NOVA SCHOOL OF SCIENCE & TECHNOLOGY
DEPARTAMENTO DE INFORMÁTICA

Reliable Data Transmission over an Unreliable Network
(TPC3 – Redes de Computadores)

This assignment concerns the problem of delivering information reliably across a network where a degree of packet loss is expected. In the assignment, two programs cooperate in the reliable transfer of a file, using UDP datagrams, in an environment where data and acknowledgement datagrams can be (and will be) lost. The application is composed by two processes:

- **Sender**: sends the contents of the file in blocks of fixed size to a receiver process located in another machine
- **Receiver**: receives the blocks of the file acknowledging each block received

Sender and receiver handle the loss of data and acknowledge using a sliding window of dimension N and Go-Back-N strategy

The format and purpose of each datagram is as follows:

DATA

|0|seqN| data|

Represents a block of file data. The block is identified by a sequence number ('seqN'), starting at 1,

for the first block of a file.

data - the file block payload encoded as raw bytes. The payload varies between 0 and 1024 bytes.

ACK

|1|cSeqN|

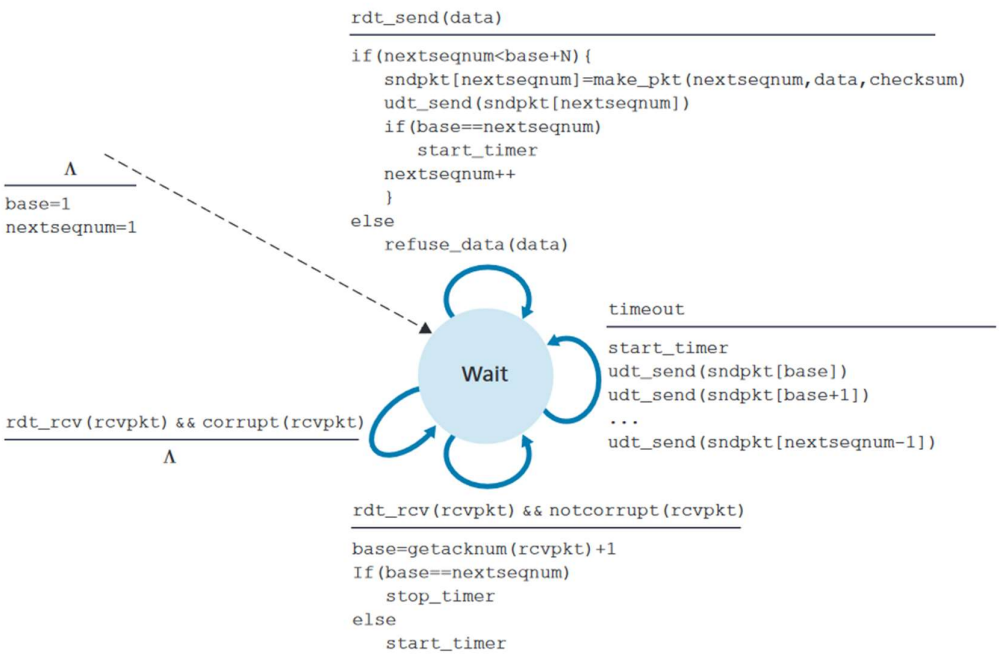
Confirms correctly received packets. *cSeqN* represents a cumulative sequence number that acknowledges all packets up to and including the given value.

To build messages, you can use the *pickle* package as in homework 2.

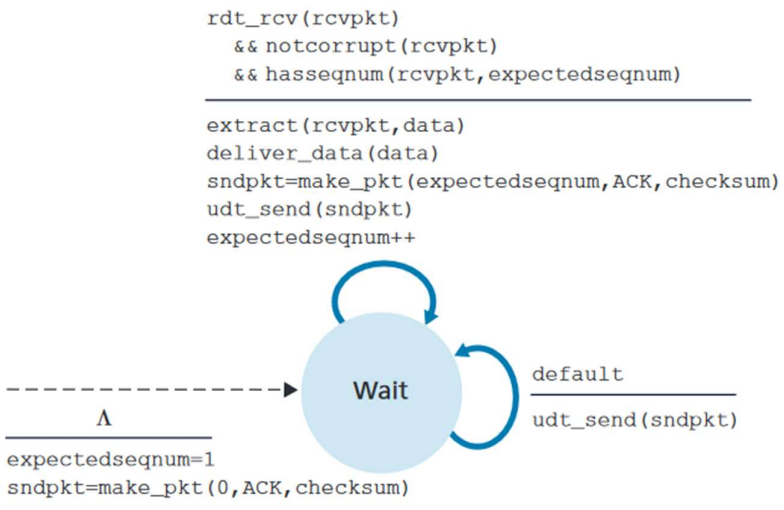
2. State diagrams of sender and receiver using Go-Back-N

The following figures, extracted from the book *Computer Networking: a Top-Down Approach* de J. Kurose and K. Ross, specify the behavior of the sender and receiver transferring a sequence of messages using a Sliding Window N:1 with GBN Protocol used for datagram loss recovery.

Sender State Diagram



Receiver State Diagram



Two notes are relevant when adapting these two state diagrams to this assignment:

- There is no packet corruption.
- The two processes transfer one file and then terminat. It will be necessary to handle the end of the file at both sender and receiver.

As you most certainly know, it is possible to produce a program that behaves according to specification made in the state diagram:

```
main()
	execute actions in the edge conducting to the initial state
	state = INITIAL_STATE
	while state != FINAL_STATE:
		match state:
			case STATE_1:
				# consider all the edges departing from STATE_1
				if condition in the first edge considered is true:
					execute actions described
					state = state where the edge terminates

				elif condition in the second link considered is true:
					execute action specified in the edge
					…
				else:
			case STATE_2:
				…
			case STATE_x:
				…
				if  …:
				…
					State = FINAL_STATE
```

3. Implementation

The task of the group is to write two Python programs called *sender.py* and *receiver.py*:

- Receiver is invoked by writing in the command line:

```
python receiver.py receiverIP receiverPort fileNameInReceiver
```

- After launching of receiver, *sender* is invoked by writing in the command line:

```
python sender.py senderIP senderPort receiverIP receiverPort filename
windowSizeInBlocks
```

- Both programs should terminate after guaranteeing that the file was transferred correctly.

Code should work if the sender and the receiver run in distinct machines. The best easy way to guarantee this goal is by using two containers over *Docker*.

Simulation of datagram loss

To simulate datagram loss, programs should call the following function, already used in homework 2. There is no other possibility of sending datagrams between *sender* and *receiver*.

Note that losses can occur in both directions of communication, from sender to receiver and from receiver to sender.

```
import random
…

def sendDatagram (msg, sock, address):
	# msg is a byte array ready to be sent
	# Generate random number in the range of 1 to 10
	rand = random.randint(1, 10)

	# If rand is less is than 3, do not respond (20% of loss probability)
	if rand >= 3:
		sock.sendto(msg, address)
```

Waiting for a datagram with timeout

We will again use code already present in homework 2.

```
import select
…

def waitForReply( uSocket, timeOutInSeconds ):
	rx, tx, er = select.select( [uSocket], [], [], timeOutInSeconds)

	# waits for data or timeout
	if rx == []:
		return False
	else:
		return True
```

Sliding window support

The sliding window can be implemented using a dictionary where the key is the block number and the information is the packet payload.

4. Delivery

The delivery should be done before 10:00 on November,7 2023. The submission has two parts:

- a Google form containing the identification of the students that submit the work and questions about the functionality of the code
- The code developed will be uploaded through Moodle

Details will be sent later.