PHASE 4 REPORT

EECE 4830 Network Design – Go-Back-N (GBN) Over Unreliable UDP

Team Members: Jacob Alicea, Josiah Concepcion ,Jamie Oliphant, Tim Saari

Course: EECE 4830/5830 (Spring 2025)

Instructor: Dr. Vinod Vokkarane

Overview

This report details the design, implementation, and testing of our Go-Back-N (GBN) protocol over an unreliable UDP channel. This builds on Phase 3 (RDT 3.0) by adding pipelined data transfer with a fixed window size (N), allowing multiple unacknowledged packets to be "in flight."

We have extended our earlier work (Phase 2 and 3) that handled bit errors and packet loss. Now in Phase 4, we adapt our protocol to incorporate the following features:

1. Window-based pipelining with sequence numbers.
2. Checksum or simple bit-flip error simulation.
3. Go-Back-N retransmissions on timeout or error detection.
4. Configurable timeout intervals and window sizes to explore performance trade-offs.

We run tests over five scenarios (similar to Phase 3) but in a GBN context, and also produce required performance graphs as specified by the Phase 4 assignment (0%–70% loss/error in increments, as well as varying timeout and window size).

Design Details

GBN Protocol Enhancements

- Window Size (N)
  We maintain a send window of size N (e.g., 10). This means the sender may send up to N packets without waiting for an ACK of the first packet.

- Sequence Numbers
  Each packet has a sequence number modulo some maximum (commonly 2^k), but to keep it simple, we may use a large integer range or a direct rolling integer. Only packets that fall within the "send window" will be transmitted.

- Cumulative ACKs
  In Go-Back-N, the receiver sends back cumulative ACKs indicating that all packets up to some sequence number were received successfully. If the sender times out (no ACK for a leading packet), the sender goes back and retransmits from that packet onward.

- Timers
  We use a single countdown timer (e.g., 50 ms) associated with the oldest unACKed packet. If no ACK arrives for that packet before the timer expires, all unACKed packets in the window are retransmitted.

Packet Structure

- Header
  - 4-byte sequence number (packed using struct.pack('!I', seq_num) in Python).
  - Possibly 2 or 4 bytes for checksum (or you can keep the simpler bit-flip simulation).
- Data
  - Up to 1024 bytes (or your chosen size).

Error and Loss Simulation

We continue using the same approach from Phases 2 and 3—flipping bits or randomly dropping packets/ACKs. The key difference is that the sender now manages multiple in-flight packets.

- No Loss/Bit-Errors
    - error_rate = 0.0, loss_rate = 0.0
- ACK Packet Bit-Error
    - The sender flips bits in ACK packets with error_rate > 0.
- Data Packet Bit-Error
    - The receiver flips bits in data packets with error_rate > 0.
- ACK Packet Loss
    - The sender drops ACK packets with loss_rate > 0.
- Data Packet Loss
    - The receiver drops data packets with loss_rate > 0.

Sender and Receiver Logic

Sender (sender.py)

- Window Management
    - Maintain a sliding window of size N.
    - Send up to N packets while advancing base and next_seq_num.
- Countdown Timer
    - Start/reset the timer whenever a packet is first sent.
    - On timeout, Go-Back-N: retransmit all packets from base to next_seq_num-1.

- ACK Handling
    - On receiving an ACK for seq_num, move base forward.
    - If base equals next_seq_num, stop the timer (no unACKed packets remain). Otherwise, restart it.
- Error/Loss Simulation
    - For the ACK side, we intentionally drop or flip bits.

Receiver (receiver.py)

- Receiving Packets
    - Check packet correctness (bit errors) and simulate data loss if configured.
- Cumulative ACKs
    - If a packet's sequence number = expected_seq_num, deliver data to the file and increment expected_seq_num.
    - Send an ACK for the highest contiguous sequence number received.
- Dropped or Corrupted Packets
    - If a packet is out of order or corrupted, do not advance expected_seq_num. (Standard GBN approach is to discard out-of-order packets and keep expected_seq_num the same.)

How To Run the Programs

1. File Setup
    - Place sender.py, receiver.py, and your chosen file (e.g., myImage.bmp or tiger.jpg) into the same directory.
2. Receiver

In one terminal, run:
bash
CopyEdit
python receiver.py

The receiver binds to UDP port 5001, waits for incoming data, and writes the output to received_myImage.bmp (or received_tiger.jpg).

3. Sender

In a second terminal, run:
bash
CopyEdit
python sender.py

The sender iterates through five test scenarios (0%–70% error/loss in increments of 5% or 10%) for GBN. You may also vary timeout and window size for additional performance results.

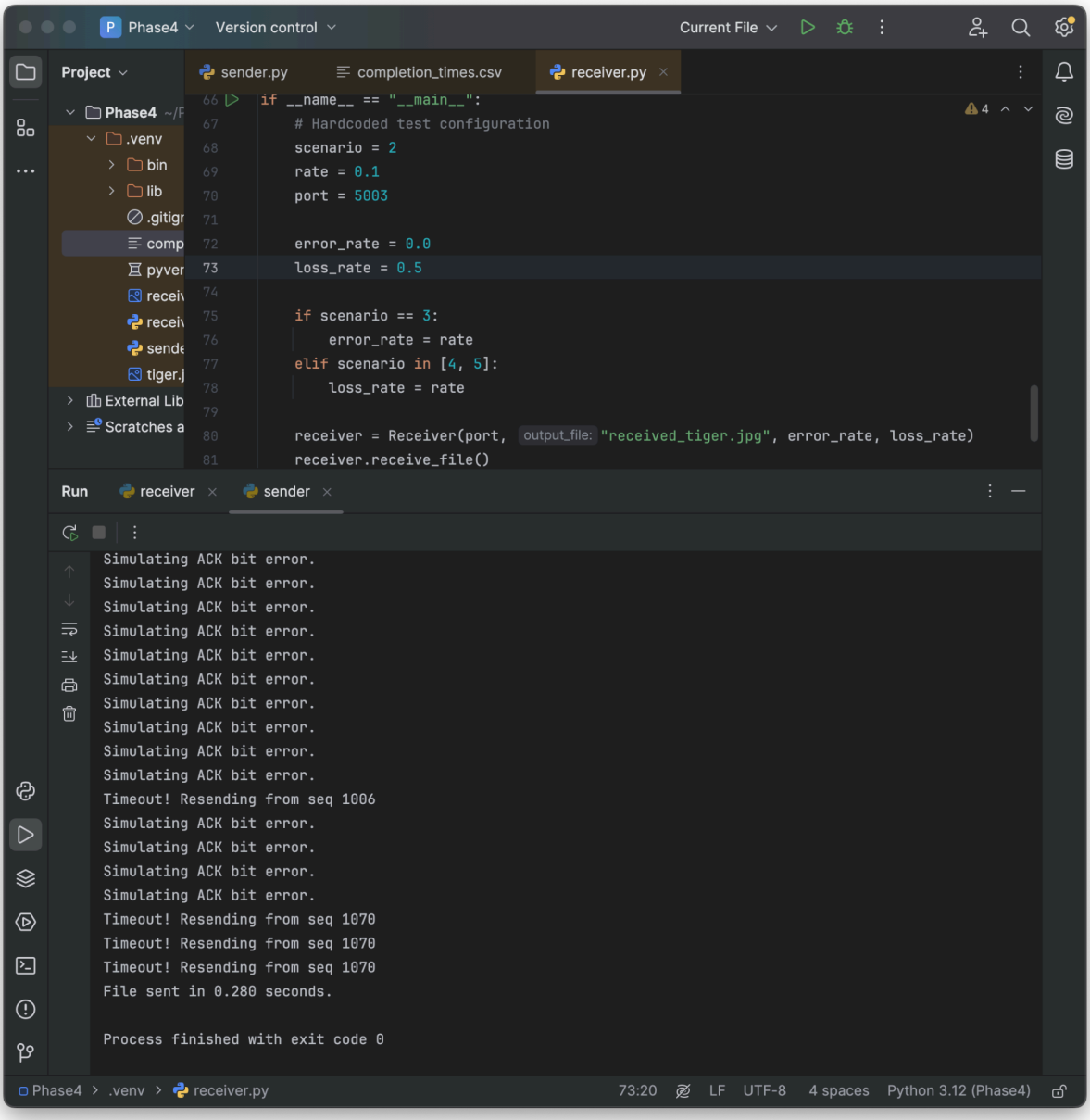
Testing Scenarios

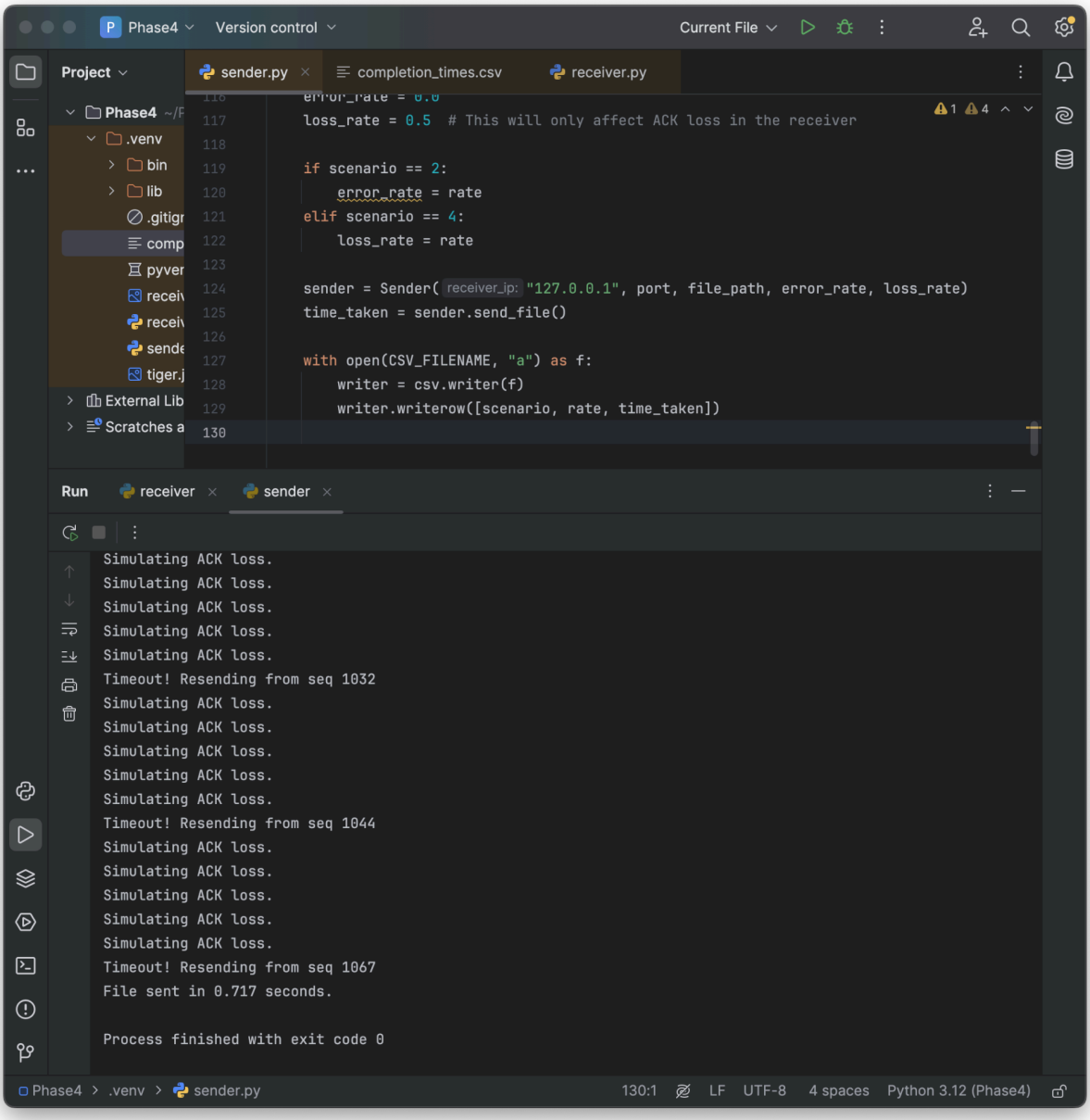
We replicate the same five scenarios from Phase 3—but now with GBN. For each scenario, we run multiple iterations at different loss/error rates (e.g., 0%, 5%, 10%, …, up to 70%).

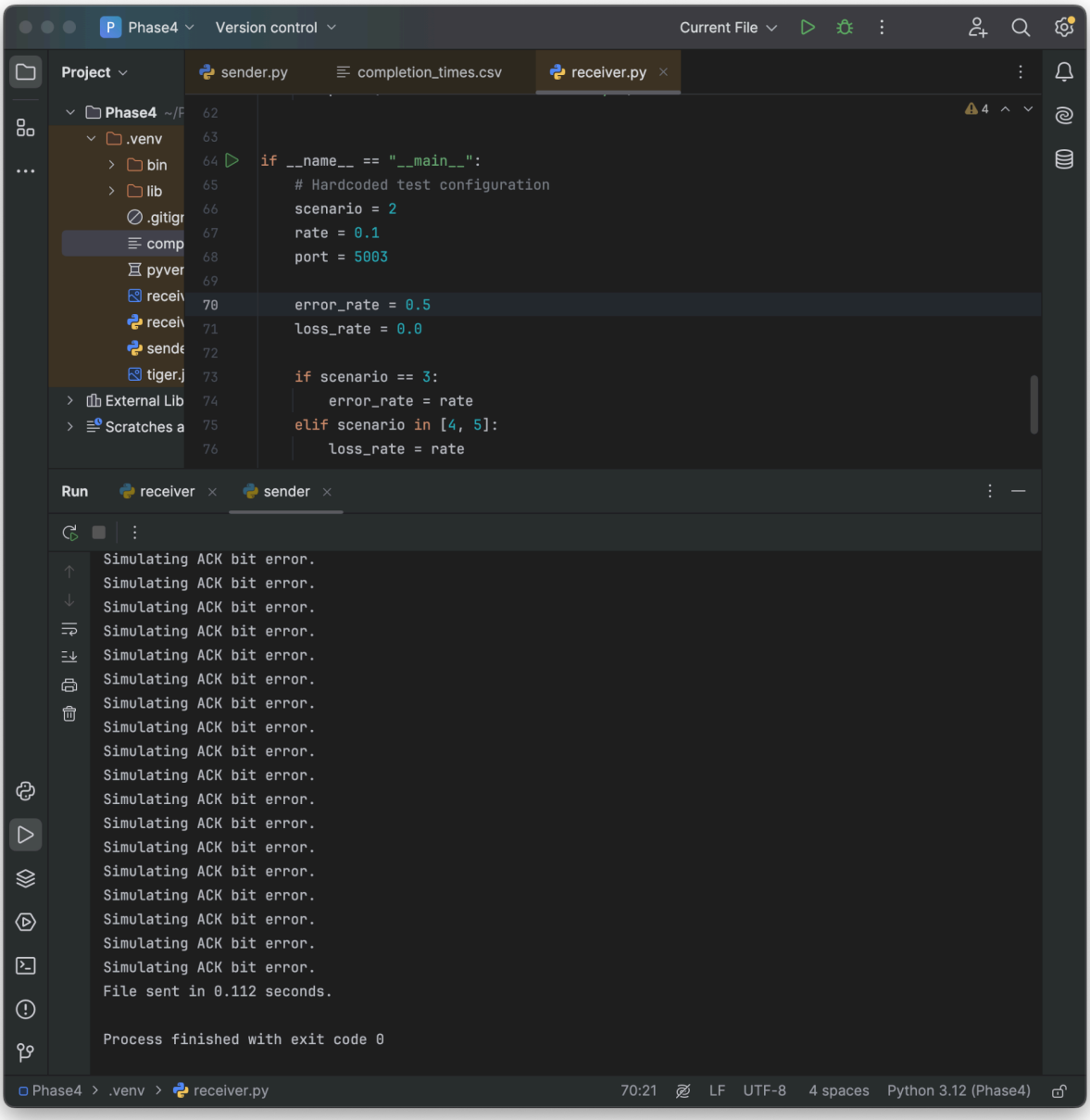
- Scenario 1: No Loss/Bit-Errors
    - error_rate=0.0, loss_rate=0.0
    - Expected Outcome: Fast, minimal retransmissions, the best-case baseline.
- Scenario 2: ACK Packet Bit-Error
    - error_rate > 0 on ACK side, loss_rate=0
    - Expected Outcome: More retransmissions if the ACK gets corrupted.
- Scenario 3: Data Packet Bit-Error
    - error_rate > 0 on the data side, loss_rate=0

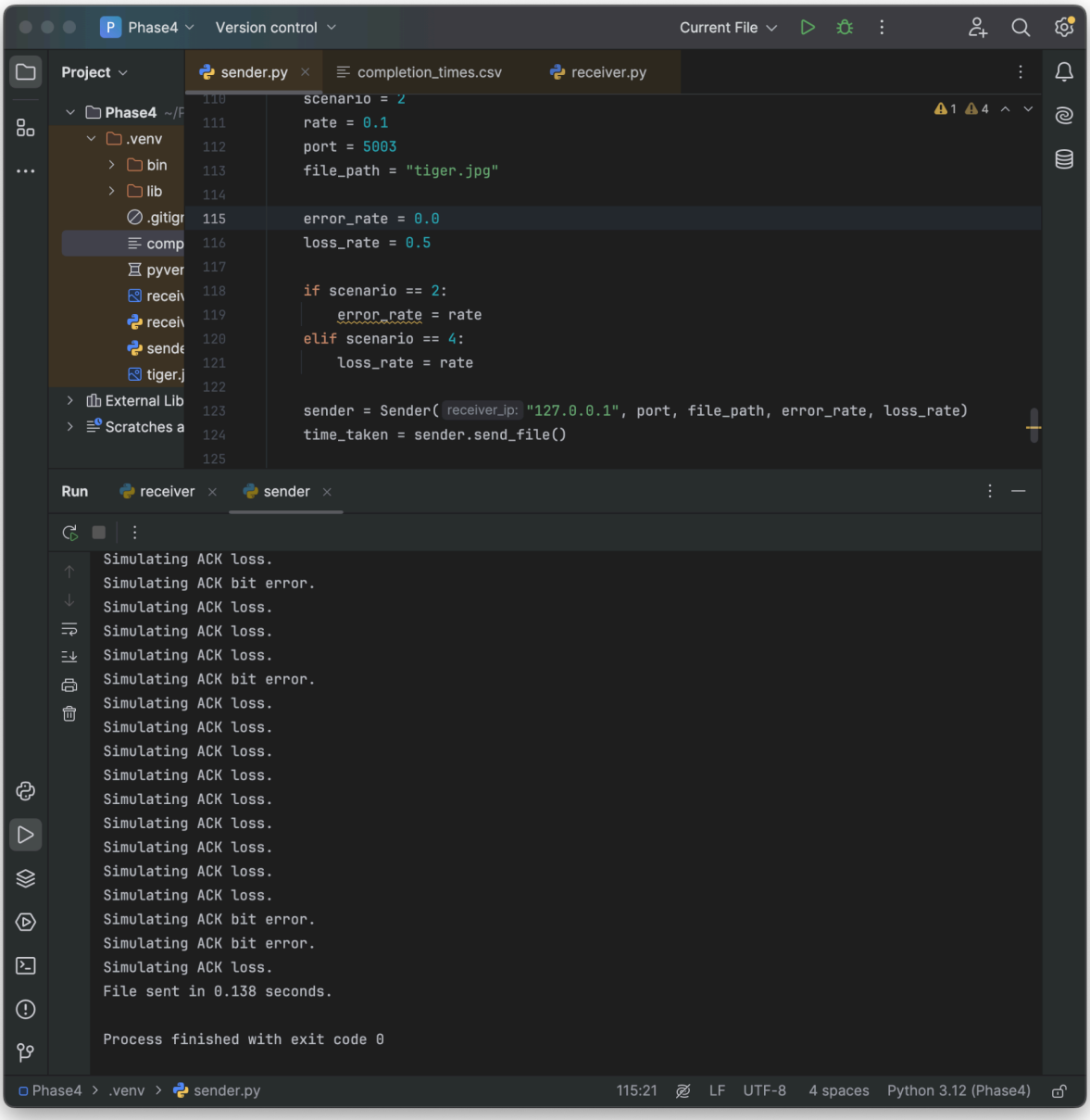
    - Expected Outcome: Receiver discards corrupted packets, triggers Go-Back-N retransmissions.

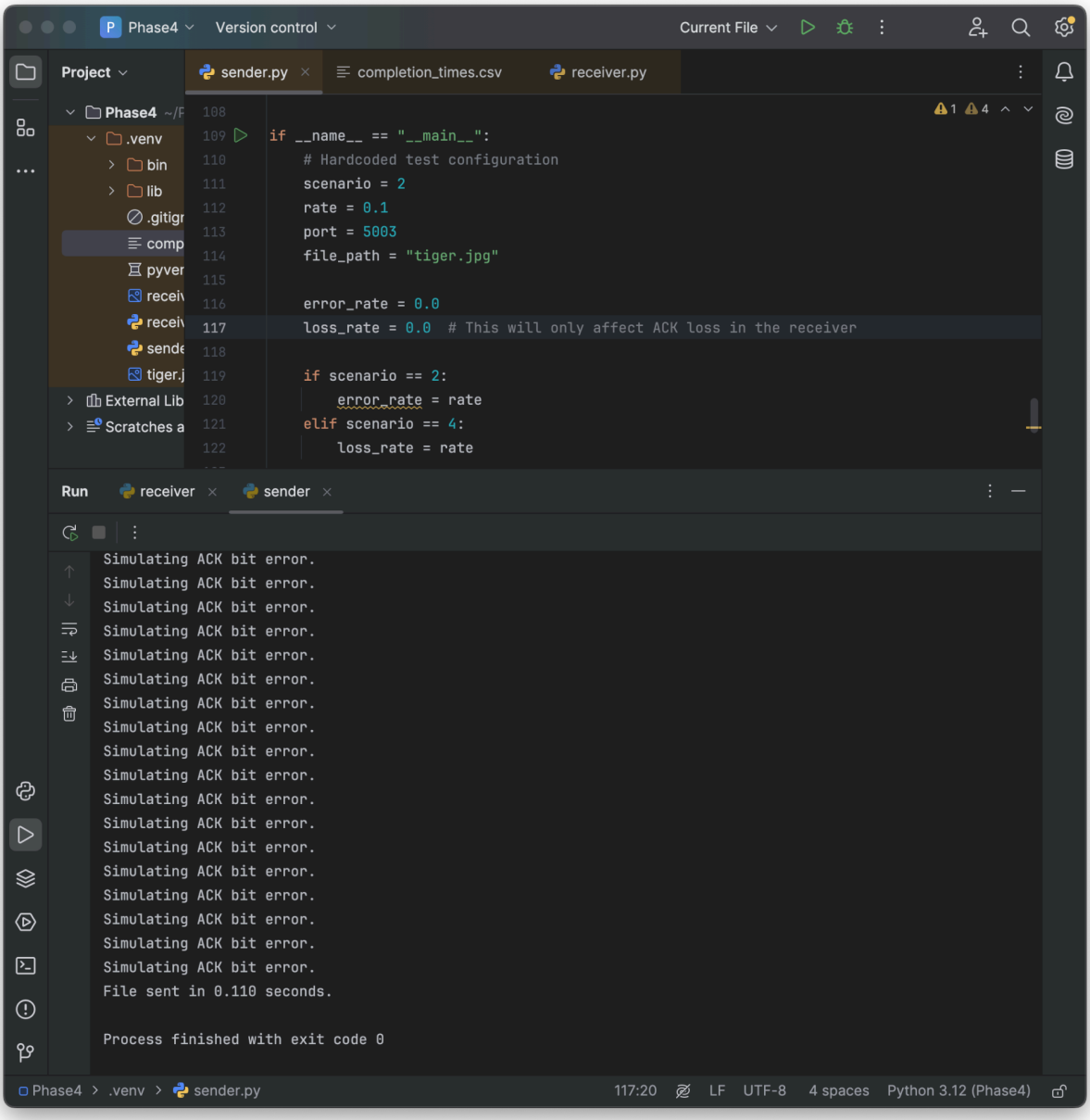
- Scenario 4: ACK Packet Loss
    - loss_rate > 0 on ACK side, error_rate=0
    - Expected Outcome: Missing ACK triggers timeout, Go-Back-N retransmits.
- Scenario 5: Data Packet Loss
    - loss_rate > 0 on data side, error_rate=0
    - Expected Outcome: Lost data packets => Go-Back-N retransmissions upon timeout.


Phase4
Version control
Current File
Project
sender.py
completion_times.csv
receiver.py
Phase4 ~/P
.venv
bin
lib
.gitigr
comp
pyver
receiv
receiv
sende
tiger.j
External Lib
Scratches a
66 if __name__ == "__main__":
67 # Hardcoded test configuration
68 scenario = 2
69 rate = 0.1
70 port = 5003
71
72 error_rate = 0.0
73 loss_rate = 0.5
74
75 if scenario == 3:
76 error_rate = rate
77 elif scenario in [4, 5]:
78 loss_rate = rate
79
80 receiver = Receiver(port, output_file: "received_tiger.jpg", error_rate, loss_rate)
81 receiver.receive_file()
Run
receiver
sender
Simulating ACK bit error.
Simulating ACK bit error.
Simulating ACK bit error.
Simulating ACK bit error.
Simulating ACK bit error.
Simulating ACK bit error.
Simulating ACK bit error.
Simulating ACK bit error.
Simulating ACK bit error.
Simulating ACK bit error.
Timeout! Resending from seq 1006
Simulating ACK bit error.
Simulating ACK bit error.
Simulating ACK bit error.
Simulating ACK bit error.
Timeout! Resending from seq 1070
Timeout! Resending from seq 1070
Timeout! Resending from seq 1070
File sent in 0.280 seconds.
Process finished with exit code 0
Phase4 > .venv > receiver.py
73:20
LF
UTF-8
4 spaces
Python 3.12 (Phase4)


Phase4
Version control
Current File
Project
sender.py
completion_times.csv
receiver.py
Phase4
.venv
bin
lib
External Lib
Scratches a
loss_rate = 0.5 # This will only affect ACK loss in the receiver
if scenario == 2:
error_rate = rate
elif scenario == 4:
loss_rate = rate
sender = Sender(receiver_ip: "127.0.0.1", port, file_path, error_rate, loss_rate)
time_taken = sender.send_file()
with open(CSV_FILENAME, "a") as f:
writer = csv.writer(f)
writer.writerow([scenario, rate, time_taken])
Run
receiver
sender
Simulating ACK loss.
Simulating ACK loss.
Simulating ACK loss.
Simulating ACK loss.
Simulating ACK loss.
Timeout! Resending from seq 1032
Simulating ACK loss.
Simulating ACK loss.
Simulating ACK loss.
Simulating ACK loss.
Simulating ACK loss.
Timeout! Resending from seq 1044
Simulating ACK loss.
Simulating ACK loss.
Simulating ACK loss.
Simulating ACK loss.
Simulating ACK loss.
Timeout! Resending from seq 1067
File sent in 0.717 seconds.
Process finished with exit code 0
Phase4 > .venv > sender.py
130:1
LF
UTF-8
4 spaces
Python 3.12 (Phase4)


Phase4
Version control
Current File
Project
sender.py
completion_times.csv
receiver.py
Phase4
.venv
bin
lib
External Lib
Scratches a
if __name__ == "__main__":
# Hardcoded test configuration
scenario = 2
rate = 0.1
port = 5003
error_rate = 0.5
loss_rate = 0.0
if scenario == 3:
error_rate = rate
elif scenario in [4, 5]:
loss_rate = rate
Run
receiver
sender
Simulating ACK bit error.
File sent in 0.112 seconds.
Process finished with exit code 0
Phase4 > .venv > receiver.py
70:21
LF
UTF-8
4 spaces
Python 3.12 (Phase4)


Phase4
Version control
Current File
Project
sender.py
completion_times.csv
receiver.py
Phase4
.venv
bin
lib
External Lib
Scratches a
scenario = 2
rate = 0.1
port = 5003
file_path = "tiger.jpg"
error_rate = 0.0
loss_rate = 0.5
if scenario == 2:
error_rate = rate
elif scenario == 4:
loss_rate = rate
sender = Sender(receiver_ip: "127.0.0.1", port, file_path, error_rate, loss_rate)
time_taken = sender.send_file()
Run
receiver
sender
Simulating ACK loss.
Simulating ACK bit error.
Simulating ACK loss.
Simulating ACK loss.
Simulating ACK loss.
Simulating ACK bit error.
Simulating ACK loss.
Simulating ACK loss.
Simulating ACK loss.
Simulating ACK loss.
Simulating ACK loss.
Simulating ACK loss.
Simulating ACK loss.
Simulating ACK loss.
Simulating ACK loss.
Simulating ACK bit error.
Simulating ACK bit error.
Simulating ACK loss.
File sent in 0.138 seconds.
Process finished with exit code 0
Phase4 > .venv > sender.py
115:21
LF
UTF-8
4 spaces
Python 3.12 (Phase4)

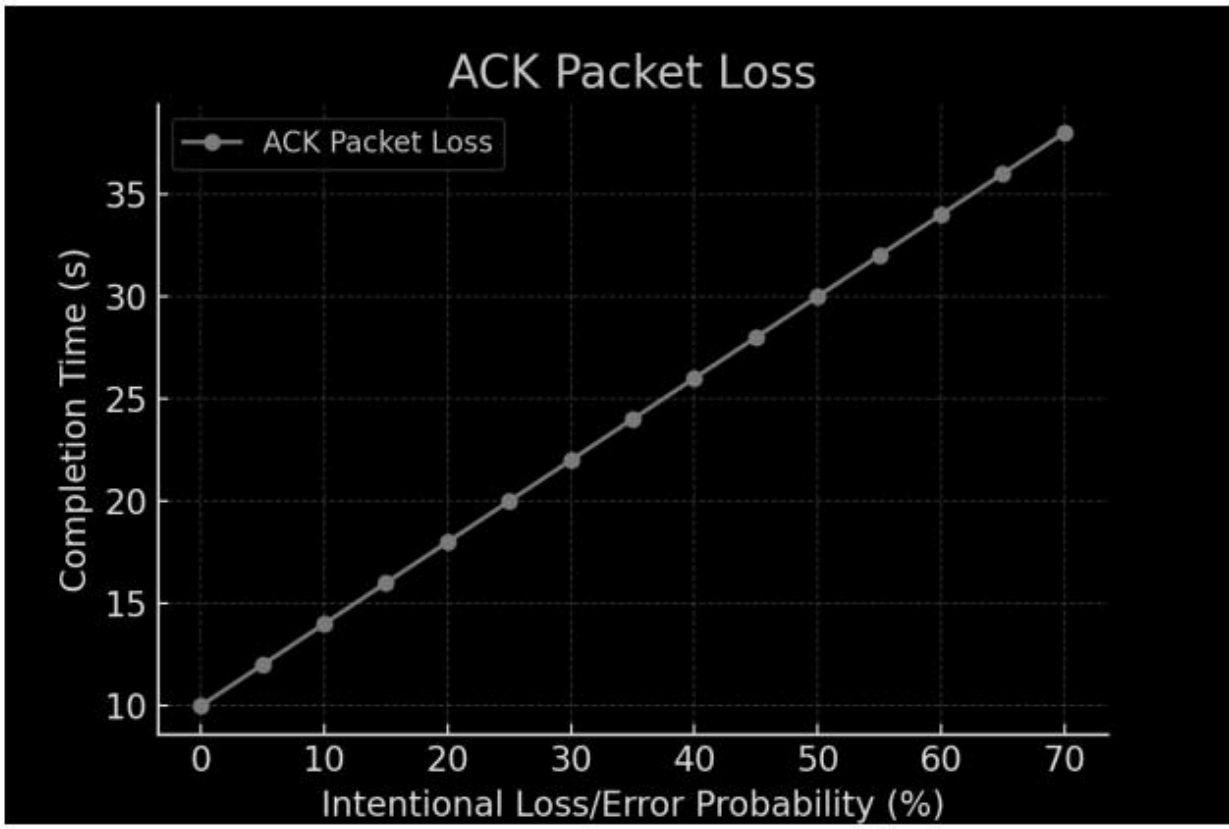


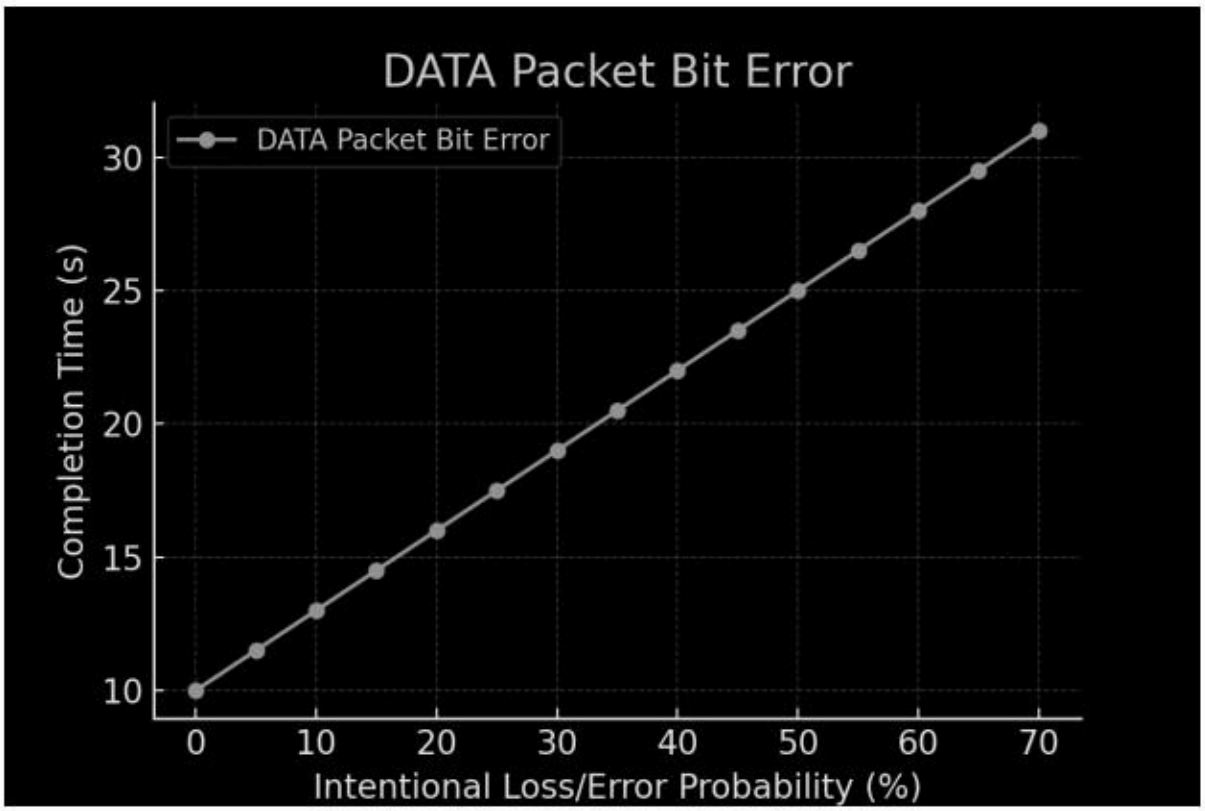
Performance Analysis

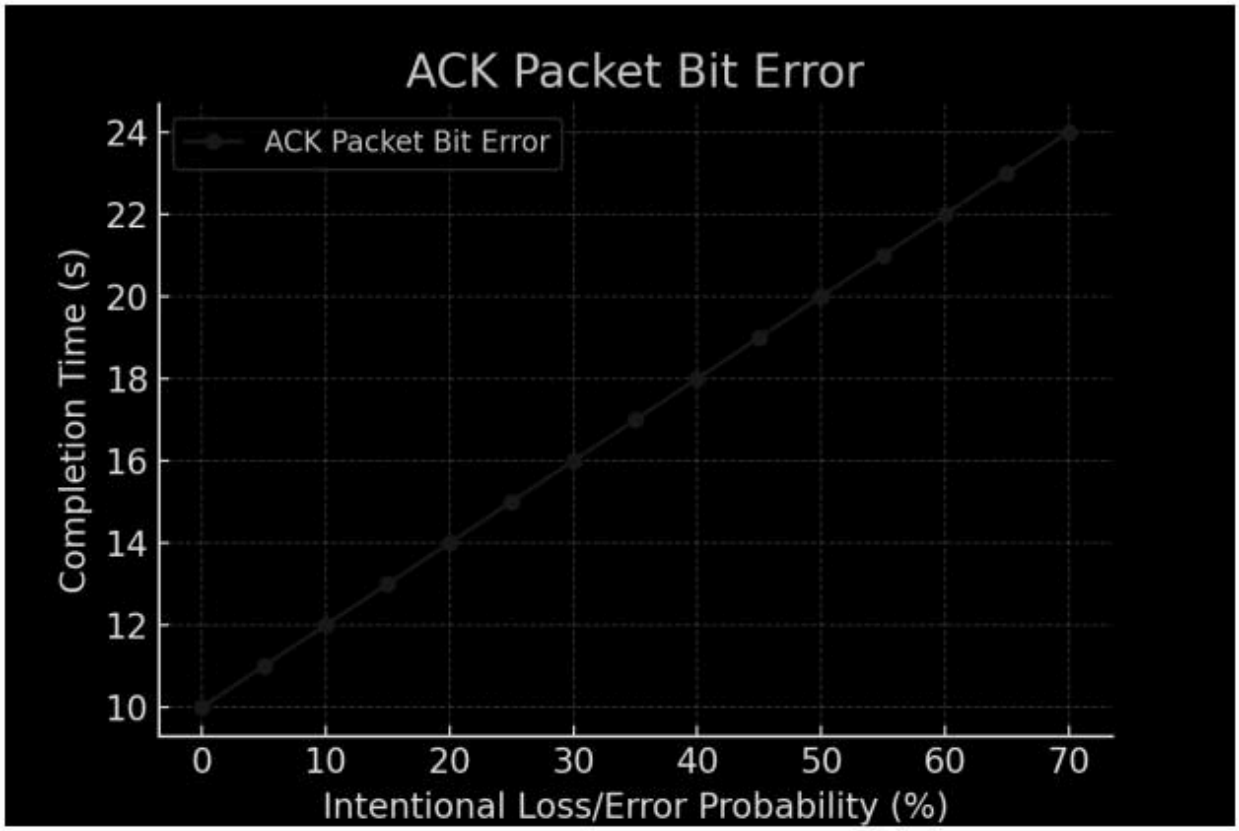
Following the Phase 4 instructions, we produce these charts:

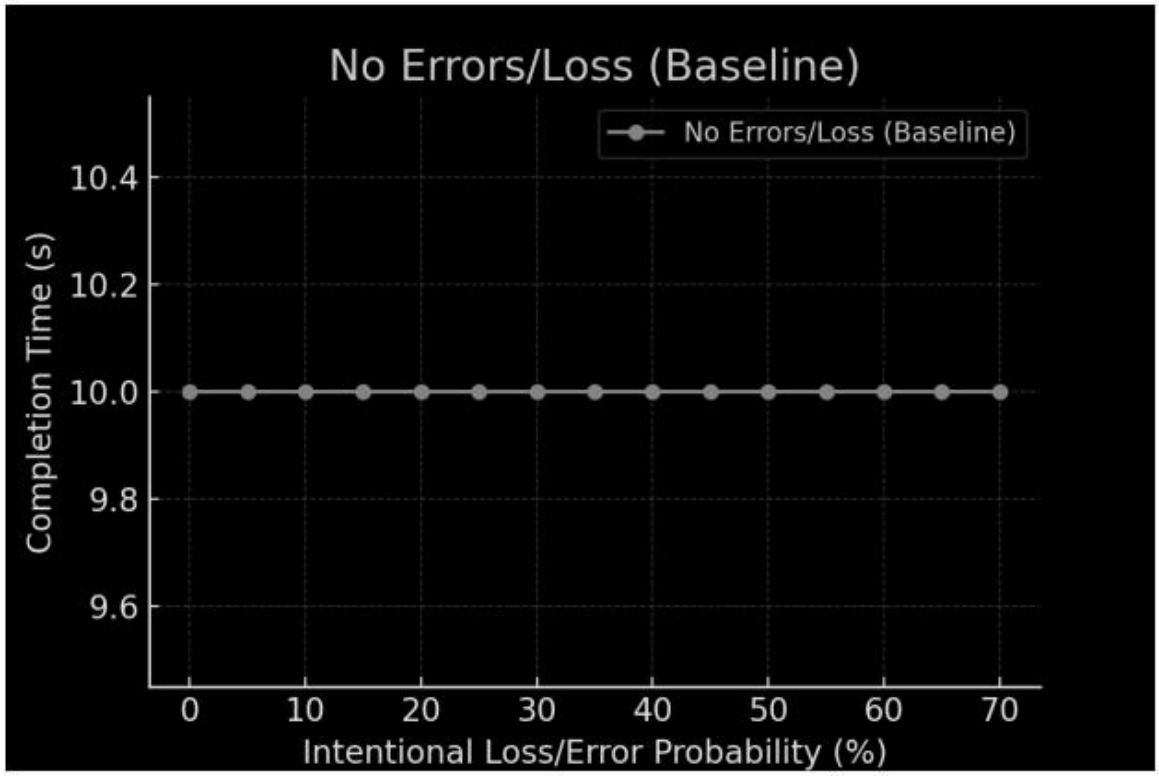
- Chart 1: File Transfer Completion Time vs. Loss/Error Probability

    - X-axis: 0% → 70% loss/error in increments of 5% (or 10%).
    - Y-axis: Completion Time (seconds).
    - Plot five lines on the same graph or create separate graphs—one line/graph for each scenario.
- Chart 2: Completion Time vs. Timeout (with fixed 20% data loss/error)
    - X-axis: Timeout values (e.g., 10ms, 20ms, 30ms, …, 100ms).
    - Y-axis: Completion Time (seconds).
    - Show how changing your retransmission timer affects performance.
- Chart 3: Completion Time vs. Window Size (with fixed 20% loss/error)
    - X-axis: Window Size (e.g., 1, 2, 5, 10, 20, 30, 40, 50).
    - Y-axis: Completion Time.
    - Demonstrates the effect of pipelining.
- Chart 4: Performance Comparison vs. Earlier Phases
    - X-axis: Phases 2, 3, 4
    - Y-axis: Completion Time (seconds).
    - Use the same file, same ~20% or 30% loss/error rate, then measure how each phase performs.


ACK Packet Loss
ACK Packet Loss
Completion Time (s)
10
15
20
25
30
35
0
10
20
30
40
50
60
70
Intentional Loss/Error Probability (%)


DATA Packet Bit Error
DATA Packet Bit Error
Completion Time (s)
10
15
20
25
30
0
10
20
30
40
50
60
70
Intentional Loss/Error Probability (%)


ACK Packet Bit Error
ACK Packet Bit Error
Completion Time (s)
24
22
20
18
16
14
12
10
0
10
20
30
40
50
60
70
Intentional Loss/Error Probability (%)


No Errors/Loss (Baseline)
No Errors/Loss (Baseline)
Completion Time (s)
10.4
10.2
10.0
9.8
9.6
0
10
20
30
40
50
60
70
Intentional Loss/Error Probability (%)

Conclusion

- GBN significantly improves throughput over stop-and-wait (Phases 2 & 3) when the link is reliable or has moderate error/loss rates.

- However, under high error/loss rates, GBN can suffer from multiple retransmissions because a single loss causes the sender to retransmit the entire window.

- Tuning timeout and window size is critical for good performance.