CY350 - Computer Networks Project

Milestone C – Web Client & Server with Routing and Forwarding

In Milestone C, you will implement routers to sit between the client and server, simulating network-layer functionality. The router will replicate how real-world routers forward packets based on routing tables, handling multiple paths between nodes. By the end of this milestone, you should be comfortable with the following concepts:

- Understanding how routers forward packets in a network.
- Implementing and managing routing tables.
- Simulating packet forwarding between multiple nodes.
- Further developing your Python classes and methods.

A diagram of the setup for this milestone is on the next page.

General Information

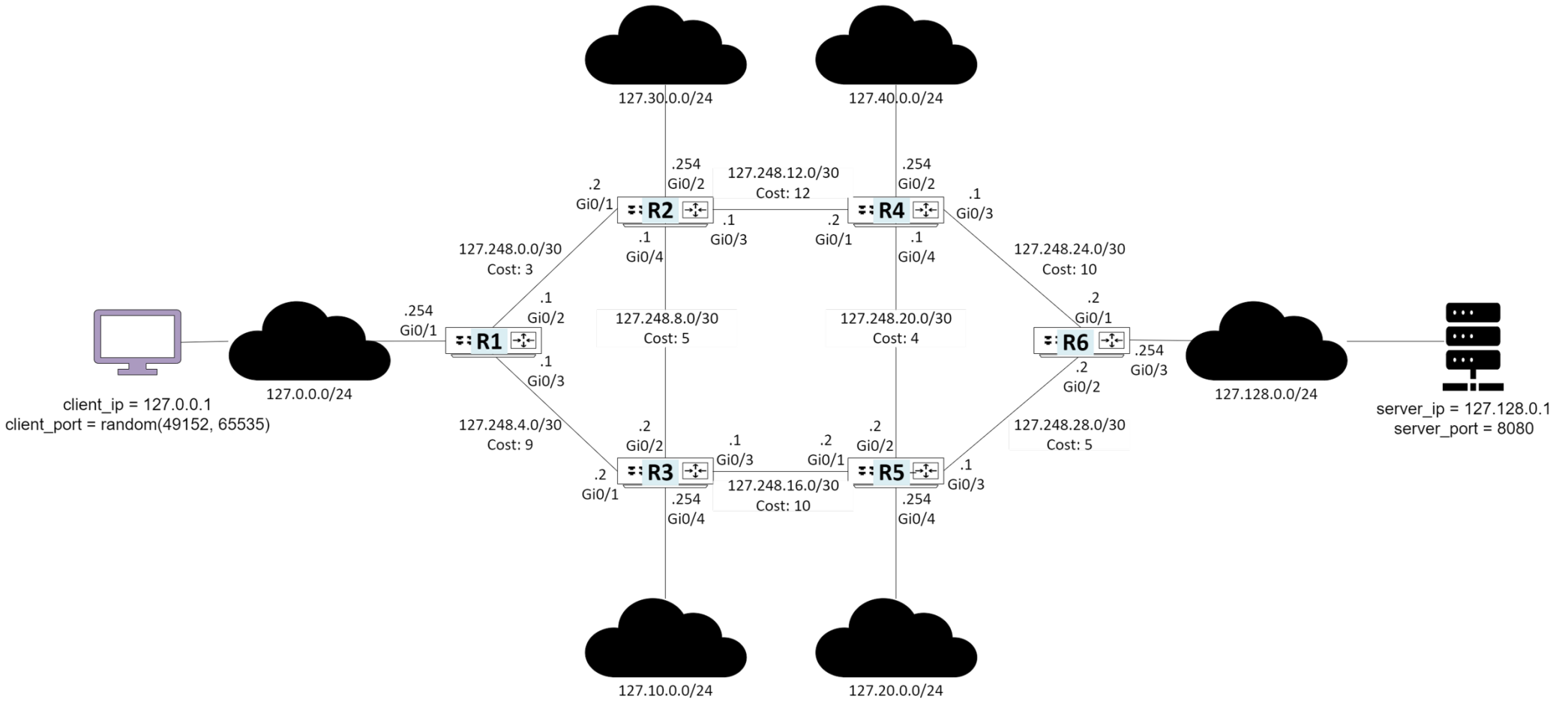
As a reminder, we will manually grade any project that does not get full credit from the autograder. However, we reserve the right to factor in your effort when determining the points that are assigned. By the end of this milestone, you should have a fully functional router that can route packets between the client and server. This lays the foundation for Milestone D, where we will introduce switches to simulate link-layer services and expand upon the previously implemented portions of the network application.

Project Points (100 total)

- Code Structure Questions in Canvas (25 pts)
- Router class implementation – submitted to Gradescope (60 pts)
- Wireshark packet capture – submitted in Canvas (15 pts)

Code: You must write your solution in Python 3. We highly recommend that you use the programming environment configured in your EECSNet Virtual Machine (VM) for the course. Since the milestone requires raw sockets, the code is best run on Linux and must be run using administrative privileges.

The starter file (*milestone_c_starter_files.zip*) includes a full implementation of the web application and network in the diagram below, with the exception of the methods you must implement in the Router class. You are encouraged to review all of the code to understand how the client/server network communication has been implemented. Questions #1-5 of the project assignment in Canvas assess your understanding of the code structure.


127.30.0.0/24
127.40.0.0/24
.254
Gi0/2
127.248.12.0/30
Cost: 12
.2
Gi0/1
R2
.1
Gi0/3
.2
Gi0/1
R4
.1
Gi0/3
.1
Gi0/4
.1
Gi0/4
127.248.0.0/30
Cost: 3
127.248.24.0/30
Cost: 10
.1
Gi0/2
.2
Gi0/1
.254
Gi0/1
R1
R6
.254
Gi0/3
127.248.8.0/30
Cost: 5
127.248.20.0/30
Cost: 4
.1
Gi0/3
.2
Gi0/2
client_ip = 127.0.0.1
client_port = random(49152, 65535)
127.0.0.0/24
127.128.0.0/24
server_ip = 127.128.0.1
server_port = 8080
127.248.4.0/30
Cost: 9
.2
Gi0/2
.1
Gi0/3
.2
Gi0/1
.2
Gi0/2
127.248.28.0/30
Cost: 5
.2
Gi0/1
R3
R5
.1
Gi0/3
127.248.16.0/30
Cost: 10
.254
Gi0/4
.254
Gi0/4
127.10.0.0/24
127.20.0.0/24

A summary of the files provided:

- pdu.py
  - IPHeader class: Handles the basic functionality of an IP header.
  - LSADatagram class: Inherits from IPHeader and adds attributes and methods for handling Link-State Advertisements (LSAs).
  - HTTPDatagram Class: Inherits from IPHeader and adds TCP-like attributes for handling HTTP-like packets in custom datagram formats.

- tcp_client.py
  - Client class: represents a custom HTTP-like client that communicates with a server using raw sockets, mimicking the TCP/IP stack for connection establishment, data transmission, and response handling. It operates using the Go-Back-N protocol for reliable data transmission.
- graph.py
  - Graph class: represents a directed, weighted graph used for network routing algorithms. The graph consists of nodes (routers or hosts) and edges (connections) between them with associated costs (e.g., latency, distance) and network interfaces.
    - Note: The Graph class supports the implementation of Djikstra's algorithm in the Router class.
- router.py
  - Router class: designed to simulate the basic functionality of a network router, including processing Link-State Advertisements (LSAs), running a routing algorithm, forwarding packets based on a forwarding table, and managing network interfaces.
    - Note: You are required to implement the portions of the Router class as described below.
- resources.json
  - represents a collection of web pages and related metadata for a web server. Each entry corresponds to a webpage or resource that the server can serve to a client.
- tcp_server.py
  - Server class: simulates a basic server using raw sockets and demonstrates core concepts such as connection establishment (three-way handshake), reliable data transmission (Go-Back-N protocol), and HTTP request/response processing.
- network_app.py
  - NetworkApp class: simulates a simple network of routers, a client, and a server (as shown in the diagram above), demonstrating core networking principles such as link-state routing, client-server communication, and concurrent processing of network data.
    - Note: Once you have finalized the implementation of the Router class, you can run this script to simulate the operation of the network, including

running routing protocols among the routers and forwarding an HTTP request and response over the network between the client and server.

Building the Router:

For this milestone, you will complete the implementation of a **Router** class that forwards packets between the client and the server. You will need to simulate a routing protocol, ensuring that packets are forwarded based on the destination IP address and the appropriate next hop. Upon completion, the Router will be capable of:

- Initializing the Link-State Database (LSDB) of the router based on the direct connections.

  Hint: Review the code documentation to understand the data structure for storing the LSDB on the router.

- Processing a Link-State Advertisement (LSA) to either:
  - Update the LSDB with the information provided and forward the LSA to all other interfaces except the one on which it was received.
  - Or, discard the LSA since it has already been seen and processed by the router.

  Hints:

  - The LSA has a sequence number that can be stored locally and compared with that of future LSAs.
  - Review the code documentation to understand the format for LSA data in packet.
- Running a link-state routing protocol using Djikstra's algorithm once the LSDB is complete.

  Hints:
  - Refer to Lesson 17 to review Djikstra's algorithm.
  - Review the code documentation to understand the data structure for storing the forwarding table on the router.
- Forwarding TCP segments at each router between the client and server.
  - Must perform longest prefix matching between the destination IP address and networks in the forwarding table.

The class diagram below describes the design of the code for the Router class. Methods you are required to implement are highlighted in bold.

Router	
Data Type	*Attribute Name*
str	router_id
dict	router_interfaces
dict	direct_connections
int	self.lsa_seq_num
dict	interface_sockets
dict	router_lsa_num
dict	lsdb
time	lsa_timer
dict	forwarding_table
Method Name	*Purpose*
initialize_lsdb()	Initialize the LSDB with the router's direct connections.
update_lsdb()	Updates the LSDB with new information from a received LSA.
send_initial_lsa()	Broadcast the initial LSA containing the router's direct connections.
forward_lsa()	Forwards a received LSA to all interfaces except the one on which it was received.
process_link_state_advertisement()	Processes a received Link-State Advertisement (LSA) and updates the LSDB. If the LSA contains new information, the router broadcasts the LSA to its other interfaces.
forward_datagram()	Forwards an HTTP datagram to the appropriate next hop based on the forwarding table.
run_route_alg()	Runs Dijkstra's shortest path algorithm to calculate the shortest paths to all nodes in the network based on the LSDB and updates the forwarding table accordingly.
process_datagrams()	Receives, processes, and forwards incoming datagrams or LSAs. Updates the LSDB and forwarding table as needed, and then forwards datagrams to their correct next hop.
shutdown()	Shuts down the router by closing all open sockets.

Data Structures for the Router class

- The Router class is provided information about the interfaces and direct connections at initialization, akin to configuring a router when it is installed.

  Example router interfaces (R1):

  {

  'Gi0/1': ('127.0.0.254', '127.0.0.1'),

  'Gi0/2': ('127.248.0.1', '127.248.0.2'),

  'Gi0/3': ('127.248.4.1', '127.248.4.2')

  }

  Example router direct connections (R1):

  {

  '127.0.0.0/24': (0, 'Gi0/1'),

  '2.2.2.2': (3, 'Gi0/2'),

  '3.3.3.3': (9, 'Gi0/3')

  }

- The LSDB is a dictionary of the link-states known to the router in the form {advertising_router: [(network/router_id, cost, interface), …]}.

  Example LSDB for R1:

  {

  '1.1.1.1': [('127.0.0.0/24', 0, 'Gi0/1'), ('2.2.2.2', 3, 'Gi0/2'), ('3.3.3.3', 9, 'Gi0/3')],

  '2.2.2.2': [('127.30.0.0/24', 0, 'Gi0/2'), ('1.1.1.1', 3, 'Gi0/1'), ('3.3.3.3', 5, 'Gi0/4'), ('4.4.4.4', 12, 'Gi0/3')],

  '3.3.3.3': [('127.10.0.0/24', 0, 'Gi0/4'), ('1.1.1.1', 9, 'Gi0/1'), ('2.2.2.2', 5, 'Gi0/2'), ('5.5.5.5', 10, 'Gi0/3')],

  '4.4.4.4': [('127.40.0.0/24', 0, 'Gi0/2'), ('2.2.2.2', 12, 'Gi0/1'), ('5.5.5.5', 4, 'Gi0/4'), ('6.6.6.6', 10, 'Gi0/3')],

  '5.5.5.5': [('127.20.0.0/24', 0, 'Gi0/4'), ('3.3.3.3', 10, 'Gi0/1'), ('4.4.4.4', 4, 'Gi0/2'), ('6.6.6.6', 5, 'Gi0/3')],

  '6.6.6.6': [('127.128.0.0/24', 0, 'Gi0/3'), ('4.4.4.4', 10, 'Gi0/1'), ('5.5.5.5', 5, 'Gi0/2')]

  }

- The forwarding table is a dictionary of the best paths to all known nodes (networks or routers) in the form {network/router_id: (interface, cost)}.

Example Forwarding Table for R1:

{

'1.1.1.1': (None, 0),

'127.0.0.0/24': ('Gi0/1', 0),

'2.2.2.2': ('Gi0/2', 3),

'3.3.3.3': ('Gi0/2', 8),

'127.30.0.0/24': ('Gi0/2', 3),

'4.4.4.4': ('Gi0/2', 15),

'127.10.0.0/24': ('Gi0/2', 8),

'5.5.5.5': ('Gi0/2', 18),

'127.40.0.0/24': ('Gi0/2', 15),

'6.6.6.6': ('Gi0/2', 23),

'127.20.0.0/24': ('Gi0/2', 18),

'127.128.0.0/24': ('Gi0/2', 23)

}

Wireshark

In addition to the coding aspect of this milestone, you will also need to demonstrate that you have your code working locally and can capture traffic sent across the loopback. In Canvas, you need to submit two screenshots of your Wireshark capture (from your CY350 EECSNet VM):

- One that contains the exchange of LSAs between routers.
- One that contains the HTTP GET request and HTTP response as they traverse the network between the client and server.

Submission Requirements:

- In Canvas:
    - Complete the Code Structure Questions for the Project Milestone C Assignment.
    - Submit your Wireshark screenshots as .jpg files.
    - Submit a PDF acknowledgement statement, as required in previous milestones.
- In Gradescope:
    - Submit the router.py file with full implementation of the Router class.