ASSIGNMENT – 4 (miniProject)

Adaptive Hybrid Transport Protocol for Games

School of Computing, National University of Singapore

This is a Group Project with group size of 4 members per group. As informed early, pls register your groups @ *Canvas->**People->A4-ProjectGroup***. We strongly recommend teams to **start early**.

You are allowed to use any tool/opensource codes to support you in this project, just acknowledge them in the report. We will be looking only for correct functionality of the protocol and features. This project also contributes to an ongoing research study on improving transport-layer learning.

Specification: Introduction

This assignment challenges students to design and implement a hybrid transport layer protocol that dynamically manages both reliable and unreliable data delivery over UDP (or QUIC) for real-time multiplayer games. The protocol should allow game developers to mark data packets based on their reliability requirements — e.g., reliable for critical game state updates, unreliable but low-latency for movement or voice chat packets.

Note: For consistency across implementations, teams should assume packets carry simple mock game data (e.g., numeric IDs, short strings, or JSON snippets). This ensures comparability of latency and delivery results across submissions.

You can use any programming language, Python is recommended for implementation.

1. Objectives

a) Understand and apply key concepts of transport layer design and protocol trade-offs.
b) Implement hybrid data transmission (reliable and unreliable) using UDP (or QUIC).
c) Evaluate latency, jitter, throughput, and reliability (packet delivery ratio) under different network conditions.
d) Promote inquiry-based learning and collaborative problem solving.
e) Visualize and discuss trade-offs between reliability and latency using plots or summary tables.
f) Contribute to ongoing research analysis of learning impact.

2. Assignment Tasks

a) Design a custom transport layer protocol (H-UDP/H-QUIC) with separate channels for reliable and unreliable data.

> *Implementation Note:* Define your packet header explicitly — for example,
> `| ChannelType (1 B) | SeqNo (2 B) | Timestamp (4 B) | Payload |`.
> Include this format in your report.

b) Provide a H-UDP API that allows marking data as reliable or unreliable by the sending application. You can call this as *gameNetAPI*.

c) The *gameNetAPI* should receive the packets and pass to receiver application. The *gameNetAPI* reorders the packets (if they are not in order) in reliable channel.

d) Assume each packet is independent, not chunks of a file and hence there no need to reconstruct.

e) Implement retransmission based on timer, buffering and reordering for reliable packets. [Refer to Kruse and Ross text book for RDT]. Reliable packets should be delivered to the receiver application in order. If any packet is lost and retransmission is not reached by *t* milliseconds threshold, you should skip that packet and display rest of the data.

> *Clarification:* Let `t = 200 ms` by default, but you may tune this value in experiments; report your chosen threshold in the analysis section.

f) A sender application (represents game client) that can tag outgoing data packets as reliable or unreliable randomly and sends them through your *gameNetAPI*.

g) A receiver application displays the data received through *gameNetAPI*. Print logs showing SeqNo, ChannelType, Timestamp, retransmissions, packet arrivals and RTT, etc. for clarity in demos.

h) Simulate real-world network conditions (packet loss, delay, jitter) using a network emulator (e.g., `tc-netem` for linux, `clumsy` for windows; or use mininet with 'tc-netem'). You may briefly justify your chosen network emulator in the report (e.g., realism, reproducibility, ease of use).

[optional] Alternatively, you can write your own emulator to work between your sender/receiver applications, which intercepts the packets and introduces artificial delay, jitter, packet drops, etc.

> When reporting results, describe your test duration (30 s–1 min) and packet rate (10–100 packets/s) for reproducibility.

i) The H-UDP system/API should measure performance metrics: Latency (one-way or RTT) and Jitter following [RFC 3550]. Throughput = *total bytes received / duration*. Packet delivery ratio = *Number of Packets Received/ Number of Packets*

*Sent * 100%*. Report metrics separately for reliable and unreliable channels. Present results for at least two network conditions (e.g., low loss < 2%, high loss > 10%).

3. Details

a) **Logical UDP Channels:** Use **Single UDP socket with multiple logical channels**
   i. Each packet includes a **header field (channel ID or type)**:
      - `Channel Type = 0` → reliable
      - `Channel Type = 1` → unreliable
   ii. Each packet should include a sequence number and timestamp. The receiver should buffer and reorder reliable packets using either the Selective Repeat or Go-Back-N method..
   iii. The receiver demultiplexes packets by `Channel Type` and processes them differently.

b) **QUIC Strems:** If you are using QUIC, then,
   i. Each QUIC stream naturally acts as a "channel".
   ii. QUIC allows marking streams as reliable/unreliable or implementing partial reliability policies.
   iii. For example:
      - Stream 1 = reliable control/game state.
      - Stream 2 = unreliable position updates.
   iv. You may use existing QUIC libraries (e.g., aioquic for Python, quic-go for Go) to implement partial reliability instead of coding QUIC from scratch. Your API simply becomes a wrapper for QUIC API.

4. Submissions

a) ZIP of the Source code or text file containing URL to github public repo.
b) Submit a short technical report (4 pages) describing design choices, *gameNetAPI spec*, test setup, results, and reflection. Include, Protocol header diagram, protocol flowchart, charts for latency/jitter/throughput. Add acknowledgements for any external code, libraries, or AI-assisted tools used. Add instructions for compiling and running your application source code.
   – [Optional]Comparison table (H-UDP/HQUIC vs UDP baseline).
   - Charts and latency plots can be auto-generated using Python/matplotlib or Excel.
c) Submit a short demo (screen capture) showing… 1) reliable and unreliable channel data at sending and receiving applications, 2) retransmissions, 3) reordering, 4) performance metrics

d) Your reflection should be combination of learning reflection of all members of the team, answering the following questions.

   i. What new knowledge you learned through this experiment? If you have used LLMs, any specific insights/knowledge/skills your acquired through LLMs.
   ii. What are the trade-offs in transport layer protocol design?

e) Combine all the submissions to a ***single ZIP/RAR file***, rename it to **your group number** and upload to Canvas.
f) Peer-Review will be conducted after the due date to access individual contributions.
g) Submission Due: **Friday 07-Nov-2025** Late submission penalty: **10% per day.**

4. Evaluation Criteria

a) Protocol Correctness: Correct design and implementation of the H-UDP Protocol [30%]
b) Measurements: measure performance metrics: latency, jitter, throughput and packet delivery ratio. [30%]
c) Sender/receiver API design (10%) — integration quality and usability.
d) Implementation of Sender and receiver applications that use the API to send/receive, show received packets and performance metrics
e) Technical report (20%) — clarity, analysis, and reflection.
f) Video Demo (screen capture) of the features (10%)

5. Data Collection for Educational Research

To assess the educational impact, the following data will be collected:

a) Pre- and post-assignment survey on students' conceptual understanding of transport protocols.
b) Reflection prompts evaluating critical thinking and design decision-making.
c) System logs and submission data (e.g., latency, throughput metrics) for performance analytics.
d) Peer review ratings on teamwork and contribution.
e) Optional interviews or focus group discussions for qualitative insights.
f) Participation in surveys is is optional and will not influence grades or evaluation.

Data Ethics: All collected logs will be anonymized and used solely for educational-research purposes under institutional data-protection policies.

If you have any question/clarification, discuss through **discord forum 'assignment4'**.

[We have a Zero Tolerance for Plagiarism Policy. If you are here only for marks/grades, just let me the grade/mark you prefer to have!]

Bhojan Anand /NUS

Learning in depth is the key focus of CS3103. Try more beyond the questions above.

'Students who approach education from a **deep-learning perspective** make significant improvements, remember what they learn, and feel empowered to make a difference. Those who take a surface approach to learning may get good grades, but they rarely benefit much in the long term'. – *from several research findings.*

Students Who Take a Deep Approach--

- *Attempt to understand material for themselves*
- *Seek rigorous and critical interaction with knowledge content*
- *Relate ideas to previous knowledge and experience*
- *Discover and use organizing principles to integrate ideas*
- *Relate evidence to conclusions*
- *Examine the logic of arguments*