Implemented programming language and code organization

I used the Python programming language to implement the sender program of this STP protocol. The code is organized into a single Python script and adopts the object-oriented programming paradigm. The entire program is divided into several parts:

1.Import the necessary modules and libraries: At the beginning of the program, I imported the necessary modules and libraries, including socket, sys, threading, etc. These modules are used to implement functions such as network communication and multi-threading.

2.Define STP segment class: I defined a class named STP_segment to represent the data segment in the STP protocol. This class contains information such as the segment type, sequence number, and data, and provides methods to convert segments into byte representations and parse segments out of byte representations.

3.Define the control block class: I defined a data class named Control to store the control parameters of the program, including host name, port number, socket, etc.

4.Utility functions: I implemented some utility functions, such as parsing command line parameters, setting sockets, parsing runtime, etc.

5.Receiving thread and timer thread: I defined the functions of receiving thread and timer thread. The receiving thread is used to receive ACK and FIN segments from the receiving end, and the timer thread is used to terminate the program after timeout.

6.Send ISN and receive ISN: I implemented the function of sending ISN and receiving ISN to establish a connection.

7.Sending data: I implemented the function of sending data, including segmenting data, sending data segments, processing lost data segments, etc.

8. Main function: In the main function, I parse the command line arguments, set up the sender and receiver sockets, and then start sending data, handling timeouts and retransmissions if necessary.

Implementation of STP protocol

I implemented the sender program of the STP protocol. The program first sends ISN to establish the connection, then sends the data segment, processes the received ACK segment and retransmits the lost data segment, and finally sends FIN to close the connection.

Data structure design: I used a queue to store the received ACK sequence number, lost data segment, received data segment and other information. These queues provide convenience in handling retransmissions, losses, etc.

Programming: The entire program is designed as a multi-threaded program, including receiving threads, timer threads and main threads. The receiving thread is responsible for receiving the ACK and FIN segments from the receiving end. The timer thread terminates the program after timeout. The main thread is responsible for sending data and processing ACK, retransmission, etc.

Design trade-offs: During the design process, I considered a variety of factors, such as the program's reliability, performance, implementation difficulty, etc. For example, I decided to use multi-threading to handle receiving and sending to improve the concurrency and performance of the program. I also considered parameters such as retransmission mechanism and timeout settings to ensure the reliability of data transmission.