# Redes de Computadores
## Computer Networks

Lab 7

Presentation and framework

for the mandatory 3rd Frequency Work Assignment

***Reliable Data Transmission over an Unreliable Network***

(TPC 3)

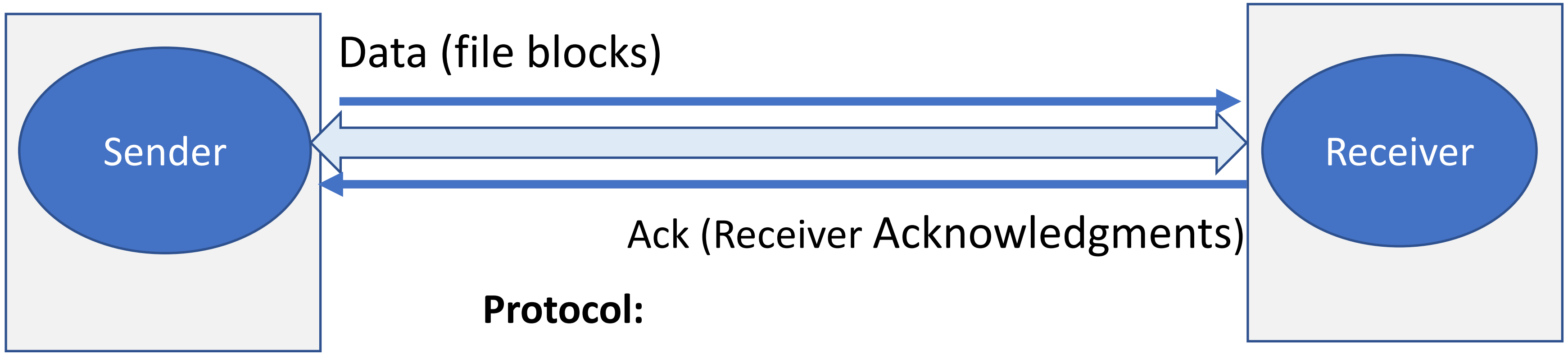
# Summary

- TPC-3 Context, Goal and Requirements
- Implementation Guidelines
  - Sliding Window and Go Back N (Review)
- Delivery process and deadline

# TPC-3 Context and Goals

**Context:** delivering information reliably across a network

- **Reliable file transfer using UDP datagrams**
- **Internetworking environment: data and ack datagrams can be (and will be) lost**
- **Implementation of a Sliding Window & Go-Back-N protocol model**




**Sender**:
Sends file contents in blocks of fixed block size

**Protocol:**

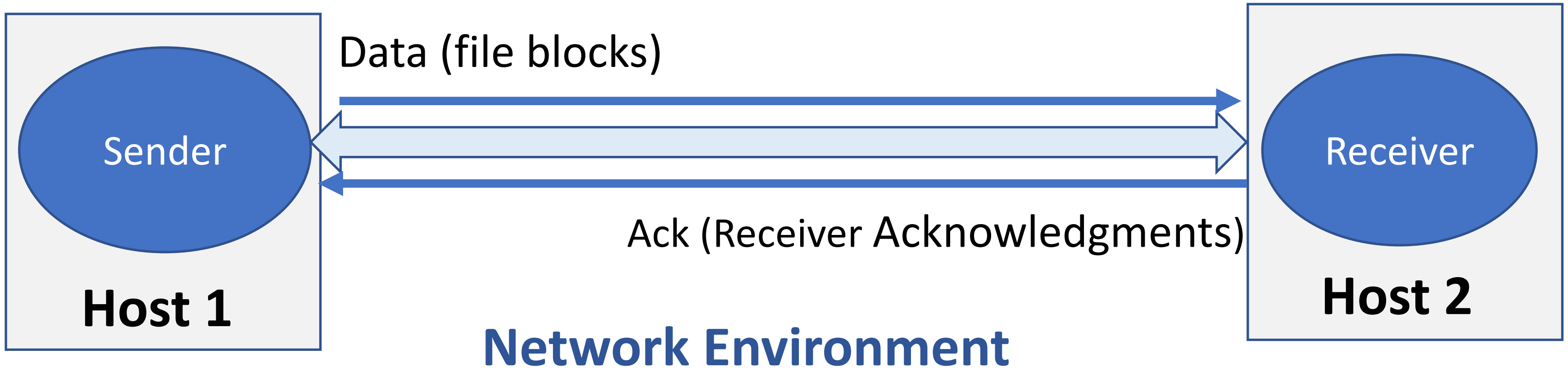
Reliable file transfer protocol using a sliding window of dimension N and a Go-Back-N strategy

**Receiver**:
Receives the blocks of the file acknowledging each received block

# TPC-3 Context and Goals

**Context:** delivering information reliably across a network

- **Reliable file transfer using UDP datagrams from a Sender** to **Receiver**
- **Internetworking environment: data and ack datagrams can be (and will be) lost**
- **Implementation of a Sliding Window & Go-Back-N protocol model**

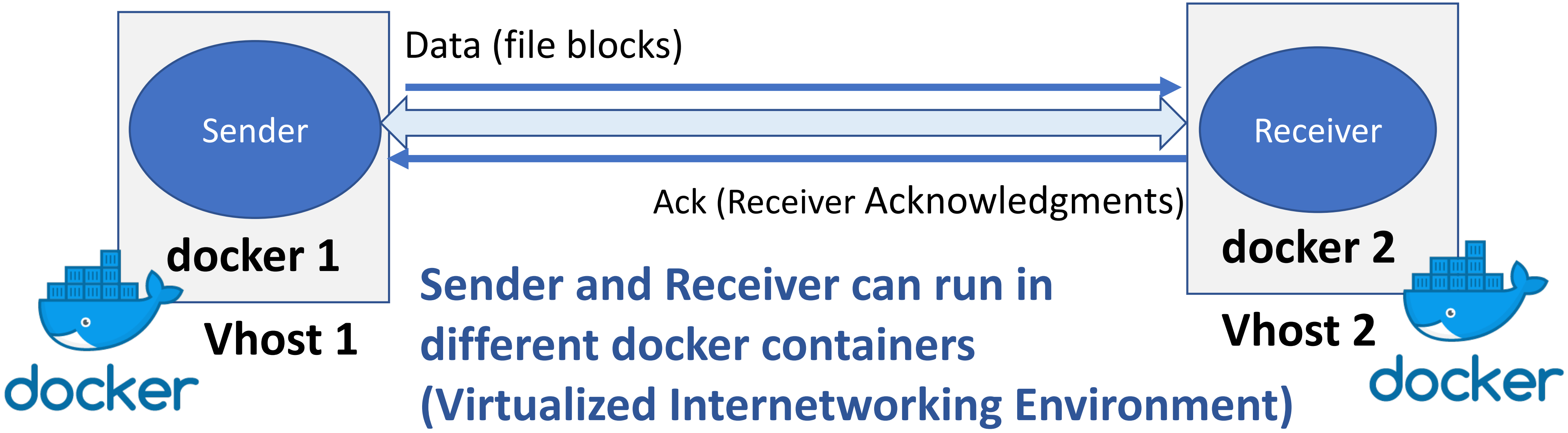



**Sender and Receiver can run in different hosts**
**(as well as processes in localhost)**

# TPC-3 Context and Goals

**Context:** delivering information reliably across a network

- **Reliable file transfer using UDP datagrams from a Sender** to **Receiver**
- **Internetworking environment: data and ack datagrams can be (and will be) lost**
- **Implementation of a Sliding Window & Go-Back-N protocol model**




***Note: context of Lab 5, Lab 6***

# Implementation

- Python
- Datagram Sockets
  - Pickle Package for Data Serialization of Datagram Payloads (Encoding/Decoding Protocol Message Formats)
- Protocol using a Sliding Window and Go Back N Approach
- Docker

# Requirements for the Implementation

Two python programs called *sender.py* and *receiver.py*:

- Receiver must be invoked in he following way (command line with arguments)

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

- After launching of receiver, sender must be invoked in the following way (command line and arguments)

```
python sender.py senderHostname senderPort receiverHostname receiverPort filename_receiver windowSizeInBlocks
```

- Both programs should terminate after guaranteeing that the file was transferred correctly.
- The code developed can be tested in a single machine (ex. localhost, 127.0.0.1)
- Code should work if the sender and the receiver run in distinct machines.
- Test in different machines or test building and running in two docker containers.

# Protocol message formats

**Must support reliable transfers of binary files !!!**

- Sender to receiver

**| 0 | seqN | data |**

**Data messages:**

**Blocks of file data**. identified by a **sequence number** (`***seqN***´), starting at 1, for the first block.

***data*** – file block payload encoded as raw bytes (**1024 bytes max**).

- Receiver to Sender

**| 1 | cSeqN |**

**ACK messages:**

Confirmations of correctly received packets.

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

***Note:*** To build such messages: use the **pickle package** as in TPC-2

# Sliding Window and GoBack N Model Analysis, review and discussion

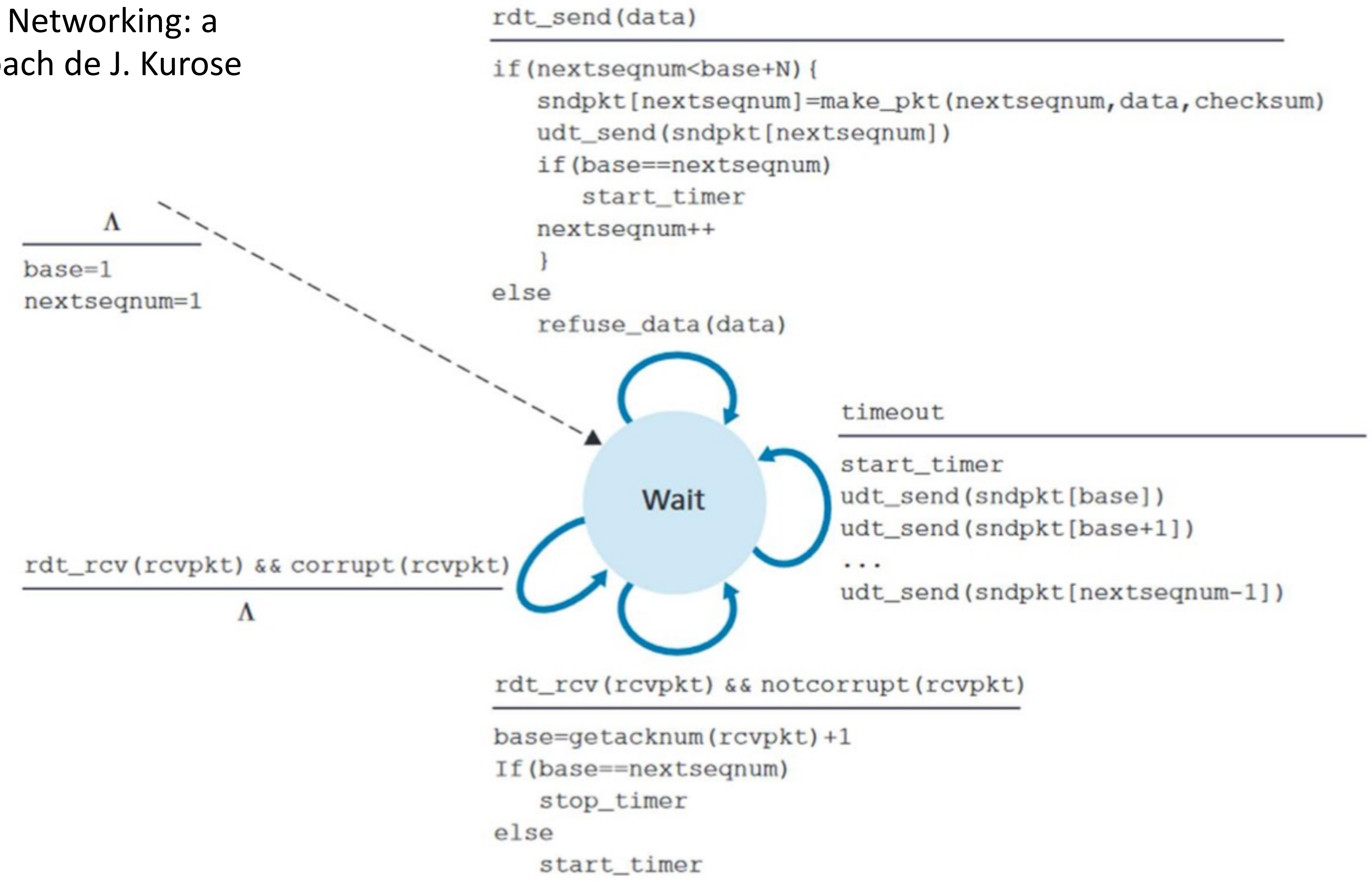
An animated representation for Go Back N *

- https://www2.tkn.tu-berlin.de/teaching/rn/animations/gbn_sr/

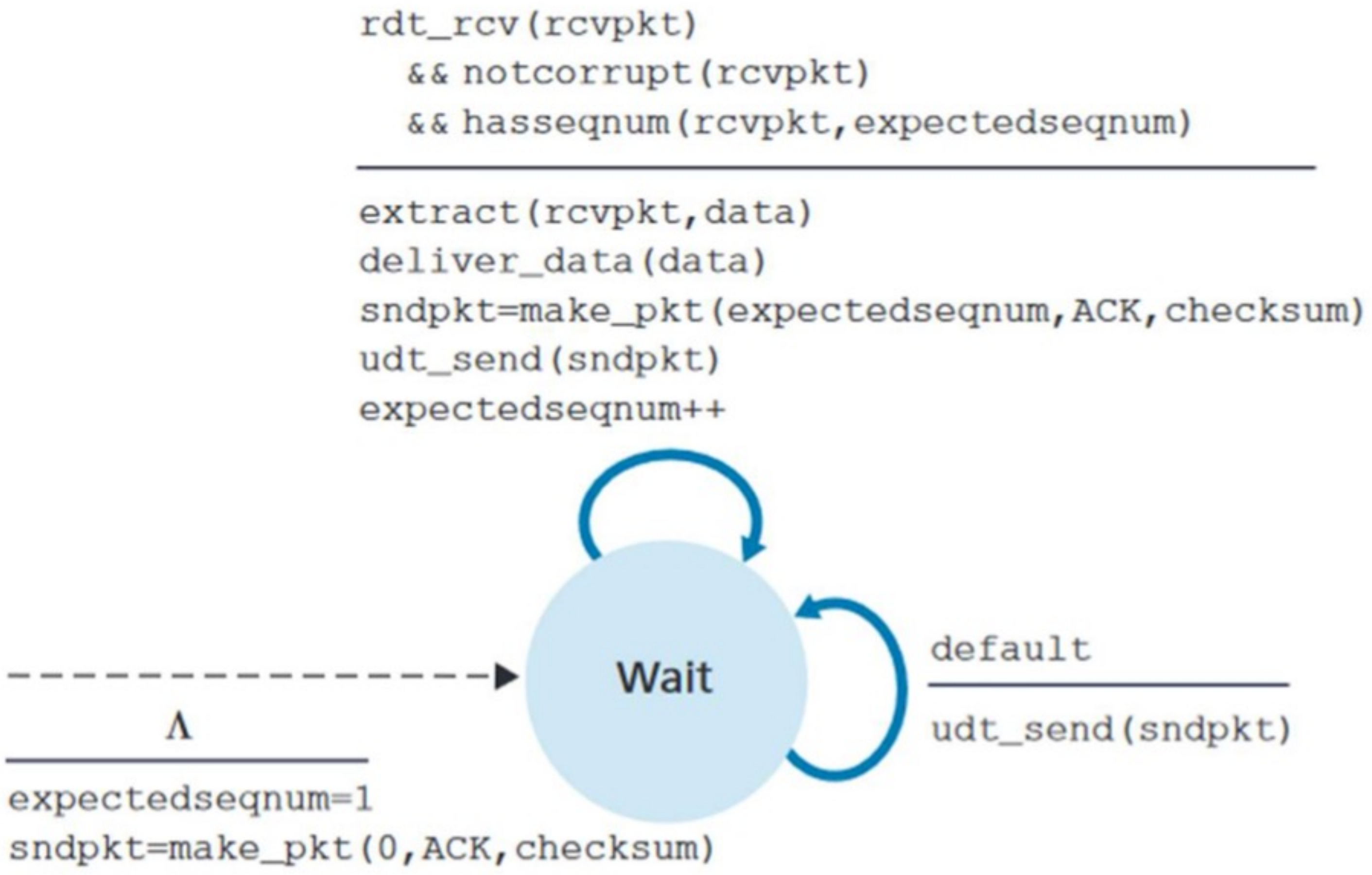
*) Also have the alternative Selective Repeat approach (which is not used for the context of TPC-3)

# Go Back N Synthetic Protocol Model: Sender State Machine Diagram

See in Computer Networking: a Top-Down Approach de J. Kurose and K. Ross



# Go Back N Synthetic Protocol Model: Receiver State Machine Diagram



# Important notes

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

# Program (Simple Pseudo-Code) implementing the State Machine Specification

```
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
```

# Datagram Loss (Loss Simulations)

- Simulation with the following function, already used in TPC-2.
- There is no other possibility of sending datagrams between *sender* and *receiver*.

```
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

- Again, using the code already present in TPC2

```
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
```

# Implementation strategy for the Sliding Window Protocol

Must support data structure for the notion of “Sliding Window”

Can implement using  a dictionary where …

- the Key is the block number
- the information is the packet payload.

# For your TPC-3 Work Delivery

- **The delivery should be done before 10:00 on November, 7 2023.**
- **Submission has two parts**:
  - The implementation **code** developed: **uploaded to Moodle**
  - A **Google Form** with the identification of the students/groups and questions about the functionality and tests of the code

Similar to TPC-2 delivery process.

Other details will be sent later.