The Reliable Data Transfer (RDT) Lab is a computer networking project that involves implementing a simple reliable data transfer protocol using Python. The focus is on developing a basic version of a protocol that ensures reliable communication over an unreliable channel. This project will provide hands-on experience in dealing with issues like packet loss, corruption, and reordering, and implementing techniques like checksums, acknowledgments, and retransmissions for reliable communication.

Project Title: Reliable Data Transfer (RDT) Lab in Python

Project Description:

Overview:

The goal of this project is to implement a basic reliable data transfer protocol that ensures the accurate and ordered delivery of data between a sender and a receiver, despite potential issues like packet loss, corruption, and reordering.

Components:

1. **Sender:**
   - The sender is responsible for sending data to the receiver.
   - Divides the data into packets and sends them to the receiver.
   - Implements techniques for ensuring reliability, such as checksums and acknowledgments.
2. **Receiver:**
   - The receiver is responsible for receiving data from the sender.
   - Checks for errors in received packets using checksums.
   - Sends acknowledgments to the sender to confirm successful receipt.
3. **Unreliable Channel:**
   - Simulate an unreliable channel between the sender and receiver.
   - Introduce potential issues like packet loss, corruption, and reordering.

Requirements:

1. **Programming Language:**
   - Use Python for both the sender and receiver implementations.
2. **Packetization:**
   - Implement the division of data into packets at the sender.
3. **Checksums:**
   - Include checksums in each packet to detect errors.
4. **Acknowledgments:**
   - Implement acknowledgments at the receiver to confirm successful receipt.
5. **Retransmissions:**
   - Implement retransmissions at the sender for packets not acknowledged in a timely manner.

Sample Workflow:

1. The sender runs the sender script and provides the data to be sent.
2. The sender divides the data into packets and sends them to the receiver.
3. The receiver runs the receiver script and receives the packets.
4. The receiver checks for errors in the received packets using checksums.
5. The receiver sends acknowledgments to the sender for successfully received packets.
6. If the sender doesn't receive acknowledgments for specific packets within a timeout period, it retransmits those packets.
7. The sender and receiver continue this process until all data is successfully transmitted and acknowledged.

Additional Features (Optional):

- **Selective Repeat or Go-Back-N:** Implement a selective repeat or go-back-N mechanism for more efficient retransmissions.
- **Dynamic Adjustment of Timeout:** Implement dynamic adjustment of the timeout period based on the network conditions.
- **Error Correction (Optional):** Extend the project to include error correction mechanisms like Forward Error Correction (FEC).

Note:

Before starting the implementation, ensure you have a solid understanding of the principles of reliable data transfer protocols, including concepts like packetization, checksums, acknowledgments, and retransmissions. Consider thoroughly testing your implementation with various network conditions and scenarios to ensure its reliability. Additionally, explore Python libraries or modules that can facilitate the development of network protocols, such as `socket` for low-level socket programming.