EECE 4830-5830 Network Design, Dr. Vinod Vokkarane

Programming Project Phase 5 (Part 1):
Implement TCP protocol over an unreliable UDP channel

Project description: In this extended phase, you will implement a simplified version of the **Transmission Control Protocol (TCP)** over an **unreliable UDP socket connection**. The UDP channel simulates the IP network layer, and your goal is to design all necessary TCP functionality at the transport layer.

You are required to implement the following TCP protocol features:

1. **Connection Setup and Teardown**
    a. Simulate the TCP 3-way handshake: `SYN`, `SYN-ACK`, `ACK`.
    b. Implement proper connection closing: `FIN`, `ACK` sequences.
2. **Checksum Implementation**
    a. Add a checksum field in your TCP segment format.
    b. Manually compute and validate checksums to detect corrupted segments.
3. **Reliable Data Transfer (RDT)**
    a. Ensure that packets are delivered in order and without duplication.
    b. Use sequence numbers, acknowledgments, and retransmissions to recover lost data.
4. **Timeout Estimation and RTT Calculation**
    a. Measure **Sample RTT** for each acknowledged packet.
    b. Use the **Estimated RTT** and **Deviation** to compute **Retransmission Timeout (RTO)** using TCP's exponential weighted moving average formulas.
5. **Dynamic Window Size (Sliding Window Mechanism)**
    a. Adjust the sender's congestion window (cwnd) dynamically based on ACKs and network feedback.
    b. Maintain proper receiver window (rwnd) to simulate flow control.
6. **Congestion**
    Implement the following standard TCP congestion control strategies:
    a. **Slow Start (SS)** – exponential window growth until threshold.
    b. **Additive Increase Multiplicative Decrease (AIMD)** – linear growth and sharp decrease after congestion.

c. **TCP Tahoe** – uses slow start, and drops to cwnd = 1 on packet loss.
d. **TCP Reno** – implements Fast Retransmit and Fast Recovery upon triple duplicate ACKs.

Required Output and Plots

To evaluate your TCP implementation, you must generate the following graphs based on a single simulation run:

- **Plot 1:** Congestion Window Size vs Time
- **Plot 2:** Sample RTT vs Time
- **Plot 3:** Retransmission Timeout (RTO) vs Time

In addition, you must regenerate all **Phase 5 performance plots** using your TCP implementation:

- Completion Time vs Loss/Error Rate (0%–70%)
- Completion Time vs Timeout Value (10ms–100ms)
- Completion Time vs Window Size (1–50)
- Performance comparison of Phase 2, Phase 3, Phase 4 (and TCP)

1. Cross-Layer Simulation: MAC + TCP Interaction (+35%)

Objective:

Simulate the interaction between a basic MAC layer (e.g., CSMA/CA) and TCP over a shared channel. Study the effect of MAC-level contention and collisions on TCP retransmissions, congestion control behavior, and overall throughput.

- **Project Description:**
  - Implement a simplified MAC layer simulating CSMA/CA logic.
  - Multiple TCP flows should share a single channel with channel access coordination:
    - Channel sensing (carrier sensing)
    - Collision simulation (if two or more send simultaneously)
    - Randomized backoff timer (before retransmission)
    - Model channel state (idle/busy) and sender behavior on collisions.
- **Scenario Examples:**
  - 2, 3, or 4 concurrent TCP flows sharing the same channel
  - Comparison between MAC-enabled TCP vs ideal TCP (no MAC contention)

2. Delay-Based Congestion Control (TCP Vegas) + Adaptive RTO (+15%)

Objective:

Implement a delay-sensitive congestion control strategy based on TCP Vegas, and enhance your retransmission timeout estimation using adaptive techniques like Kalman filtering or dynamic EWMA.

- **Project Description:**
  - Track **BaseRTT** and use it to compare expected vs actual throughput.
  - Adjust congestion window accordingly:
    - Decrease if nearing congestion

    - Increase if underutilized
  - Replace fixed RTO estimation with:
    - Kalman filter or
    - Modified EWMA with adaptive parameters
- **Scenario Examples:**
  - TCP Reno vs TCP Vegas in same network
  - Delay spike scenario to test RTO response

3. Security-Aware TCP Simulation (+25%)

Objective:

Simulate adversarial scenarios targeting TCP behavior to evaluate how standard TCP reacts to malicious or manipulated network traffic.

- **Project Description:**

  Implement three types of TCP-layer attacks:
  - **ACK Spoofing** – send forged ACKs with incorrect sequence numbers.
  - **Duplicate ACK Flooding** – rapidly send duplicate ACKs to trigger fast retransmit.
  - **RTT Manipulation** – artificially delay or speed up ACKs to distort RTT estimates.
- **Scenario Examples:**
  - TCP flow with/without attacker interference
  - Monitor response under individual and combined attacks

4. Multi-Threaded TCP Sender and Receiver (+25%)

Objective:

Implement a multi-threaded version of your TCP protocol to improve efficiency and concurrency in packet sending, receiving, and timeout handling. This project

emphasizes system-level design and synchronization to simulate real-world TCP behavior.

- **Project Description:**
  - Use multiple threads within both sender and receiver applications to parallelize operations:
    - **Sender Threads**:
      - One thread for sending data segments
      - One thread for receiving ACKs
      - One thread for managing retransmission timers
    - **Receiver Threads**:
      - One thread for receiving and processing data
      - One thread for sending ACKs
  - Implement proper synchronization:
    - Use mutexes, locks, or semaphores to protect shared resources (e.g., congestion window, buffers, timers).
    - Ensure threads do not race to access or modify variables (e.g., cwnd updates, packet queues).
  - Simulate realistic concurrent behavior:
    - Overlapping send and receive operations
    - Concurrent timer management
    - Overlapping computation and I/O
- **Scenario Examples:**
  - Compare single-threaded and multi-threaded TCP under the same test conditions.
  - Run high-traffic scenarios to highlight concurrency benefits.

Expectations: In this phase of the project, you will learn about the basic principles used to provide pipelined reliable data transfer over a data channel with bit errors and packet losses.

Programming language: C, C++, Python or Java (your choice).

Deliverables:

1. **ReadMe File**: Name of the team members, list the names of files submitted and their purpose, and explain steps required to set up and execute your program. Also include instructions to run and test the following scenarios:
    1. No loss or bit error
    2. ACK packet bit error (simulate bit errors in ACKs)
    3. Data packet bit error (simulate bit errors in data packets)
    4. ACK packet loss (simulate dropped ACKs)
    5. Data packet loss (simulate dropped data packets)
    6. TCP-specific scenarios such as congestion control (Slow Start, Tahoe, Reno), multiple flow fairness, and optional attack/resilience tests (if extra credit attempted)
2. **Design Documents**: This document should contain 4 sections: Introduction, flow chart, code description, and snapshot of results.
    1. Introduction: Should provide the TA a clear picture of your project design.
    2. Flowchart: Flowchart should demonstrate the author's logic. This section is optional. Ref: https://creately.com/blog/diagrams/flowchart-guide-flowchart-tutorial/.
    3. Code Description: Explain the logic of your code based on functional blocks. Please do not just copy and paste commented code into this document.
    4. Execution Screenshots: Include the screenshots of your executions of the 5 difference scenarios in this section.
    5. Performance Plots: Plot charts for completion time for all the data transfer options for the same file starting from 0% loss/error to 60% loss/error in increments of 5%.
3. Plot completion time graphs for file transfers using at least a 500KB file. Each point on the plot can be an average of multiple runs.

For accurate measurements, remove print/debug statements before measuring time.

1. **Chart 1: TCP performance with varying loss/error rate**
   X-axis: Intentional loss probability (0% - 70% in 5% increments)
   Y-axis: File Transfer Completion Time
2. **Chart 2: Completion time vs timeout value**
   Fixed loss/error probability (e.g., 20%)
   X-axis: Retransmission Timeout (10ms – 100ms)
   Y-axis: File Transfer Completion Time
3. **Chart 3: Completion time vs window size**
   Fixed loss/error probability (e.g., 20%)
   X-axis: Window size (1, 2, 5, 10, 20, 30, 40, 50)
   Y-axis: File Transfer Completion Time
4. **Chart 4: Protocol performance comparison**
   Compare performance under same conditions
   X-axis: Protocols (Phase 2, Phase 3, Phase 4, TCP)
   Y-axis: File Transfer Completion Time
5. **Additional Charts:**

   Window size (cwnd) vs time, Sample RTT vs time, RTO vs time, (Optional) Fairness or throughput comparison for multi-flow experiments

4. **Sender and Receiver source files**: well commented source code; mention ALL references for reuse of source code (if any).
5. **Deliverables for Extra Credit:**
   1. Design and Implementation Description
      1. Include the following for each advanced project:
      2. Brief Introduction: One paragraph summarizing the goal of the implemented feature (e.g., cross-layer interference, delay-based window adjustment, attack simulation).

3. Feature Design Explanation: Describe core logic, implementation approach, and any architectural diagrams (e.g., MAC scheduler flowchart, thread structure diagram, RTT estimation formula).
4. Execution Workflow: Outline how threads, timers, or logic interact during runtime. Include pseudocode or high-level flow diagrams where applicable.

2. Performance Graphs

Provide the following charts and label them clearly with figure titles and axes:

1. Chart Type, Description
2. TCP Completion Time vs Loss Rate, For MAC + TCP, Security-aware TCP, or Vegas
3. TCP Retransmissions vs Number of Contending Senders, For MAC + TCP comparison
4. Congestion Window (cwnd) vs Time, Required for TCP Vegas, Security-aware TCP, and MAC comparisons
5. RTT and RTO vs Time, Required for Vegas and RTT manipulation scenarios
6. Completion Time: Single-threaded vs Multi-threaded, For Multi-threaded TCP
7. Throughput Comparison (Baseline vs Attacks), For Security-aware TCP
8. Optional: Kalman Filter Estimates, RTT mean and variance tracking over time

3. Execution Logs and Screenshots
    1. Annotated logs or debug printouts showing:
    2. MAC-level collisions and backoffs
    3. Attack injection events and TCP response
    4. Vegas-style congestion control decisions
    5. Simultaneous thread execution for multi-threading
    6. Screenshots of terminal outputs or GUI (if applicable)

4. Analysis & Discussion (Short Report)

Discuss the outcome of each advanced feature:

1. MAC + TCP: Effect of contention on congestion control
2. TCP Vegas: Comparison with loss-based TCP, stability of RTO
3. Security-aware TCP: TCP behavior under attack, protocol vulnerabilities
4. Multi-threaded TCP: Responsiveness and performance improvements
5. Summarize key insights with 3–5 bullet points per feature.
6. Mention known limitations or areas for further improvement.

6. **Individual Contribution (contribution.txt/doc):**
    1. Bulleted list of tasks implemented by each team member and
    2. Teammate rating between 1 (not good) – 5 (excellent) on
        1. Project time commitment
        2. Design contribution
        3. Coding contribution
        4. Debugging contribution, and
        5. Report preparation (confidential submission for each member).

One team member can submit all your documents (except extra credit items) in a single compressed file with the name: **TeamNumber_Phase5.zip.**

References:

1. Socket Programming
2. Python: Socket Programming (Textbook Sec 2.7)
    a. JAVA: Socket Programming (PDF attached with Phase 1 assignment)
    b. Socket Programming by InfoWorld
    c. C/C++:
    d. Linux Gazette's Socket Programming

    e. Beej's Guide

3. UDP: RFC768
4. Principles of Reliable Data Transfer, Section 3.4.1, Page 206 -207, Kurose/Ross (7th Ed).