Sender Module Documentation

Overview

The `sender.py` module is responsible for transmitting a file in fixed-size packets over a simulated network. It ensures reliable delivery using a sliding window protocol with acknowledgments (ACKs), retransmissions, and timeouts. The sender sends packets to a network simulator, which introduces loss, reordering, and corruption for testing.

Features

- Reads a file and splits it into fixed-size packets.
- Implements a sliding window mechanism to manage packet flow.
- Handles acknowledgments from the receiver.
- Detects and retransmits lost packets.
- Sends an end-of-transmission signal upon completion.

Classes

`Sender`

The `Sender` class manages packet transmission, acknowledgment handling, and retransmissions.

Constructor

- Sender(sender_socket, simulator_address, file_path)

- `sender_socket`: The UDP socket used for communication.
- `simulator_address`: Tuple containing the IP and port of the network simulator.
- `file_path`: Path to the file being transmitted.

Methods

`send_packet(seq_num, data)`

Sends a packet with the given sequence number and data to the network simulator. The packet is stored in a window for potential retransmission.

`start()`

Begins reading the file, splitting it into packets, and sending them using a sliding window approach. Manages acknowledgments and retransmissions.

`send_end_of_transmission()`

Sends a special end-of-transmission signal to indicate the completion of file transfer.

`receive_acks()`

Listens for acknowledgement packets from the receiver and updates the sliding window accordingly.

`handle_timeouts()`

Retransmits packets that have exceeded the timeout threshold.

Main Execution

The script accepts a file path as a command-line argument and initiates the sender process.

Usage

- python sender.py <file_path>

Example

- python sender.py sample.txt

Network Configuration

- Sends packets to the network simulator at `localhost:9999`.
- Receives acknowledgments on port `9997`.

Notes

- The sender uses `SIMULATED_LATENCY` to introduce a small delay between packets.
- `WINDOW_SIZE` controls the number of unacknowledged packets in flight.

- `TIMEOUT` determines when unacknowledged packets should be retransmitted.

Dependencies

- Python 3
- `socket`, `threading`, `time`, `struct`, `sys`, `os`
- `utils.py` (for utility functions like `make_packet`)

Receiver Module Documentation

Overview

The `receiver.py` module is responsible for receiving file packets sent by the sender through a network simulator. It ensures correct file reconstruction by handling out-of-order packets, detecting corrupted packets, and acknowledging received data.

Features

- Listens for incoming packets from the network simulator.
- Detects and discards corrupted packets.
- Buffers out-of-order packets until missing packets arrive.
- Sends acknowledgments (ACKs) for received packets.
- Writes the complete file to disk after receiving all packets.

Classes

`Receiver`

The `Receiver` class handles packet reception, buffering, acknowledgment, and file reconstruction.

Constructor

Receiver(receiver_socket, simulator_address, output_file)

- `receiver_socket`: The UDP socket used for communication.
- `simulator_address`: Tuple containing the IP and port of the network simulator.
- `output_file`: Path where the received file will be stored.

Methods

`start()`

Begins receiving packets, handling out-of-order arrivals, and writing data to disk when complete.

Main Execution

The script accepts an output file name as a command-line argument and starts the receiver process.

Usage

python receiver.py <output_file>

Example

python receiver.py received.txt

Network Configuration

- Listens for packets on port `9998`.
- Sends acknowledgements to the network simulator at `localhost:9999`.

Notes

- Uses `utils.py` for packet parsing and corruption detection.
- Packets are stored in a buffer if received out of order.
- The receiver will not overwrite an existing file.
- The special sequence number `99999999` signals the end of transmission.

Dependencies

- Python 3
- `socket`, `struct`, `sys`, `os`
- `utils.py` (for utility functions like `parse_packet` and `is_corrupted`)

Network Simulator Module Documentation

Overview

The `network_simulator.py` module simulates network impairments such as packet loss, corruption, and reordering to test the reliability of the sender and receiver implementation. It acts as an intermediary between the sender and receiver, forwarding packets with a probability of introducing faults.

Features

- Listens for packets from both sender and receiver.
- Simulates real-world network conditions by:
    - Dropping packets with a probability (`LOSS_PROBABILITY`).
    - Corrupting packets with a probability (`CORRUPTION_PROBABILITY`).
    - Reordering packets with a probability (`REORDER_PROBABILITY`).
- Forwards packets between sender and receiver after applying impairments.

Classes

`NetworkSimulator`

The `NetworkSimulator` class handles receiving, modifying, and forwarding packets.

Constructor
NetworkSimulator(listen_address, forward_address)

- `listen_address`: Tuple specifying the IP and port where the simulator listens for packets.
- `forward_address`: Tuple specifying the IP and port where packets should be forwarded (receiver's address).

Methods

`start()`

Continuously listens for incoming packets, applies impairments, and forwards them to the appropriate destination.

Main Execution

The script sets up a network simulator that listens on a specific port and forwards packets to a specified destination.

Usage

python network_simulator.py

Network Configuration

- Listens on `localhost:9999` for packets from sender and receiver.
- Forwards packets to `localhost:9998` (receiver's address).

Notes

- Uses probabilities defined in `utils.py` to introduce packet loss, corruption, and reordering.
- Corruption is introduced by flipping bits in the first byte of the packet.
- Reordered packets are delayed before forwarding.

Dependencies

- Python 3
- `socket`, `random`, `time`
- `utils.py` (for constants like `PACKET_SIZE`, `LOSS_PROBABILITY`, etc.)

Utils Module Documentation

Overview

The `utils.py` module provides utility functions and constants used across the sender, receiver, and network simulator. It includes packet creation, parsing, and integrity verification functions.

Constants

- `PACKET_SIZE = 1024` → The maximum size of a packet.
- `WINDOW_SIZE = 4` → The sender's sliding window size.
- `TIMEOUT = 2` → Timeout interval in seconds.
- `LOSS_PROBABILITY = 0.1` → Probability of packet loss in the network simulator.
- `CORRUPTION_PROBABILITY = 0.1` → Probability of packet corruption.
- `REORDER_PROBABILITY = 0.1` → Probability of packet reordering.
- `SIMULATED_LATENCY = 0.5` → Artificial delay introduced between packet transmissions.

Functions

`compute_checksum(data)`

def compute_checksum(data):

Computes a simple checksum by summing the bytes of the data and taking the modulo 256.

- **Parameters:**
  - `data (bytes)`: The packet payload.
- **Returns:** `int` → The computed checksum.

`make_packet(seq_num, data)`

def make_packet(seq_num, data):

Creates a packet with a sequence number, checksum, and data.

- **Parameters:**
  - `seq_num (int)`: The sequence number of the packet.

    - `data (bytes)`: The data to be sent.
- **Returns:** `bytes` → The constructed packet.

`parse_packet(packet)`

def parse_packet(packet):

Parses a received packet, extracting the sequence number, checksum, and data. Handles truncated packets and special end-of-transmission signals.

- **Parameters:**
    - `packet (bytes)`: The received packet.
- **Returns:**
    - `seq_num (int)`: Extracted sequence number.
    - `checksum (int)`: Extracted checksum.
    - `data (bytes)`: Packet payload.

`is_corrupted(packet)`

def is_corrupted(packet):

Checks if a packet is corrupted by comparing its computed checksum with the received checksum.

- **Parameters:**
    - `packet (bytes)`: The received packet.
- **Returns:** `bool` → `True` if the packet is corrupted, `False` otherwise.

Dependencies

- `struct` (for packing and unpacking packet data)
- Constants are used by `sender.py`, `receiver.py`, and `network_simulator.py`.

Launcher Module Documentation

Overview

The `launcher.py` script serves as an entry point for automating the execution of the sender, receiver, and network simulator. It prompts the user for input and output file names, validates them, and starts the required processes.

Functionality

1. Prompts the user to enter the file they want to upload.
2. Validates the existence of the input file.
3. Prompts the user for the output file name and ensures it does not already exist.
4. Starts the network simulator as a background process.
5. Launches the receiver process, specifying the output file.
6. Launches the sender process, specifying the input file.
7. Waits for the sender and receiver to complete before terminating the network simulator.
8. Displays a confirmation message when the file transfer is complete.

Functions

`main()`

def main():

The main function orchestrates the file transfer process by:

- Asking the user for input and output file names.
- Validating file paths.
- Running the sender, receiver, and network simulator.
- Waiting for the sender and receiver to finish.
- Terminating the network simulator.

Dependencies

- `os` (for file existence checks)
- `subprocess` (for process management)

Execution

To run the launcher:

python launcher.py

The script will guide the user through the process of selecting files and starting the necessary components automatically.