Network simulator implementing entire TCP/IP protocol stack

Shivam Singh Khatri (2020BITE064)
Kundan Kumar (2020BITE069)
Kasi Sunil (2020BITE045)

1 Introduction

This document provides documentation for the implementation of a TCP/IP network using Python. The code below demonstrates the structure and functionality of the implemented network components, including routers, hubs, devices, switches, ports, and bridges.

2 Code Explanation

2.1 Language and Library

The code is implemented in Python programming language and utilizes the following library:

- `time`: The code imports the `time` module to utilize functions related to time-based operations, such as introducing delays using the `sleep()` function.

2.2 Router

The `Router` class represents a router object and includes the following functions:

- `__init__()`: Initializes the router object and sets up an empty routing table.
- `addRoute(network, interface)`: Adds a network and its associated interface to the routing table.
- `findInterface(ipAddress)`: Finds the appropriate interface for a given IP address based on the routing table.

2.3 Hub

The `Hub` class represents a hub object and includes the following functions:

- `__init__()`: Initializes the hub object and sets up empty lists for connected devices, switches, bridges, and the availability of a token.
- `connectDevice(device)`: Connects a device to the hub and establishes a bidirectional connection.
- `connectSwitch(switch)`: Connects a switch to the hub and establishes a bidirectional connection.
- `connectBridge(bridge)`: Connects a bridge to the hub and establishes a bidirectional connection.
- `broadcastMessage(sender, receiver, message)`: Broadcasts a message to all connected devices, relays it through switches if necessary, and relays it through bridges if both sender and receiver are connected to the same bridge.
- `passToken()`: Passes the token to the next device in the network.
- `addRoute(network, interface)`: Adds a network and its associated interface to the hub's internal router.
- `findInterface(ipAddress)`: Finds the appropriate interface for a given IP address using the hub's internal router.

2.4 Device

The `Device` class represents a device object and includes the following functions:

- `__init__()`: Initializes the device object with a unique ID, IP address, and a reference to its connected hub.
- `sendMessage(receiver, message, flow_control)`: Sends a message to the specified receiver device using the chosen flow control protocol (Stop-and-Wait or Sliding Window).
- `sendStopAndWait(receiver, message)`: Sends a message to the receiver device using the Stop-and-Wait flow control protocol.
- `sendSlidingWindow(receiver, message)`: Sends a message to the receiver device using the Sliding Window flow control protocol.
- `receiveMessage(sender, message)`: Receives a message from the specified sender device.

2.5 Switch

The `Switch` class represents a switch object and includes the following functions:

- `__init__()`: Initializes the switch object and sets up empty lists for connected devices, switches, and the availability of a token.
- `connectDevice(device)`: Connects a device to the switch and establishes a bidirectional connection.
- `connectSwitch(switch)`: Connects a switch to the switch and establishes a bidirectional connection.
- `broadcastMessage(sender, receiver, message)`: Broadcasts a message to all connected devices and relays it through switches if necessary.
- `passToken()`: Passes the token to the next device in the network.
- `addRoute(network, interface)`: Adds a network and its associated interface to the switch's internal router.
- `findInterface(ipAddress)`: Finds the appropriate interface for a given IP address using the switch's internal router.

2.6 Port

The `Port` class represents a port object and includes the following functions:

- `__init__()`: Initializes the port object with a unique ID, IP address, and a reference to its connected device or switch.
- `sendMessage(receiver, message)`: Sends a message to the specified receiver device or switch.
- `receiveMessage(sender, message)`: Receives a message from the specified sender device or switch.

2.7 Bridge

The `Bridge` class represents a bridge object and includes the following functions:

- `__init__()`: Initializes the bridge object and sets up empty lists for connected devices, switches, and the availability of a token.
- `connectDevice(device)`: Connects a device to the bridge and establishes a bidirectional connection.
- `connectSwitch(switch)`: Connects a switch to the bridge and establishes a bidirectional connection.
- `broadcastMessage(sender, receiver, message)`: Broadcasts a message to all connected devices and relays it through switches if necessary.

- `passToken()`: Passes the token to the next device in the network.
- `addRoute(network, interface)`: Adds a network and its associated interface to the bridge's internal router.
- `findInterface(ipAddress)`: Finds the appropriate interface for a given IP address using the bridge's internal router.

3 Conclusion

The provided code demonstrates a TCP/IP implementation using various classes to represent network components. By utilizing routers, hubs, devices, switches, ports, and bridges, the code allows for message transmission and relaying within a network. The implementation supports both stop-and-wait and sliding window flow control protocols. Overall, this implementation provides a basic framework for understanding TCP/IP functionality in a simplified network environment.