# Implement Data link layer protocols

* **Stop and Wait,**
* **Go Back N SlidingWindow**

## Go Back N

### Sender

It maintains a window size, with a low and high pointer. After a sending all frames in that window it waits for ack for TTL time. If TTL expires it resends the frames

while 1:  
 x = open(tempfile, 'rb')  
 fcntl.flock(x, fcntl.LOCK\_EX )  
 status = pickle.load(x)  
 fcntl.flock(x, fcntl.LOCK\_UN)  
 # print(status)  
 if status['transfer\_complete']:  
 print( "File transfer Complete ..")  
 print( "Total time in seconds : " + str(time.time() - start\_time))  
 break  
  
 print('Trying to send {}'.format(seq\_no\_to\_send))  
 message\_to\_send = pack\_data(seq\_no\_to\_send, frames[seq\_no\_to\_send])  
 client\_socket.sendto(message\_to\_send,(server\_host,port))  
 # set\_timeout(send\_seq\_no,tempfile)  
 seq\_no\_to\_send += 1  
 if seq\_no\_to\_send > status['window\_high']:  
 seq\_no\_to\_send = status['window\_low']

In another **thread** it receives the ack. If ack is found for a seq no, the window slides forwards.

def rcv\_thread(soc):  
 global window\_size  
 global total\_frames  
 global tempfile  
 new\_proc = os.fork()  
 if new\_proc == 0:  
 print("Receive Thread " + str(os.getpid()) + " created..")  
 while 1:  
 while os.stat(tempfile).st\_size == 0:  
 continue  
  
 status = pickle.load(open(tempfile, 'rb'))  
 print(status)  
 message, server\_addr = soc.recvfrom(max\_buff)  
 message = pickle.loads(message)  
 seq\_no, ack = message['seq\_num'], message['ack']  
 print('\033[93mAck Rec {} value {}\033[0m'.format(seq\_no, ack))  
 if seq\_no == total\_frames-1:  
 status['transfer\_complete'] = True  
 break  
 if ack:  
 status['window\_low'] = seq\_no+1  
 status['window\_high'] = status['window\_low'] + window\_size  
 x = open(tempfile, 'wb')  
 fcntl.flock(x, fcntl.LOCK\_EX )  
 pickle.dump(status, open(tempfile, 'wb') )  
 fcntl.flock(x, fcntl.LOCK\_UN)

### Receiver

Waits for frames and when found sends an ack if **valid** to the sender with the seq no received

while 1:  
 msg, client\_address = s.recvfrom(65535)  
 data = pickle.loads(msg)  
 got\_seq\_no, frame = data['seq\_num'], data['frame']  
 print('Packet Recv seq\_no {} '.format(got\_seq\_no))  
 if random() < probability: # Random packet drop simulation  
 decoded = crc.decode([frame], verbose=True)  
 if decoded is not None: # Checksum is correct  
 if got\_seq\_no == exp\_seq\_no: # Send ack  
 to\_send = rdt\_send(got\_seq\_no)  
 if to\_send:  
 s.sendto(to\_send, client\_address)  
 f\_write(decoded,output\_file)  
 exp\_seq\_no = exp\_seq\_no + 1  
 elif got\_seq\_no > exp\_seq\_no: # Future packet received, hence dropped  
 print ("Packet loss, sequence number = " + str(got\_seq\_no))  
 elif got\_seq\_no < exp\_seq\_no: # Repeat sent  
 to\_send = rdt\_send(exp\_seq\_no)  
 if to\_send:  
 # print ("Retransmitted ACK - " + str(got\_seq\_no))  
 s.sendto(to\_send, client\_address)  
 else:  
 print ("Checksum invalid. Packet dropped.")  
 else: # Random packet drop simulation  
 print ("Packet loss, sequence number = " + str(got\_seq\_no))