Akshat Patil

AIDS A

Roll: 10

CN assignment 6

Tasks: i) Go back N Sliding Window Protocol in peer to peer mode

ii) Selective Repeat Sliding Window Protocol in peer to peer mode

1)

*import* random

*import* time

class GoBackN:

def \_\_init\_\_(*self*, *window\_size*, *total\_frames*):

*self*.window\_size = *window\_size* *# Size of the sending window*

*self*.total\_frames = *total\_frames* *# Total number of frames to send*

*self*.frames = list(range(1, *total\_frames* + 1)) *# Frames numbered from 1 to N*

*self*.base = 0 *# Start of the window*

*self*.next\_seq = 0 *# Next frame to be sent*

def send\_frames(*self*):

*while* *self*.base < *self*.total\_frames:

print(f"\n[INFO] Sending frames in window: {*self*.frames[*self*.base:*self*.base + *self*.window\_size]}")

*# Transmit frames within the window*

*for* i *in* range(*self*.base, min(*self*.base + *self*.window\_size, *self*.total\_frames)):

*if* *self*.next\_seq < *self*.total\_frames:

print(f"Sent Frame {*self*.frames[*self*.next\_seq]}")

*self*.next\_seq += 1

time.sleep(0.5) *# Simulate transmission delay*

*# Simulating ACK reception with some errors*

*self*.receive\_ack()

def receive\_ack(*self*):

*# Simulate random frame loss or corruption*

*if* random.random() < 0.2: *# 20% chance of error*

error\_frame = random.randint(*self*.base, min(*self*.base + *self*.window\_size - 1, *self*.total\_frames - 1))

print(f"\n[ERROR] Frame {*self*.frames[error\_frame]} lost! Retransmitting from Frame {*self*.frames[error\_frame]}")

*self*.next\_seq = error\_frame *# Go back to the lost frame*

*else*:

print("\n[INFO] All frames acknowledged successfully!")

*self*.base += *self*.window\_size *# Move the window forward*

WINDOW\_SIZE = 4 *# Sliding window size*

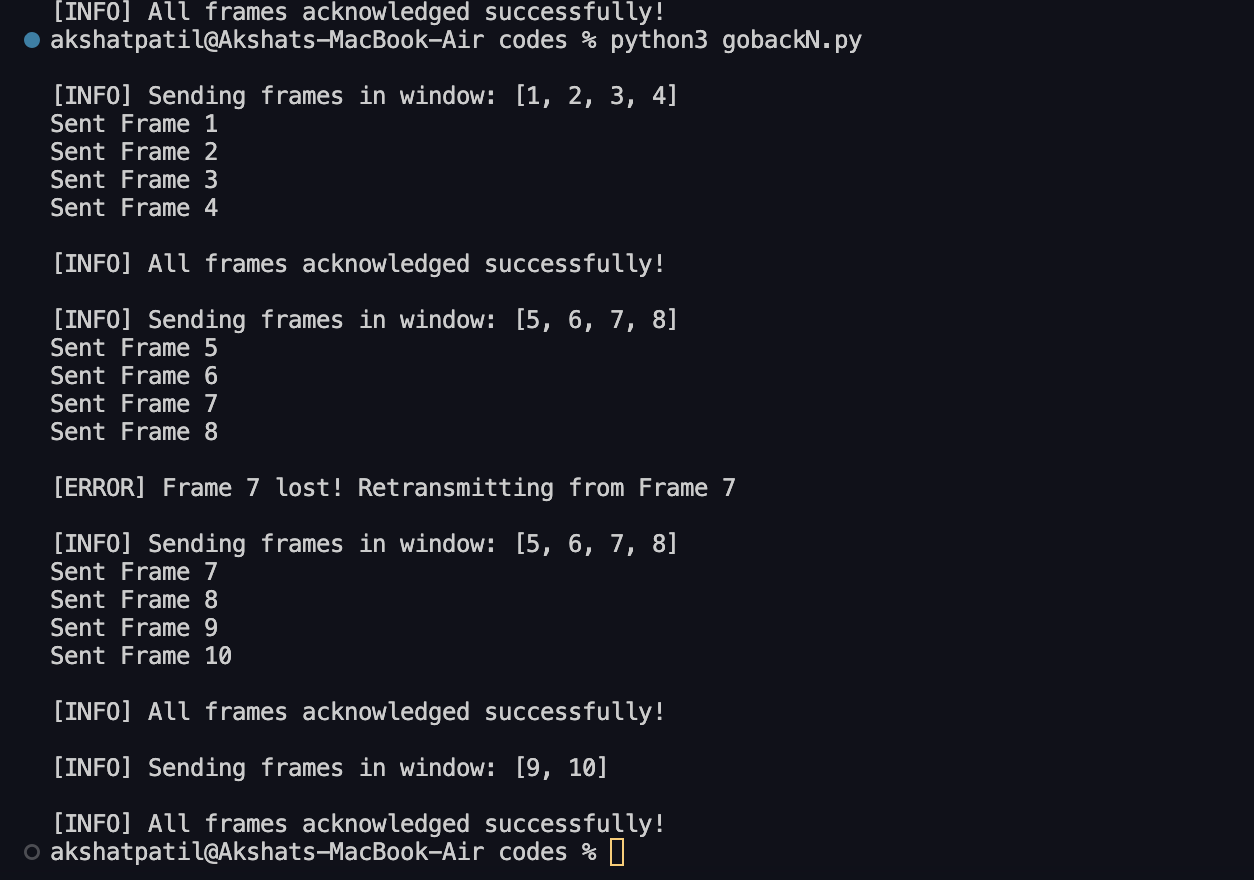
TOTAL\_FRAMES = 10 *# Number of frames to send*

*# Run the Go-Back-N simulation*

gbn = GoBackN(WINDOW\_SIZE, TOTAL\_FRAMES)

gbn.send\_frames()

**Output:**

****

**2)**

***import* random**

***import* time**

**class SelectiveRepeat:**

**def \_\_init\_\_(*self*, *window\_size*, *total\_frames*):**

***self*.window\_size = *window\_size* *# Max frames that can be sent at a time***

***self*.total\_frames = *total\_frames* *# Total frames to send***

***self*.frames = list(range(1, *total\_frames* + 1)) *# Frame sequence numbers***

***self*.acked = [False] \* *total\_frames* *# Tracks received acknowledgments***

**def send\_frames(*self*):**

**print("\n--- Start Transmission ---")**

**i = 0 *# Frame index***

***while* i < *self*.total\_frames:**

**window\_end = min(i + *self*.window\_size, *self*.total\_frames)**

**print(f"\n[INFO] Sending frames: {*self*.frames[i:window\_end]}")**

***for* j *in* range(i, window\_end):**

***if* not *self*.acked[j]: *# Only send if not acknowledged***

**print(f"Sent Frame {*self*.frames[j]}")**

**time.sleep(0.3) *# Simulating transmission delay***

***self*.receive\_ack(i, window\_end)**

***# Move the window forward***

***while* i < *self*.total\_frames and *self*.acked[i]:**

**i += 1**

**print("\n✅ All frames successfully transmitted!")**

**def receive\_ack(*self*, *start*, *end*):**

***# Simulate errors***

**lost\_frames = random.sample(range(*start*, *end*), random.randint(0, 2)) *# Randomly lose 0-2 frames***

***for* j *in* range(*start*, *end*):**

***if* j in lost\_frames:**

**print(f"\n[ERROR] Frame {*self*.frames[j]} lost! Requesting retransmission...")**

***else*:**

**print(f"[ACK] Frame {*self*.frames[j]} received successfully!")**

***self*.acked[j] = True *# Mark frame as acknowledged***

***if* \_\_name\_\_ == "\_\_main\_\_":**

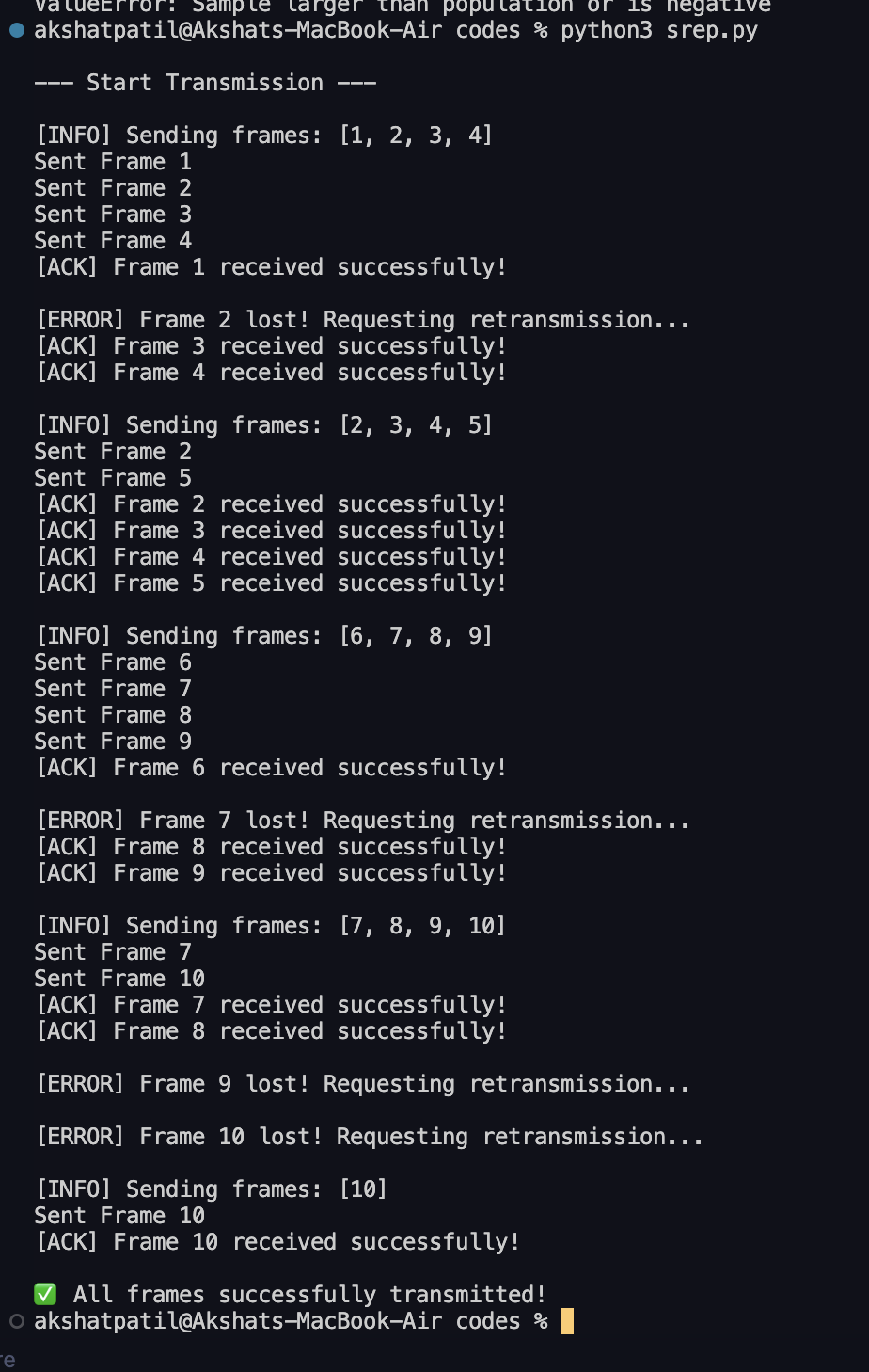
**window\_size = 4**

**total\_frames = 10**

**sr = SelectiveRepeat(window\_size, total\_frames)**

**sr.send\_frames()**

**Output:**

****