Computer Networks Lab Semester Project

SUBMITTED BY : SONIA, RISHITA SHARMA

ENROLLMENTS : 2020BITE074 , 2020BITE031

Branch : IT

Semester : 6th

SUBMITTED TO : Dr.Iqra Altaf Gilani

The provided code includes several standard C++ libraries that are commonly used for various purposes. Here's an explanation of each included library:

<iostream>: This library provides input and output stream objects such as std::cin and std::cout for console input and output.

<string>: This library provides functionalities for working with strings, including string manipulation, concatenation, and comparison.

<unordered_map>: This library provides an implementation of an unordered associative container, std:: unordered_map, which allows efficient lookup of key-value pairs using a hash-based data structure.

<vector>: This library provides an implementation of a dynamic array, std::vector, which allows flexible storage and manipulation of elements.

<algorithm>: This library provides a collection of functions for performing common algorithms and operations on sequences, such as searching, sorting, and modifying elements.

<iterator>: This library provides various types of iterators that allow traversing and accessing elements of a container.

<random>: This library provides facilities for generating pseudo-random numbers.

<chrono>: This library provides utilities for measuring time, including high-resolution clocks and duration calculations.

<thread>: This library provides functionalities for creating and managing threads, enabling concurrent execution of code.

Network class that represents a network with a specific IP address range. Here's an explanation of the class and its members:

1. private section:

   - network_ip : A string variable that stores the IP address of the network.

   - **next_subnet** : An integer variable that keeps track of the next available subnet within the network.

2. public section:

   - Network(string ip) : A constructor that takes an IP address as a parameter and initializes the “network_ip” and “next_subnet” variables. It sets the initial value of “next_subnet” to 0.

   - **string assignIPAddress() :** A method that assigns a unique IP address to a device within the network. It increments the “next_subnet” value by 1 each time it is called, converts it to a string, and appends it to the “network_ip” to form the assigned IP address. The assigned IP address is then returned as a string.

RoutingTable class that represents a routing table containing static and dynamic routes. Here's an explanation of the class and its members:

1. private section:

   - **staticRoutes** : An unordered map that stores static routes in the routing table. It maps a destination IP address to the next hop IP address.

- **dynamicRoutes** : An unordered map that stores dynamic routes in the routing table. It also maps a destination IP address to the next hop IP address.

2. public section:

- **addStaticRoute(const string& destinationIP, const string& nextHopIP)** : A method that adds a static route to the routing table. It takes the destination IP address and the corresponding next hop IP address as parameters and adds them to the **staticRoutes** map.

- **addDynamicRoute(const string& destinationIP, const string& nextHopIP):** A method that adds a dynamic route to the routing table. Similar to the **addStaticRoute** method, it takes the destination IP address and the next hop IP address as parameters and adds them to the **dynamicRoutes** map.

- **getNextHop(const string& destinationIP)** : A method that retrieves the next hop IP address for a given destination IP address. It first checks if the destination IP address exists in the **staticRoutes** map. If found, it returns the corresponding next hop IP address. If not, it checks the **dynamicRoutes** map and returns the next hop IP address if found. If neither static nor dynamic routes exist, it returns an empty string.

- **printRoutingTable()** : A method that prints the contents of the routing table. It displays the destination IP addresses and their corresponding next hop IP addresses for both static and dynamic routes.

EndDevice class that represents an end device in a network. Here's an explanation of the class and its members:

1. private section:

- **device_id** : An integer representing the unique identifier of the end device.
- **device_name** : A string representing the name or label of the end device.
- **Ip_address**: A string representing the IP address assigned to the end device.
- **mac_address**: A string representing the MAC address of the end device.

2. protected section:

- **subnet_mask** : A string representing the subnet mask of the end device.
- **network :** A pointer to a Network object representing the network to which the end device is connected.

3. public section:

- **network** : A public pointer to a Network object representing the network to which the end device is connected.
- **EndDevice(int id, string name, Network* net, string mac, string mask) :** A constructor that initializes the end device with the provided \`id\`, \`name\`, \`mac\` address, and \`mask\`. It also assigns an IP address to the end device by calling the \`assignIPAddress\` method of the \`Network\` object.
- **getDeviceId() :** A method that returns the device ID of the end device.
- **getDeviceName()** : A method that returns the name of the end device.
- **getIpAddress()** : A method that returns the IP address of the end device.
- **getMacAddress() :** A method that returns the MAC address of the end device.
- **getSubnetMask()** : A method that returns the subnet mask of the end device.
- **connect() :** A method that simulates the connection of the end device by printing a message indicating that the device is connected.
- **disconnect() :** A method that simulates the disconnection of the end device by printing a message indicating that the device is disconnected.

Here's an explanation of each class:

1. Hub class:

- It inherits from the **EndDevice class**, representing a network hub.
- It has additional members such as **hub_id**, **hub_name**, **connected_devices**, and **connected_hubs**.
- It provides methods to connect and disconnect devices and hubs, and to retrieve information about connected devices and hubs.

2. Router class:

- It also inherits from the **EndDevice class**, representing a network router.
- It has similar members as the Hub class, including **router_id, router_name, connected_devices**, and **connected_hubs.**
- It provides methods to connect and disconnect devices and hubs.

3. FlowControlProtocol class:

- It is an abstract base class that defines the interface for flow control protocols.
- It declares two pure virtual methods: **canSendPacket()** and **receiveAck(),** which need to be implemented by derived classes.

4. GoBackN class:

- It is a derived class from **FlowControlProtocol**, implementing the Go-Back-N flow control protocol.
- It has members such as **windowSize_, Sf, Sn,** and **timer**.
- It overrides the **canSendPacket()** and **receiveAck()** methods to implement the Go-Back-N protocol behavior.

5. StopNWait class:

- It is another derived class from **FlowControlProtocol**, implementing the Stop-and-Wait flow control protocol.

- It has members such as **expected_seq_num** and timer.

- It overrides the **canSendPacket()** and **receiveAck()** methods to implement the Stop-and-Wait protocol behavior.

6. SelectiveRepeat class:

- It is also a derived class from **FlowControlProtocol**, implementing the Selective Repeat flow control protocol.

- It has members such as **window_size**, received, buffer, Sf, Sn, and timer.

- It overrides the canSendPacket() and receiveAck() methods to implement the Selective Repeat protocol behavior.

- It includes an additional method **getBuffer()** to retrieve the buffer of acknowledged packets.

The provided code includes additional classes related to access control protocols and network devices, specifically bridges and switches.

1. AccessControlProtocol class:

- It is an abstract base class that defines the interface for access control protocols.

- It declares a pure virtual method canSendPacket() which needs to be implemented by derived classes.

2. PureAloha class:

- It is a derived class from AccessControlProtocol, implementing the Pure ALOHA access control protocol.

  - It overrides the canSendPacket() method to generate a random number and check if it is less than the probability of success (`p`).

  - The `p` member represents the probability of success.

3. SlottedAloha class:

  - It is another derived class from AccessControlProtocol , implementing the Slotted ALOHA access control protocol.

  - It overrides the canSendPacket() method to calculate the current time slot, generate a random number, and check if it is less than the probability of success (`p`).

  - Additionally, it ensures that the station is the first to attempt transmission in the current slot by tracking the last slot used (`lastSlot`).

4. Bridge class:

  - It implements both the FlowControlProtocol and AccessControlProtocol interfaces.

  - It has a vector of connected hubs and pointers to the flow control protocol and access control protocol objects.

  - It provides methods to connect and disconnect hubs and overrides the canSendPacket() method from both interfaces.

5. Switch class:

  - It extends the FlowControlProtocol , AccessControlProtocol, and EndDevice classes, representing a network switch.

  - It includes additional members such as switch_id, switch_name, connected_devices, connected_switches, mac_address_table, and pointers to the flow control protocol and access control protocol objects.

  - It provides methods to connect and disconnect devices, resolve MAC addresses based on IP addresses, and overrides the canSendPacket() method from both interfaces.

- The resolveMACAddress() method searches the connected devices for a matching IP address and returns the corresponding MAC address.

The code provided is a C++ program that simulates a network configuration and demonstrates the use of various network devices and protocols. Let's go through the code step by step:

1. The program starts by including necessary headers and defining the main function.

2. It declares and initializes a **FlowControlProtocol** object **(flow_control_protocol)** with the **GoBackN** protocol and an AccessControlProtocol object (**access_control_protocol**) with the **PureAloha** protocol.

3. It creates a **Switch** object (switch_obj) by passing the flow control and access control protocols.

4. It creates a **Network object** (network) with the IP address prefix **"192.168.0".**

5. It creates five **EndDevice** objects (**device1 to device5**) with their respective device IDs, names, network references, MAC addresses, and subnet masks.

6. The **connectDevice** function is called on **switch_obj** to connect each end device to the switch using their respective ports and MAC addresses.

7. The program demonstrates data transmission by checking if the switch can send a packet to **destination_mac** from **device1** and then receiving the acknowledgment.

8. Three different flow control protocols (**GoBackN, StopNWait**, and **SelectiveRepeat**) are instantiated.

9. The canSendPacket function is called on each flow control protocol object to check if a packet can be sent to **destination_mac** from a specific device ID. If successful, an acknowledgment is received.

10. The program prints the number of broadcast and collision domains in the network.

11. Two **Hub** objects (**hub1 and hub2**) are created, each with their respective hub IDs, names, device IDs, device names, IP addresses, and MAC addresses.

12. Ten more **EndDevice** objects (**device6 to device15**) are created.

13. The **connectDevice** function is called on each hub object to connect the end devices to the hubs.

14. The **connectDevice** function is called on **switch_obj** to connect **hub1** and **hub2** to the switch using their respective ports and MAC addresses.

15. Two **Router** objects (**router1 and router2**) are created with their respective router IDs, names, device IDs, device names, IP addresses, and MAC addresses.

16. The **connectDevice** function is called on each router object to connect the end devices to the routers.

17. The **connectHub** function is called on each router object to connect the hubs to the routers.

18. The program retrieves the connected devices and hubs of router1 and router2 using the appropriate getter functions.

19. It then prints the connected devices and hubs of router1 and router2.

20. The program demonstrates disconnection by calling the **disconnectDevice** and **disconnectHub** functions on router1 and router2, respectively, to disconnect specific devices and hubs.

21. It prints the updated connected devices and hubs of router1 and router2.

22. The program performs data transmission between all end devices using the switch and the flow control protocols.

23. It again prints the number of broadcast and collision domains in the network.

24. Finally, the program returns 0, indicating successful execution.