CS305 2025 Spring Programming Assignment 2

A Reliable P2P File Transfer Protocol and DV Routing Algorithm

Deadline: Saturday, May 24th, 2025, 23:55:00

1. Introduction

In this assignment, you will implement a peer-to-peer (P2P) file transfer system by developing two components:

- **Task 1:** A **simplified Reliable Data Transfer (RDT) protocol** in the application layer using UDP.
- **Task 2:** A **Distance Vector (DV)** routing algorithm in the application layer.

While RDT and DV are conventionally associated with the transport and network layers, respectively, you will implement them at the application layer to simulate their behavior.

UDP is a simple connectionless protocol that can transmit messages between hosts without prior communication to set up communication channels. However, UDP segments may be lost during transmission or delivered out of order, which can degrade application performance. Thus, it is interesting to develop a reliable transfer protocol in the application layer based on UDP to improve file transmission reliability.

Furthermore, in the network layer, a routing algorithm is necessary for message transmission between peers, especially those communicating through multiple hops. A routing algorithm can provide a path to transmit messages between a source and a destination. In this assignment, you will develop a DV-based routing algorithm.

The details of the above two tasks are as follows:

Task 1: A reliable transfer protocol in the application layer based on UDP

- You will implement file segmentation, acknowledgments (ACKs), retransmissions, and reassembly in a network environment where packet loss and corruption may occur.

Task 2: Distance Vector (DV)-based routing algorithm

- Build DV routing with Bellman-Ford updates.
- Dynamically change the link costs between peers and update their DVs.
- Check TTLs (Time to Live) during packet forwarding.

2. What you need to do

In the following figures and tables, the tasks and functions to be completed are introduced in detail:

Task 1: A reliable file transfer protocol in the application layer based on UDP

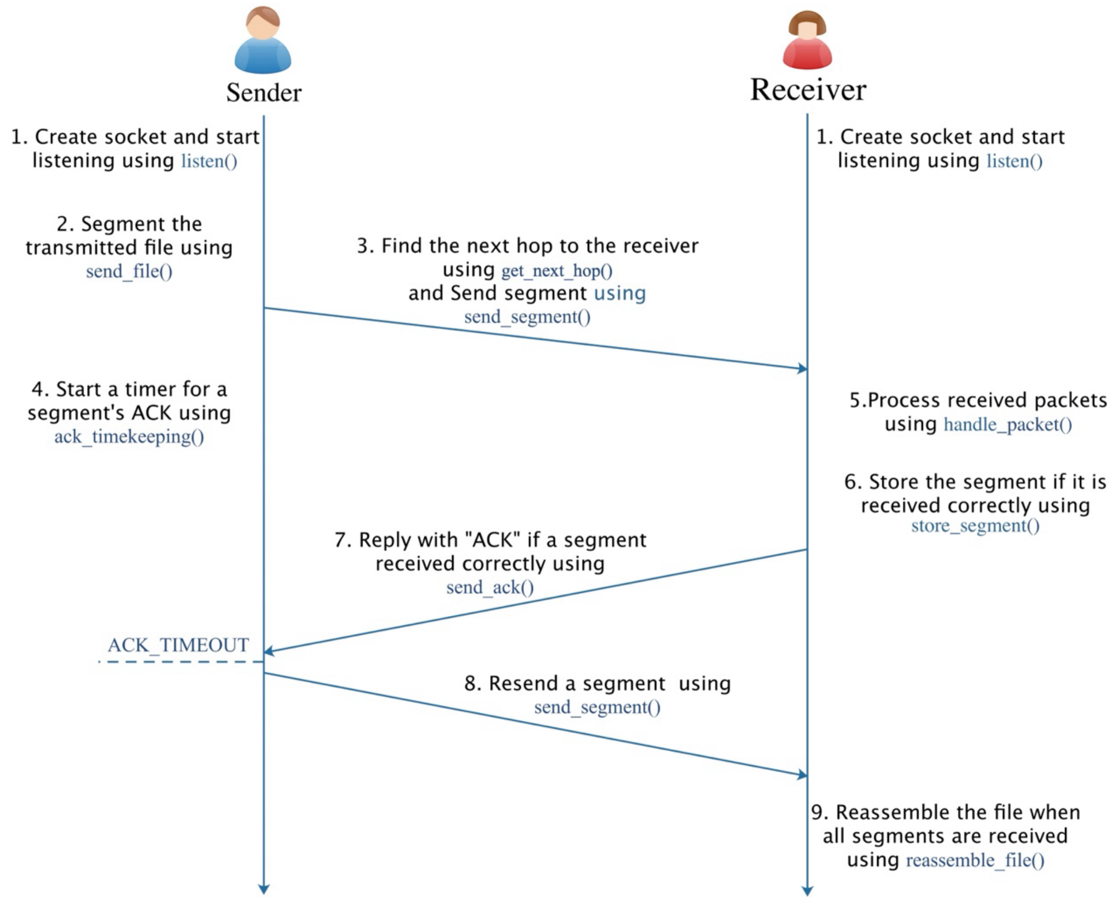
Sender
Receiver
1. Create socket and start listening using listen()
1. Create socket and start listening using listen()
2. Segment the transmitted file using send_file()
3. Find the next hop to the receiver using get_next_hop() and Send segment using send_segment()
4. Start a timer for a segment's ACK using ack_timekeeping()
5.Process received packets using handle_packet()
6. Store the segment if it is received correctly using store_segment()
7. Reply with "ACK" if a segment received correctly using send_ack()
ACK_TIMEOUT
8. Resend a segment using send_segment()
9. Reassemble the file when all segments are received using reassemble_file()

Figure 1: File Sharing Logic

Task 1.1: Listener and Packet Receiver

Function	Explanation
listen	Create a UDP socket, bind it, and listen to incoming packets.
handle_packet	(Notice: This function provides the code for simulating packet drops and corruption.) Each peer processes received packets as follows: 1. Validate the received packets with checksum. 2. Drop the packets if the TTL is reached. 3. If the packet type is DATA and peer_id is the destination, then record the payload and reply to the source with ‘ACK’. 4. If the packet type is ACK and peer_id is the destination, then record the ACK for corresponding segments. 5. If the packet type is DV, run the handle_dv_update function to update the local DV. 6. If the packet is received correctly and peer_id is not the destination, then forward the packet to the next hop or drop the packet if the TTL is reached.

Task 1.2: File Transmission

Function	Explanation
send_file	1. Segment the transmitted file with the given SEGMENT_SIZE. 2. Use a sliding window with the given WINDOW_SIZE for sending segments. 3. Print out the information that all segments have been sent successfully.
send_segment	Make a packet for a segment, and send it to the next hop based on the local DV.

Task 1.3: Segment Retransmission

Function	Explanation
ack_timekeeping	Create a timer to calculate the time it takes to receive a segment's "ACK". Resend the segment if no "ACK" is received before ACK_TIMEOUT. The times to resend a segment should not exceed the given MAX_RETRIES.

Task 1.4: Segment Reception, Reassembly, and ACK

Function	Explanation
send_ack	The receiver makes a packet for an ACK and sends it to the sender if a segment is received correctly.
store_segment	The receiver stores the segment if it is received correctly.
reassemble_file	The receiver combines and writes all segments to a file, namely "received_file.txt", and stores it in its folder.

TASK 2: Distance Vector (DV)-based routing algorithm

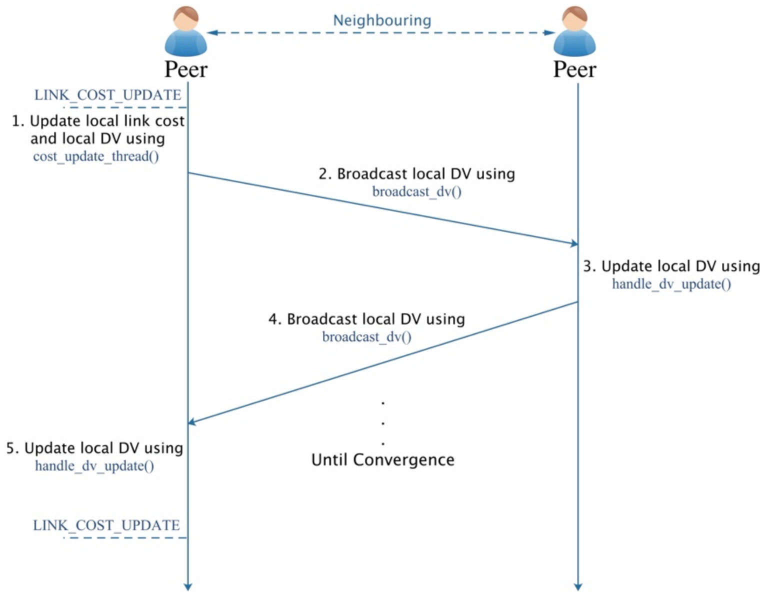
Neighbouring
Peer
Peer
LINK_COST_UPDATE
1. Update local link cost and local DV using cost_update_thread()
2. Broadcast local DV using broadcast_dv()
3. Update local DV using handle_dv_update()
4. Broadcast local DV using broadcast_dv()
5. Update local DV using handle_dv_update()
Until Convergence
LINK_COST_UPDATE

Figure 2: Dynamic DV Logic

Function	Explanation
broadcast_dv	Make a packet for the local DV and broadcast it to neighbours if the local DV has changed.
handle_dv_update	Apply the Bellman-Ford equation to update the local DV after receiving DV estimates from neighbours. Then, use the function broadcast_dv to broadcast the updated DV to neighbours.
cost_update_thread	Update the link cost with neighbours (plus/minus one or no change) and recompute the local DV. Then, use the function broadcast_dv to broadcast the updated DV to neighbours. The update period is given by COST_UPDATE_INTERVAL.
get_next_hop	Find the next hop to a given destination based on the local DV.
route_print	Print out the route from a peer to other peers.

3. Provided codes

To reduce the complexity of making typical UDP segments and IP datagrams, this assignment uses a hybrid simplified format of packets transmitted between peers, which is a mix of UDP segments and IP datagrams.

Specifically, the header is a JSON dictionary containing the following fields:

- Type: Packet type such as DATA, ACK, and DV (DV updates).
- Seq: Sequence number of the segment.
- Total: The total number of segments.
- Src: Source of the packet, which can be represented by peer_id.
- Dst: The destination of the packet, which can be represented by peer_id.
- TTL: Time to Live or the maximum number of hops.
- Checksum: Header checksum.

The payload is raw binary data (such as segments or DV updates). Therefore, the full packet format is **[4-byte header length][JSON header][binary payload]**.

To avoid errors and ensure consistency, two helper functions have been provided:

- make_packet(...): Constructs a packet from its components.
- parse_packet(...): Extracts the header and payload from received data.

These functions are already implemented in the starter code and should not be modified. Instead, focus your efforts on routing, segment handling, reliability, and protocol logic. The starter file includes structure, packet helpers, and setup logic (i.e., Main function). **The parameter values in the "Constants" part of the starter code should not be modified**.

We also have some starter files:

- **Config.json:** This file specifies each peer's IP address and port (**which should**

not be changed) and the initial link cost of peers. Kindly remind that if the IP addresses are not explicitly available in the system, the following command may help solve your problem: *sudo ifconfig lo0 alias 127.0.0.2 up*.

- **Input.txt:** This file is to be transmitted between peers. We will use it to test your code.

4. How to test your code

Start each peer in a separate terminal:

```
python3 peer.py --id Peer1
python3 peer.py --id Peer2
python3 peer.py --id Peer3
python3 peer.py --id Peer4
python3 peer.py --id Peer5
```

Then, in one terminal:

```
send Peer5 input.txt
```

Use 'routes' to print your routing table and 'check' to inspect reassembly.

5. Evaluation Criteria

#	Task	Points
1	**Listener and Packet Receiver**	20
2	**File Transmission**	10
3	**Segment Retransmission**	10
4	**Segment Reception, Reassembly, and ACK**	10
5	**Distance Vector Routing**	40
6	**Report (Print logs for events; write a summary report with screenshots)**	10

6. Submission

Please package all your files (including your report) into a zip file named "{Your student ID number}_{Your name}" and submit it on Blackboard.