Report on RDT and TCP Congestion Control Protocol Implementations

IMT2024003 Gutha Raj Vardhan

1 Introduction

This report describes the implementation of four networking protocols / algorithms:

- RDT 2.2 (Reliable Data Transfer with bit errors)
- RDT 3.0 (Reliable Data Transfer with bit errors and packet loss)
- TCP Tahoe Congestion Control
- TCP Reno Congestion Control

These implemented in Python and are designed to simulate the behaviour of reliable data transfer mechanisms and congestion control algorithms. The report covers the design choices, parameter selections. Here i have decided to group the RDT implementations together.

2 RDT Implementation

2.1 Design Overview

The Reliable Data Transfer (RDT) implementation follows a stop-and-wait protocol where the sender transmits a packet and waits for an acknowledgment before sending the next packet. Both RDT 2.2 and 3.0 are implemented within the same code skeleton with features for handling bit errors, packet corruption, and packet loss.

2.2 Key Components

2.2.1 Packet Structure

The implementation uses a packet class hierarchy with:

- Base `Packet` class containing sequence number, data payload, and checksum
- Specialized `ACK_packet` class for acknowledgments

2.2.2 Channel Simulation

The `channel` class simulates an unreliable network with:

- Packet corruption (40% probability)
- Packet loss (30% probability)
- Random transmission delays (0.1s to 1.3s)

2.2.3 End System Implementation

The implementation uses an abstract `endsystem` class with specialized `Sender` and `Receiver` subclasses to manage the state transitions and transmission parts.

2.3 RDT 2.2 & 3.0 Features

Both protocols are implemented in the same codebase, with the following key features:

2.3.1 RDT 2.2 Features:

- Handles bit errors using checksums
- Uses sequence numbers (0 and 1) for packet identification
- Retransmits upon receiving corrupted or incorrect ACKs

2.3.2 RDT 3.0 Additional Features:

- Timeout mechanism to handle packet loss
- Retransmission of packets upon timeout
- Handling of lost ACKs

The implementation uses a fsm approach, with standard states like:

- "waiting for call 0 from above"
- "waiting for ACK 0"
- "waiting for call 1 from above"
- "waiting for ACK 1"

2.4 Parameter Choices

For the RDT implementation, several key parameters were selected to ensure robust testing of the protocols:

- **Timeout Interval (2.0s)**: Set higher than the maximum delay (1.3s) to avoid premature timeouts while still ensuring reasonable detection time for lost packets.
- **Checksum Algorithm**: The Internet Checksum method was selected as it provides reliable error detection for the types of corruption being simulated.
- **Sequence Numbers**: Alternating 0 and 1 is sufficient for a stop-and-wait protocol as there is never more than one unacknowledged packet in the channel.

2.5 Premature Timeout in RDT 3.0

As mentioned, the case of premature time out is ignored. If the ACK arrives after the timeout.. it is basically ignored. This is done so as to prevent duplication of packets during the transmission.

3 TCP Congestion Control

3.1 TCP Tahoe Implementation

The TCP Tahoe implementation simulates congestion control with two phases:

1. Slow Start
2. Congestion Avoidance

3.1.1 Design Overview

The implementation models TCP Tahoe's congestion window evolution over multiple round-trip times (RTTs), with:

- Slow start phase (exponential growth)
- Congestion avoidance phase (linear growth)
- Congestion window reset to 1 upon loss
- Threshold adjustment after loss events

3.1.2 Key Parameters

The implementation includes several configurable parameters:

- **cong_win_size**: Current congestion window size (initialized to 1)
- **threshold**: Slow start threshold (user-configurable)
- **total_rtt**: Total simulation duration in RTTs (user-configurable)
- **loss_events**: RTTs at which to simulate packet loss (user-configurable in manual mode)

3.1.3 Simulation Modes

The implementation supports two loss event modes:

1. **Random**: Approximately 10% chance of packet loss in each RTT
2. **Manually configured**: User-specified RTTs at which loss occurs

3.1.4 Algorithm Implementation

The Tahoe algorithm is implemented as follows:

1. Start with congestion window size of 1
2. During slow start (when `cong_win_size < threshold`):
    - Double the congestion window each RTT
3. During congestion avoidance (when `cong_win_size >= threshold`):
    - Increase congestion window by 1 each RTT
4. Upon packet loss:
    - Set threshold to half of current congestion window
    - Reset congestion window to 1
    - Restart with slow start

3.2 TCP Reno Implementation

The TCP Reno implementation extends Tahoe with fast recovery, implementing four phases:

1. Slow Start
2. Congestion Avoidance
3. Fast Recovery In case of Triple Duplicate ACKs

3.2.1 Key Differences from Tahoe

TCP Reno differs from Tahoe primarily in its response to packet loss:

- Upon triple duplicate ACK (indicating packet loss):
  - Set threshold to half of current congestion window
  - Skip the Slow start phase and start from the congestion avoidance phase.
- Upon timeout:
  - Set threshold to half of current congestion window
  - Reset congestion window to 1
  - Enter slow start phase

3.3 Graphical Results

3.3.1 TCP Tahoe Simulation

The implementation produces plots showing the evolution of the congestion window over time. A typical TCP Tahoe plot shows:

- Initial exponential growth during slow start
- Linear growth during congestion avoidance
- Sharp drops to 1 upon packet loss
- Gradually increasing threshold as the simulation progresses

3.3.2 TCP Reno Simulation

TCP Reno plots typically show:

- Similar exponential and linear growth patterns to Tahoe
- Less severe drops upon congestion (to threshold rather than to 1)
- Faster recovery from packet loss
- Smoother overall throughput compared to Tahoe

4 Imgaes and output files

The zip file contains the corresponding images and output files, named appropriately.

5 Dependencies

The TCP tahoe and reno implementations use the library `mathplotlib`, and hence it needs to be installed. It can be installed using the command `pip install mathplotlib`

6 Conclusion

This report has described the implementation of four networking protocols: RDT 2.2, RDT 3.0, TCP Tahoe, and TCP Reno. The implementations successfully demonstrate the core principles of reliable data transfer and congestion control.