INF-2300 Computer Communication: Reliable Transport Protocol

Ahsan Ghafoor
Department of Computer Science
UiT The Arctic University of Norway
Tromsø, Norway
ahgha0019@uit.no

***Index Terms*—Go-Back-N, Sliding Window, Cumulative ACK, Checksum, Reliable Transport**

I. Introduction

We implement a reliable transport protocol that delivers an in-order, exactly-once byte stream over an unreliable channel with loss, delay, and data corruption. The simulator exposes Application, Transport, and Network layers, where only the Transport layer is modified. Our design uses Go-Back-N (GBN) ARQ with a single retransmission timer, cumulative acknowledgments, and a 16-bit one's-complement checksum computed over header and payload.

A. Objectives

- Achieve correct, in-order delivery under loss, reordering, and corruption.
- Use GBN with one timer and cumulative ACKs.
- Detect corruption via the Internet checksum [1].
- Maintain thread safety and clear invariants.
- Characterize behavior under varying impairment levels.

II. Technical Background

A. OSI Model

The simulator follows the OSI reference model, focusing on the Transport layer between the Application and Network layers. The Transport layer provides reliability, ordering, and multiplexing, abstracting away lower-layer unreliability. [2, 3, 4, 5].

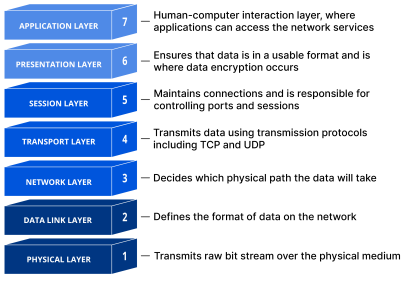

Fig. 1. The layers of the OSI model

B. Go-Back-N

GBN is a sliding-window ARQ, where the sender may have up to N unacknowledged frames in flight. When loss/corruption occurs, it retransmits from the first missing frame onward, while the receiver cumulatively ACKs the next expected sequence number [6].

C. Channel Impairments

Packets are routed units of data. Packet loss happens when packets fail to reach the destination, causing retransmissions and can degrade throughput and latency [7]. Bit errors during transmission corrupt packets. Corruption is detected by checksums and typically triggers discard and recovery by the transport protocol [8].

III. Design and Implementation

A. Assumptions

The Network layer may drop, delay, or corrupt *data*. Traffic is one-way (Alice→Bob), and the reverse direction would be symmetric.

B. Packet Design

Each packet contains `data`, `seqnum`, `acknum`, where ≥ 0 marks ACK, and -1 marks DATA, and a 16-bit checksum. The checksum is the Internet-style one's-complement sum over a serialized header `seqnum|acknum|length` and payload, and odd-length buffers are padded to even bytes. Covering header and payload detects both header and data errors.

C. Sender (GBN)

The sender tracks `base`, `next_seqnum`, fixed window N=5, sequence space S=2N, a `send_buffer` (in-flight), and a single timer. On app input, if the window has room, it builds DATA at `next_seqnum`, buffers it, starts the timer if this is the first in-flight segment, sends to the network, and advances `next_seqnum` modulo S. A cumulative ACK that advances `base` removes all acknowledged entries, stops the timer if no in-flight data remains, otherwise resets it. Window openings trigger draining of a FIFO app queue. On timeout, the sender retransmits all unACKed packets in [`base`, `next_seqnum`).

D. Receiver (GBN)

The receiver maintains `expected_seqnum` and `last_ack_sent`. Corrupt DATA causes an immediate ACK for `expected_seqnum`. If `seqnum` equals `expected_seqnum` the payload is delivered, `expected_seqnum` is incremented modulo S, and a cumulative ACK for the new value is sent. Out-of-order or duplicate packets are dropped and trigger a repeat of the last cumulative ACK.

E. Thread Safety and Invariants

The timer callback runs in a separate thread. All shared state and timer operations are guarded by an `RLock`. The implementation enforces

$$\texttt{timer\_active} \Leftrightarrow |\texttt{send\_buffer}| > 0,$$

ensuring consistent timer behaviour, and uses modular arithmetic with $S{=}2N$ so that

$$(\texttt{seq} - \texttt{base}) \bmod S < N$$

is unambiguous at wrap-around. Figure 2 summarizes the flow.

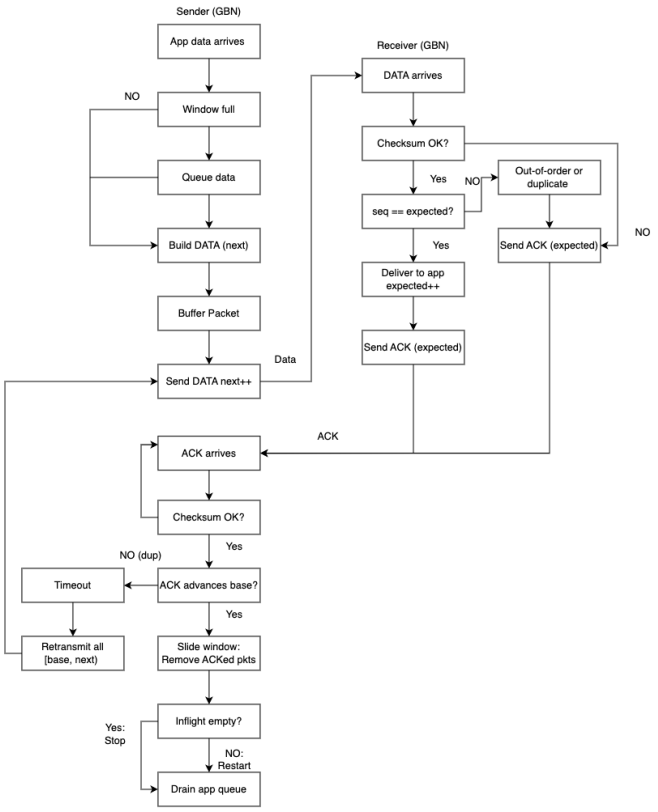

Fig. 2. Go-Back-N flow. Sender buffers unACKed DATA, runs a single timer at *base*, retransmits on timeout, and slides window on cumulative ACKs. Receiver delivers only in-order DATA and re-ACKs the next expected sequence.

IV. Experiments and Results

Sender-side counters: `sent`, `rtx` (retransmissions), `timeouts`, `dupACKs`, and `corrupt_d`.

PACKET_NUM	10
PACKET_SIZE	4
Window N	5
Seq space S	$2N$
Timeout	0.4 s
Seed	84737869
DROP/CORR/DELAY chance	0/0/0
DELAY_AMOUNT	0.5 s

TABLE I
BASELINE CONFIGURATION.

Scenario	Setting	Value
Drop 0.4	`DROP_CHANCE`	0.4
Delay 1.0×0.4 s	`DELAY_CHANCE`, `DELAY_AMOUNT`	1.0, 0.4 s
Drop+Corr+Delay	`DROP`, `CORR`, `DELAY`	0.4, 0.3, 0.3
	`Timeout`	1.2 s

TABLE II
SCENARIO-SPECIFIC OVERRIDES.

Scenario	sent	rtx	to	dupACK	corrD
No impairment	10	0	0	0	0
Drop 0.4	10	39	9	9	0
Delay 1.0×0.4 s	10	15	3	13	0
Drop 0.4, Corr 0.3, Delay 0.3	10	54	13	10	12

TABLE III
OBSERVED SENDER METRICS.

Findings. Table III confirms in-order, exactly-once delivery in all cases. Loss increases `timeouts` and `rtx` sharply, matching GBN's window replay on timeout. Pure delay produces duplicate ACKs but fewer retransmissions, while the larger timeout (1.2 s) in the mixed scenario reduces spurious timeouts but overall `rtx` remains high due to loss and corruption.

V. Discussion

A. Correctness and Progress

The receiver delivers only when `seqnum==expected`, discarding others and re-ACKing the next expected value, and the checksum blocks corrupted payloads. The single timer at `base` guarantees recovery under finite loss bursts by triggering window replay.

B. Window, Wrap-around, and Timeout

Using $S{=}2N$ keeps the window test $(\texttt{seq} - \texttt{base}) \bmod S < N$ unambiguous at wrap. The fixed 0.4 s timeout worked for the baseline but is suboptimal under varying delay, therefore, adaptive timeouts would cut delay-induced retransmissions.

C. Performance Trade-offs

GBN's simplicity comes at the cost of redundant retransmissions after a single loss. This is visible in `rtx` growth under loss. Increasing N raises throughput when RTT is large but increases the amount of data replayed on each timeout.

VI. Future Work

Selective Repeat with per-packet ACK and receiver buffering would avoid replaying correctly received frames. An adaptive timeout estimator and fast retransmit on duplicate-ACK thresholds would reduce recovery latency under moderate loss.

VII. ChatGPT Declaration

We used ChatGPT for language clarifications, structural advice, and a source finder. All technical content, implementation, and final revisions were done by the authors.

VIII. Conclusion

A Go-Back-N transport with one timer, cumulative ACKs, and a 16-bit Internet checksum achieves reliable, in-order delivery over lossy and delayed channels. Modular arithmetic with $S{=}2N$ and a strict timer/inflight invariant ensure correctness across wrap-around. Results match GBN theory and highlight its main trade-off, which is redundant retransmissions after loss.

References

[1] R. Braden, D. Borman, and C. Partridge. *Computing the Internet Checksum*. RFC 1071. 1988. URL: https://www.rfc-editor.org/rfc/rfc1071.

[2] GeeksforGeeks. *Open Systems Interconnection Model (OSI)*. Last Accessed 2025-10-22. 2025-09-19. URL: https://www.geeksforgeeks.org/computer-networks/open-systems-interconnection-model-osi/.

[3] GeeksforGeeks. *Application Layer in OSI Model*. Last Accessed 2025-10-22. 2025-10-15. URL: https://www.geeksforgeeks.org/computer-networks/application-layer-in-osi-model/.

[4] GeeksforGeeks. *Transport Layer in OSI Model*. Last Accessed 2025-10-22. 2025-10-13. URL: https://www.geeksforgeeks.org/computer-networks/transport-layer-in-osi-model/.

[5] GeeksforGeeks. *Network Layer in OSI Model*. Last Accessed 2025-10-22. 2025-10-04. URL: https://www.geeksforgeeks.org/computer-networks/network-layer-in-osi-model/.

[6] GeeksforGeeks. *Sliding Window Protocol (Set 2) — Receiver Side*. Last Accessed 2025-10-22. 2025-10-01. URL: https://www.geeksforgeeks.org/computer-networks/sliding-window-protocol-set-2-receiver-side/.

[7] TechTarget. *Packet Loss*. Last Accessed 2025-10-22. 2021-08-12. URL: https://www.techtarget.com/searchnetworking/definition/packet-loss.

[8] *Packet Corruption (ACM Digital Library article PDF)*. Last Accessed 2025-10-22 via ACM Digital Library. 2017. DOI: 10.1145/3098822.3098849. URL: https://dl.acm.org/doi/pdf/10.1145/3098822.3098849.