```
#############################################
#                                           #
# Filename    : sw_receiver.py              #
# Description : Binds to a Stop and Wait server socket. For    #
#               Packets received, it sends back an acknowledg- #
#               ment. Inorder to simulate an noisy communcation #
#               channel, Packets successfully are arrived are  #
#               dropped with some pre-determined probability.  #
#                                           #
#############################################

import socket
import sys
import random
import logging

from common import *


logging.basicConfig(level=logging.DEBUG,
                format='%(asctime)s.%(msecs)-03d : %(message)s',
                datefmt='%H:%M:%S')

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('', int(sys.argv[1]) if len(sys.argv) >= 2 else 3300))
recvd_data = []
expected_seq_no = 0


def to_network_layer(frame):
    recvd_data.append(frame)


while True:
    try:
        pack = recv_packet(sock)
        if pack.seq_no != expected_seq_no:
            logging.info("[ERR]  : %s arrived out of order." % pack)
            ack = Packet(expected_seq_no, ptype=Packet.TYPE_ACK)
            send_packet(sock, ack)
        else:
            # No need to replicate 'packet not arrived' scenorio.
            if not pack.is_corrupt():
                expected_seq_no = 1 - expected_seq_no
                to_network_layer(pack.data)
                logging.info("[RECV] : %s" % pack)
                send_packet(sock,
                    Packet(expected_seq_no, Packet.TYPE_ACK))
                logging.info('[ACK]  : ack_no = %d' % expected_seq_no)
            else:
```

```
                    # Simply drop the packet.
                logging.debug("[CHKSERR] : Dropping %s" % pack)
        except ConnectionResetError:
            break
        except KeyboardInterrupt:
            break

logging.info("Closing connection..")
sock.close()
logging.info("Data received = %s", ''.join(recvd_data))
sys.exit(0)
```

4.2.4 Output

```
$ python3 sw_sender.py
Enter a message : hello
21:17:28.608 : [SENT]    : Packet(seq_no=0, data=h)
21:17:28.694 : [ACK]     : Packet(ack_no=1)
21:17:29.696 : [SENT]    : Packet(seq_no=1, data=e)
21:17:29.826 : [ACK]     : Packet(ack_no=0)
21:17:30.828 : [SENT]    : Packet(seq_no=0, data=l)
21:17:30.958 : [ACK]     : Packet(ack_no=1)
21:17:31.959 : [SENT]    : Packet(seq_no=1, data=l)
21:17:32.90  : [CHKSERR] : Received corrupted ACK packet.
                           Sending Packet(seq_no=1, data=l) again.
21:17:32.178 : [ACK]     : Packet(ack_no=0)
21:17:33.179 : [SENT]    : Packet(seq_no=0, data=o)
21:17:33.310 : [ACK]     : Packet(ack_no=1)
21:17:34.311 : Closing chanel..

$ python3 sw_receiver.py
21:17:28.608 : [RECV] : Packet(seq_no=0, data=h)
21:17:28.608 : [ACK]  : ack_no = 1
21:17:28.650 : [ERR]  : Packet(seq_no=0, data=h) arrived out of order.
21:17:29.738 : [RECV] : Packet(seq_no=1, data=e)
21:17:29.739 : [ACK]  : ack_no = 0
21:17:29.782 : [ERR]  : Packet(seq_no=1, data=e) arrived out of order.
21:17:30.870 : [RECV] : Packet(seq_no=0, data=l)
21:17:30.871 : [ACK]  : ack_no = 1
21:17:30.914 : [ERR]  : Packet(seq_no=0, data=l) arrived out of order.
21:17:32.2   : [RECV] : Packet(seq_no=1, data=l)
21:17:32.2   : [ACK]  : ack_no = 0
21:17:32.46  : [ERR]  : Packet(seq_no=1, data=l) arrived out of order.
21:17:32.134 : [ERR]  : Packet(seq_no=1, data=l) arrived out of order.
21:17:33.222 : [RECV] : Packet(seq_no=0, data=o)
21:17:33.223 : [ACK]  : ack_no = 1
21:17:33.266 : [ERR]  : Packet(seq_no=0, data=o) arrived out of order.
21:17:36.167 : Closing connection..
21:17:36.167 : Data received = hello
```

4.3 GoBack N Protocol

4.3.1 Theory

To improve the efficiency of transmission (filling the pipe), multiple frames must be in transition while waiting for acknowledgment. In other words, we need to let more than one frame be outstanding to keep the channel busy while the sender is waiting for acknowledgment. In this section, we discuss one protocol that can achieve this goal. The first is called Go-Back-N Automatic Repeat Request. In this protocol we can send several frames before receiving acknowledgments; we keep a copy of these frames until the acknowledg- ments arrive.

Frames from a sending station are numbered sequentially. However, because we need to include the sequence number of each frame in the header, we need to set a limit. If the header of the frame allows m bits for the sequence number, the sequence numbers range from 0 to 2 m - 1. For example, if m is 4, the only sequence numbers are 0 through 15 inclusive. However, we can repeat the sequence. So the sequence num- bers are 0,1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0, 1, 2, 3,4 ... In other words, the sequence numbers are modulo-2^m

In this protocol, the sliding window is an abstract concept that defines the range of sequence numbers that is the concern of the sender and receiver. The range which is the concern of the sender is called the send sliding window; the range that is the concern of the receiver is called the receive sliding window. The send window is an imaginary box covering the sequence numbers of the data frames which can be in transit. In each window position, some of these sequence numbers define the frames that have been sent; others define those that can be sent. The maximum size of the window is $2^m - 1$. The window at any time divides the possible sequence numbers into four regions. The first region, defines the sequencenumbers belonging to frames that are already acknowledged. The second region defines the range of sequence numbers belonging to the frames that are sent and have an unknown status. The sender needs to wait to find out if these frames have been received or were lost. We call these outstanding frames. The third range, defines the range of sequence numbers for frames that can be sent; however, the corresponding data packets have not yet been received from the network layer. Finally, the fourth region defines sequence numbers that cannot be used until the window slides, as we see next.

The send window can slide one or more slots when a valid acknowledgment arrives. The receive window is an abstract concept defining an imaginary box of size 1 with one single variable Rn. The window slides when a correct frame has arrived; sliding occurs one slot at a time.

4.3.2 Algorithm

Algorithm 8 GoBack-N Protocol - Sender

```
1: S_w ← 2^m − 1
2: S_f = S_n = 0
3: while True do
4:     WaitForEvent()
5:     if Event(RequestToSend) then
6:         if S_n − S_f ≥ S_w then
7:             Sleep()
8:         end if
9:         GetData()
10:        MakeFrame(S_n)
11:        StoreFrame(S_n)
12:        SendFrame(S_n)
13:        S_n ← (S_n + 1)%S_w
14:        if Timer is not running then
15:            StartTimer()
16:        end if
17:    end if
18:    if Event(ArrivalNotification) then
19:        Receive(ACK)
20:        if Corrupted(ACK) then
21:            Sleep()
22:        end if
23:        if ackNo > S_f and ackNo <= S_n then
24:            while S_f ≤ ackNo do
25:                PurgeFrame(S_n)
26:                S_f ← (S_f + 1)%S_w
27:            end while
28:        end if
29:        StopTimer()
30:    end if
31:    if Event(Timeout) then
32:        StartTimer()
33:        temp ← S_f
34:        while temp < S_n do
35:            SendFrame(S_n)
36:            S_f ← (S_f + 1)%S_w
37:        end while
38:    end if
39: end while
```

Algorithm 9 GoBack-N Receiver

```
1: R_n ← 0
2: while True do
3:     WaitForEvent()
4:     if Event(ArrivalNotification) then
5:         Receive(frame)
6:         if Corrupted(frame) then
7:             Sleep()
8:         end if
9:         if seqNo == R_n then
10:            DeliverData()
11:            R_n ← (R_n + 1)%2^m
12:        end if
13:        SendACK(R_n)
14:    end if
15: end while
```

4.3.3 Program

```
#############################################
#                                           #
# Filename    : gbn_sender.py               #
# Description : A typical sender implementation for Go-Back N  #
#               protocol. Note that only a single timer is used #
#               and that too for the first outstanding packet.  #
#               If the timer timeouts, all the outstanding      #
#               packets are send again.                         #
#                                           #
#############################################

import socket
import sys
import os
from threading import *
from common import *
from time import time, sleep
from math import floor
from collections import deque
import logging


sock   = None
pbuffer = deque([], maxlen=GBN_WINDOW_SIZE)
S_f, S_n = 0, 0
message, msglen = '', 0

# Time to wait for an acknowledgement.
ACK_WAIT_TIME = 3000

# Configure logging
logging.basicConfig(level=logging.DEBUG,
                    format='%(asctime)s.%(msecs)-03d : %(message)s',
                    datefmt='%H:%M:%S')


class basic_timer:

    def __init__(self):
        self.start_time = None

    def start(self, interval):
        self.start_time = basic_timer.current_time_in_millis()
        self.interval = interval

    def has_timeout_occured(self):
        cur_time = basic_timer.current_time_in_millis()
        return cur_time - self.start_time > self.interval
```

```
    def is_running(self):
        return self.start_time != None

    def stop(self):
        self.start_time = None
        self.interval = None

    def restart(self, interval):
        self.start(interval)

    @staticmethod
    def current_time_in_millis():
        return int(floor(time() * 1000))


def outstanding_frames():
    return len(pbuffer)


def is_valid_ackno(ack_no):
    if outstanding_frames() <= 0: return False
    t = (S_f + 1) % (MAX_SEQ_NO + 1)
    while t != S_n:
        if t == ack_no: return True
        t = (t + 1) % (MAX_SEQ_NO + 1)
    return ack_no == S_n


def main():
    global S_n, S_f, pbuffer

    timer = basic_timer()
    next_msg_index = 0
    while 1:
        while outstanding_frames() < GBN_WINDOW_SIZE and \
                                    next_msg_index < msglen:
            # There is space in buffer
            pack = Packet(S_n, data=message[next_msg_index])
            send_packet(client, pack)
            logging.info('[SEND]     : Sending %s.' % pack)
            pbuffer.append(pack)
            S_n = (S_n + 1) % (MAX_SEQ_NO + 1)
            if not timer.is_running():
                timer.start(ACK_WAIT_TIME)
            next_msg_index += 1

        sleep(.7)
        resp = recv_packet_nblock(client)

        if resp is not None and not resp.is_corrupt():
            if not is_valid_ackno(resp.seq_no):
                logging.info('[EACK]     : Invalid ACK %s.' % resp)
```

```
            else:
                # Remove packets from buffer
                tmp = []
                while len(pbuffer) > 0 and \
                        pbuffer[0].seq_no != resp.seq_no:
                    tmp.append(str(pbuffer.popleft().seq_no))
                    S_f = (S_f + 1) % (MAX_SEQ_NO + 1)
                logging.info(('[ACK]      : Ack received %s.' +\
                'Packets(%s) are acknowledged.') % \
                (resp, ','.join(tmp)))

        sleep(.8)

        if timer.has_timeout_occured():
            for p in pbuffer:
                logging.info('[TIMEOUT] : Resending %s.' % p)
                send_packet(client, p)
            timer.start(ACK_WAIT_TIME)

        if outstanding_frames() == 0 and next_msg_index >= msglen:
            logging.info('Transfer complete.')
            break
        # else
        sleep(1)


if __name__ == '__main__':

    message = input('Enter a message : ')
    msglen = len(message)
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.bind(('', 3300 if len(sys.argv) <= 1 else int(sys.argv[1])))
    sock.listen(5)
    client, _addr = sock.accept()
    main()
    sock.close()
    client.close()
```

```
#########################################
#                                       #
# Filename    : gbn_receiver.py         #
# Description : A typical receiver for Go-Back N Protocol.  #
#               It remembers the sequence number of the next  #
#               packet in order and only accept that one. In  #
#               other words, packets arriving out of order are #
#               simply dropped. For each packet received and  #
#               and found to be not corrupted, an acknowledge- #
#               ment is sent to the sender with the sequence no.#
#               of the next expected packet.                  #
#                                       #
#########################################

import socket
import sys
import random
import logging


from common import *


logging.basicConfig(level=logging.DEBUG,
                    format='%(asctime)s.%(msecs)-03d  %(message)s',
                    datefmt='%H:%M:%S')

# Socket for listening for incoming connections
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('', 3300 if len(sys.argv) <= 1 else int(sys.argv[1])))
logging.debug('Connected..')
expected_seq_no = 0

data_recvd = []

while True:
    try:
        # Wait for packet
        pkt = recv_packet(sock)
        if pkt.is_corrupt():
            continue
        if pkt.seq_no == expected_seq_no:
            logging.info('[RECV]   : Received %s.' % pkt)
            expected_seq_no = (expected_seq_no + 1) % (MAX_SEQ_NO + 1)
            data_recvd.append(pkt.data)
        else:
            logging.info('[ERR]    : %s arrived out of order.' % pkt)
        ack_pkt = Packet(expected_seq_no, ptype=Packet.TYPE_ACK)
        logging.info('[ACK]    : %s' % ack_pkt)
        send_packet(sock, ack_pkt)
    except socket.error as e:
```

```
            logging.error(str(e))
            break
        except KeyboardInterrupt as e:
            break

logging.info('Transfer complete. Data received = \"%s\"' % ''.join())
sock.close()

sys.exit(0)
```

4.3.4 Output

```
$ python3 gbn_sender.py
Enter a message : world
21:22:45.629 : [SEND]    : Sending Packet(seq_no=0, data=w).
21:22:45.629 : [SEND]    : Sending Packet(seq_no=1, data=o).
21:22:45.629 : [SEND]    : Sending Packet(seq_no=2, data=r).
21:22:45.630 : [SEND]    : Sending Packet(seq_no=3, data=l).
21:22:45.630 : [SEND]    : Sending Packet(seq_no=4, data=d).
21:22:46.331 : [EACK]    : Invalid ACK Packet(ack_no=0).
21:22:48.834 : [EACK]    : Invalid ACK Packet(ack_no=0).
21:22:49.635 : [TIMEOUT] : Resending Packet(seq_no=0, data=w).
21:22:49.636 : [TIMEOUT] : Resending Packet(seq_no=1, data=o).
21:22:49.636 : [TIMEOUT] : Resending Packet(seq_no=2, data=r).
21:22:49.636 : [TIMEOUT] : Resending Packet(seq_no=3, data=l).
21:22:49.636 : [TIMEOUT] : Resending Packet(seq_no=4, data=d).
21:22:51.338 : [EACK]    : Invalid ACK Packet(ack_no=0).
21:22:53.841 : [ACK]     : Ack received Packet(ack_no=1).
                           Packets (0) are acknowledged.
21:22:54.643 : [TIMEOUT] : Resending Packet(seq_no=1, data=o).
21:22:54.643 : [TIMEOUT] : Resending Packet(seq_no=2, data=r).
21:22:54.643 : [TIMEOUT] : Resending Packet(seq_no=3, data=l).
21:22:54.643 : [TIMEOUT] : Resending Packet(seq_no=4, data=d).
21:22:56.345 : [ACK]     : Ack received Packet(ack_no=2).
                           Packets (1) are acknowledged.
21:22:58.849 : [ACK]     : Ack received Packet(ack_no=3).
                           Packets (2) are acknowledged.
21:22:59.650 : [TIMEOUT] : Resending Packet(seq_no=3, data=l).
21:22:59.650 : [TIMEOUT] : Resending Packet(seq_no=4, data=d).
21:23:01.353 : [ACK]     : Ack received Packet(ack_no=4).
                           Packets (3) are acknowledged.
21:23:03.856 : [ACK]     : Ack received Packet(ack_no=5).
                           Packets (4) are acknowledged.
21:23:04.657 : Transfer complete.


$ python3 gbn_receiver.py
21:22:45.629  Connected..
21:22:45.629  [ERR]   : Packet(seq_no=1, data=o) arrived out of order.
21:22:45.629  [ACK]   : Packet(ack_no=0)
21:22:45.630  [ERR]   : Packet(seq_no=3, data=l) arrived out of order.
21:22:45.630  [ACK]   : Packet(ack_no=0)
```

```
21:22:45.630  [ERR]   : Packet(seq_no=4, data=d) arrived out of order.
21:22:45.630  [ACK]   : Packet(ack_no=0)
21:22:49.678  [RECV]  : Received Packet(seq_no=0, data=w).
21:22:49.678  [ACK]   : Packet(ack_no=1)
21:22:49.679  [RECV]  : Received Packet(seq_no=1, data=o).
21:22:49.679  [ACK]   : Packet(ack_no=2)
21:22:49.679  [RECV]  : Received Packet(seq_no=2, data=r).
21:22:49.679  [ACK]   : Packet(ack_no=3)
21:22:49.680  [RECV]  : Received Packet(seq_no=3, data=l).
21:22:49.680  [ACK]   : Packet(ack_no=4)
21:22:49.680  [RECV]  : Received Packet(seq_no=4, data=d).
21:22:49.680  [ACK]   : Packet(ack_no=5)
21:22:54.686  [ERR]   : Packet(seq_no=1, data=o) arrived out of order.
21:22:54.686  [ACK]   : Packet(ack_no=5)
21:22:54.687  [ERR]   : Packet(seq_no=3, data=l) arrived out of order.
21:22:54.687  [ACK]   : Packet(ack_no=5)
21:22:54.687  [ERR]   : Packet(seq_no=4, data=d) arrived out of order.
21:22:54.687  [ACK]   : Packet(ack_no=5)
21:22:59.694  [ERR]   : Packet(seq_no=3, data=l) arrived out of order.
21:22:59.694  [ACK]   : Packet(ack_no=5)
21:22:59.695  [ERR]   : Packet(seq_no=4, data=d) arrived out of order.
21:22:59.695  [ACK]   : Packet(ack_no=5)
21:23:04.658  Transfer complete. Data received = "world"
```