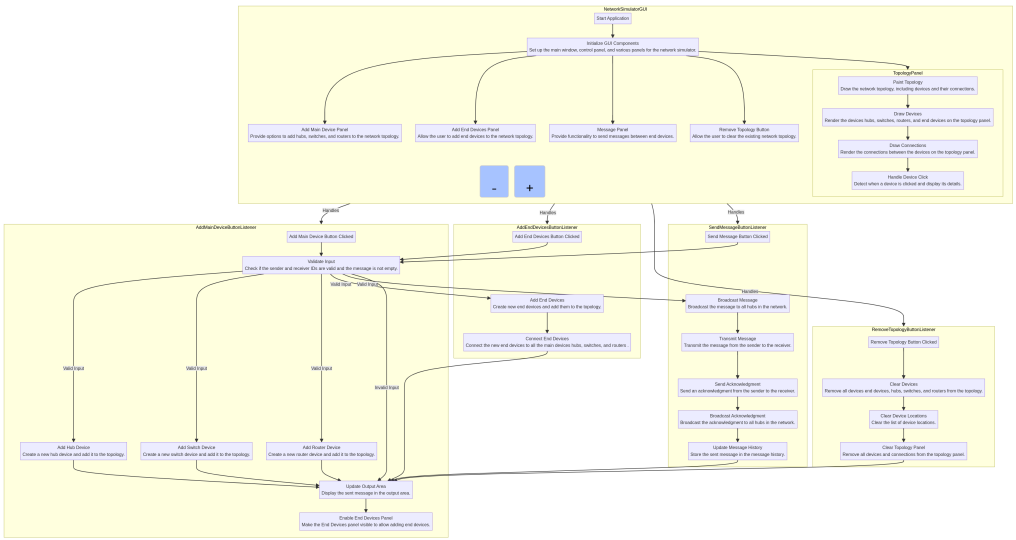
10. The flow diagram is structured in a way that avoids any cycles or self-referencing relationships, ensuring a clear and linear flow of execution.

Overall, this Mermaid flow diagram provides a detailed and comprehensive representation of the code's logic and structure, making it easier for the user to understand the functionality and flow of the application.

Figure 2.9: Flowchart of Prompt Class

2.10 NETWORKSIMULATORGUI

NetworkSimulatorGUI.java : This class provides a graphical user interface (GUI) for the network simulator. It includes components for adding devices, sending messages, and visualizing the network topology.



Explanation:

1. The `NetworkSimulatorGUI` class is the main entry point of the application, responsible for initializing and managing the various components of the network simulator GUI.

2. The `AddMainDeviceButtonListener` handles the addition of main devices (hubs, switches, and routers) to the network topology. It validates the input, creates the appropriate device instances, and updates the topology panel accordingly.

3. The `AddEndDevicesButtonListener` handles the addition of end devices to the network topology. It validates the input, creates the end devices, and connects them to all the main devices in the topology.

4. The `RemoveTopologyButtonListener` handles the removal of the entire network topology, clearing all devices, device locations, and the topology panel.

5. The `SendMessageButtonListener` handles the sending of messages between end devices. It validates the input, broadcasts the message to the hubs, transmits the message to the receiver, sends an acknowledgment, and updates the message history and output area.

6. The `TopologyPanel` is responsible for rendering the network topology, including the devices and their connections. It handles device clicks and displays the details of the clicked device.

The flow diagram provides a detailed breakdown of the functionality and interactions between the various components of the `NetworkSimulatorGUI` application. It highlights the key steps, decision points, and data flows involved in adding main devices, adding end devices, removing the topology, and sending messages between devices.

The subgraphs represent the logical units of the application, such as the button listeners and the topology panel, and provide a granular view of the internal logic and processing within each unit. The explanations embedded within the diagram help to clarify the purpose and functionality of each element, making it easier for the user to understand the overall structure and behavior of the network simulator application.

Figure 2.10: Flowchart of Network Simulator GUI Class

Chapter 3

CONCLUSION

This document provided an overview of the Network Simulator project, including descriptions of its components and usage examples. For more details, refer to the source code provided.

REFERENCES

[1] **W3Schools**, Java language tutorial, for Java Libraries and Classes, https://www.w3schools.com/java/

[2] **Youtube**, Java Swing tutorial for developing the GUI for this project https://youtu.be/Kmgo00avvEw?si=vzm9yV-1MfaiPHkH

[3] **Behrouz A. Forouzan** Theory related to variours functionalities added in this project.

[4] **Java Swing Documentation**, Used it throughly for learning Java Swing https://docs.oracle.com/javase/tutorial/uiswing/events/actionlistener.html

[5] **FlowChart**, For visualizing the flow of code, for better understanding https://codetoflow.com/