# Project

This project is an all-inclusive implementation experience of major topics in Computer Networks, i.e., end-to-end protocol design and implementation: including error control, and the sliding window protocol.

The User Datagram Protocol (UDP) provides point-to-point, unreliable datagram service between a pair of hosts. It does not provide any reliability or sequencing guarantees – i.e. packets may arrive late, may not arrive at all, or arrive corrupted.

Your project is to implement a reliable and sequenced message transport protocol on top of this unreliable UDP. Your protocol will ensure reliable, end-to-end delivery of messages in the face of packet loss and will preserve ordering of messages.

Whereas TCP allows fully bidirectional communication, this implementation will be asymmetric. The two endpoints will be designated the "sender" and the "receiver" respectively. Data packets will only flow in the "forward" direction from the sender to the receiver, while ACKs will only flow in the "reverse" direction from the receiver back to the sender.

(Project inspired by my project assignment at UMN and more recently by Stefan Savage, UC San Diego.)

##### Implementation Nodes:

You will implement the client and server component of a transport layer. The client reads a stream of data (from a file), breaks it into fixed-sized packets suitable for UDP transport, prepends a control header to the data, and sends each packet to the server. The server reads these packets and writes the corresponding data, in order, to a reliable stream (a file).

A high-level overview of the system:

**File 🡪 Sender 🡨-- UDP -- 🡪 Receiver 🡪 File**

In general, since you are using UDP, the implementation (both at the receiver and sender) should:

* The receiver’s output should be identical to the other side's input, regardless of a **lossy**, **congested**, or **corrupting** network layer. You will ensure reliable transport by having the recipient acknowledge packets received from the sender; the sender will detect missing acknowledgements and resend the dropped or corrupted packets.
* Handle packet drops/delays  
  The sender should resend a packet if the receiver does not acknowledge it within an appropriate time period. You resend packet(s) whenever a sent packet has gone unacknowledged for the timeout period. The default timeout period is 2000 msec, but you may change this with the -t command-line option.
* Handle packet reordering  
  Your server and client should ensure that data is written in the correct order, even if the network layer reorders packets.
* Detect and handle packet corruption using the Internet checksum algorithm
* Provide trivial flow control – based on the send window. You should support arbitrary window sizes. The window size may be supplied by the -w command-line option.
* Allow multiple packets to be outstanding at any time (using a limit given to your program as a run-time parameter, via the -w option)

File formats that must transfer over can be of any type, e.g., .txt, .doc, .mov, .jpg, .exe, etc.

##### Testing considerations:

Because we are doing this between two processes on a single machine, latency is near 0, and no network faults can occur. As a result, you should introduce errors to test the correctness of your implementation.

The user of your program must have the ability to force packets to be lost, to be arbitrarily delayed at both processes, or arrive with checksum errors (i.e., both the sender’s packet and the receiver’s ACK may be lost/delayed/corrupt).

You will need to deal with:

1. multiple data packets in flight.
2. Retransmission of packets which have timed out.
3. Sequence number wrap-around.   
   Your senders should choose an ISN at random, and should deal with the problem of the sequence number wrapping around to zero.

##### User Interface

Design a simple (text-based or graphical) UI for your program. The user should be prompted for parameter values for the selected protocol simulation. For example, the user should be prompted for either default values (determined by the designer) or user-defined values to be entered.

At a minimum, prompt for:

1. Size of packet
2. Timeout interval
3. Size of sliding window
4. Range of sequence numbers
5. Situational Errors: none, randomly generated, or user-specified (drop packets 2, 4, 5; lose ACKs 11, 12 etc.)

##### Program output:

Your UI design should provide enough information when you demo your project.

You will be required to open two windows, one for the client and one for the server.

Both windows should present enough information to demonstrate the protocol in action.

The information provided in the **sender’s window** should include:

* Packet sequence number of packet sent.
* Packet (sequence number) that timed out
* Packet that was re-transmitted.
* All packets in the current window.
* Acks received.
* Which packets are damaged, i.e., deliberately trigger a checksum error on the receiver side.

The information provided in the **receiver’s window** should include:

* Packets (sequence number) received.
* Damaged packet(s) (checksum error did not match)
* corresponding damaged packet from sender window).
* Packets in the current receiver window.
* Packets (duplicated) that are discarded.
* ACK packets that are sent.

The receiver must re-sequence any out of order frames received.

##### Implementation Details

**Packet Types and Fields**

There are two kinds of packets, Data packets and Ack-only packets. You can tell the type of a packet by its length.

public class Packet {

short cksum; //16-bit 2-byte

short len; //16-bit 2-byte

int ackno; //32-bit 4-byte

int seqno ; //32-bit 4-byte Data packet Only

char data[500]; //0-500 bytes. Data packet only. Variable

}

* **cksum**: 2 byte IP checksum.
* **len**: 2 byte total length of the packet.
  + For Ack packets this is 8: 2 for cksum, 2 for len, and 4 for ACK no
  + For data packets, this is 12 + payload size: 2 for cksum, 2 for len, 4 for ackno, 4 for seqno, and as many bytes are there in data[]   
    An end-of-file condition is transmitted to the other side of a connection by a data packet containing 0 bytes of payload, and hence a len of 12.   
      
    Note: You must examine the length field, and should not assume that the UDP packet you receive is the correct length. The network might truncate or pad packets.
* **ackno**: 4-byte cumulative acknowledgment number.   
  ackno is the sequence number you are waiting for, that you have not received yet – it is the equivalent of Next Frame Expected.   
  This says that the sender of a packet has received all packets with sequence numbers earlier than ackno, and is waiting for the packet with a seqno of ackno.   
  The first sequence number in any connection is 1, so if you have not received any packets yet, you should set ackno to 1.

The following fields only exist in a data packet:

* **seqno**: Each packet transmitted in a stream of data must be numbered with a seqno. The first packet in a stream has a seqno of 1. Note that in TCP, sequence numbers indicate bytes. By contrast, this protocol just numbers packets. That means that once a packet is transmitted, it cannot be merged with another packet for retransmission. This should simplify your implementation.
* **data**: Contains (len - 12) bytes of payload data for the application.   
  To conserve packets, a sender should not send more than one unacknowledged Data frame with less than the maximum number of bytes (500)

##### Testing

Test for interoperability with other students in the class.

##### Technical paper:

Document your design and implementation in a technical paper. Describe and justify your design decisions. In your paper, you should discuss the pros and cons of the following design issues:

1. the size of your packet.
2. Frame format, i.e., format of the header and payload
3. Sequence numbers
4. The size of your sliding window.
5. The time-out interval (static or dynamically determined).
6. How to implement timeouts.

You will be asked to demo your project. Hence, it is important to capture as much information as possible of your run.

There should be a separate area where all the necessary information is logged and printed on the screen.

## Grading Criteria

Technical paper: 10%

Implementation:

* stop and wait: 40%
* sliding window: 50%
  + resends after timeout: 10%
  + handling reordered packets: 10%
  + handling corrupt packets: 10%
  + handling delayed packets: 10%
  + handling lost packets: 20%

**You will lose points:**

if you do not demonstrate your project to me: 30% penalty

if you submit past the due date: 30% penalty

if you demonstrate/submit your project after the last class meeting (before finals week): 100% penalty