Report on STP Protocol Implementation

Introduction

This report describes the design and implementation of a Simple Transport Protocol (STP) for reliable data transfer over an unreliable network. The sender and receiver programs are implemented in Python and utilize UDP sockets for communication.

Language and Code Organization

Both sender and receiver programs are written in Python, leveraging its standard libraries for socket programming and random number generation. The code is organized into different modules for better readability and maintainability:

- **Window.py**: Manages the sending window for the STP protocol.
- **clock.py**: Implements the timer functionality used for retransmission and timeout handling.
- **Segment.py**: Provides functions for creating, parsing, and manipulating STP segments.
- **Main Programs**: Contains the main logic for sender-side and receiver-side STP protocol implementations.

Both programs use command-line arguments to configure various parameters such as ports, file paths, maximum window size (Max_win), Retransmission Timeout (rto), Packet Loss Probability (flp), and Acknowledgment Loss Probability (rlp).

Implementation Details

Overall Program Design

Sender

1. **Connection Establishment**:
    - The sender sends a SYN segment to the receiver.
    - The sender waits for a SYN-ACK segment from the receiver.
    - Upon receiving the SYN-ACK, the sender sends an ACK segment to complete the connection establishment.
2. **Data Transfer**:
    - The sender splits the file data into segments.
    - It maintains a sending window to keep track of sent but unacknowledged segments.
    - The sender sends segments within the window, handles timeouts, and retransmits lost segments if necessary.
    - The sender also handles duplicate acknowledgments (DupACKs) to trigger fast retransmission.
3. **Connection Termination**:
    - After sending all data segments, the sender sends a FIN segment to initiate connection termination.
    - The sender waits for a FIN-ACK segment from the receiver to confirm the termination.

Receiver

1. **Connection Establishment**:
    - The receiver waits for a SYN segment from the sender.
    - Upon receiving the SYN, the receiver sends a SYN-ACK segment.

    - The receiver waits for an ACK from the sender to complete the connection establishment.
2. **Data Reception**:
    - The receiver waits for data segments from the sender.
    - It acknowledges received segments and handles out-of-order segments using cumulative acknowledgments.
    - The receiver discards duplicate segments and sends acknowledgments for correctly received segments.
3. **Connection Termination**:
    - The receiver waits for a FIN segment from the sender to initiate connection termination.
    - The receiver sends a FIN-ACK segment to acknowledge the FIN.
    - After sending the FIN-ACK, the receiver waits for an ACK from the sender to confirm the termination.

Data Structure Design

- **Sending Window**: Managed by the **Window.py** module, it maintains a buffer of sent but unacknowledged segments. The window size is dynamically adjusted based on acknowledgments received.
- **Timer**: Implemented in **clock.py**, the timer is used to handle timeouts and retransmissions.
- **STP Segment**: Defined in **Segment.py**, it encapsulates the STP header and data. Functions are provided to create, parse, and manipulate segments.

Operation

Sender

The sender initiates the connection by sending a SYN segment and waits for the SYN-ACK from the receiver. Once the connection is established, it sends the data segments, manages the sending window, handles timeouts, retransmissions, and terminates the connection.

Receiver

The receiver waits for the connection request from the sender, acknowledges received segments, handles out-of-order segments, discards duplicates, and terminates the connection upon receiving the FIN segment from the sender.

Design Trade-offs

- **Timeout Value**: The Rto value affects the reliability and efficiency of the data transfer. A shorter timeout can lead to unnecessary retransmissions, while a longer timeout can increase the overall transfer time.
- **Window Size**: The Max_win parameter determines the maximum number of unacknowledged segments that can be sent. A larger window size can increase throughput but may also lead to congestion and packet loss.
- **Loss Probabilities**: The flp and rlp parameters are used to simulate packet loss. Adjusting these values can affect the protocol's resilience to network errors.

Code Attribution

No segments of code have been borrowed from external sources or books.