EE2T21 Data Communications Networking - 2024
Bonus Assignment # 1

Names: Sjoerd Terlouw Student ID: 5852455

1 Abstract

Model of the S&W protocol in Python. The implementation consists of two functions simulation the sender and receiver. The simulation of corruption and keeping track of the position in the message is done by the main function. The corruption itself is simulated by a flag (1 or 0). Using this model several expirements done.

2 Code

The code is also provided seperately and is on github.

```python
import numpy as np
import matplotlib.pyplot as plt
# Path
import os
dir_path = os.path.dirname(os.path.realpath(__file__))

# IEEE plots
import scienceplots
plt.style.use(['science','ieee'])

# Progress bar
import tqdm

def corrupt(p, df):
    """
    Description
    -----------
    Corruption of a message with probability p

    Example
    -------
    >>> df = [1, 0, 72]
    >>> p = 1
    >>> print(corrupt(p, df))
    [0, 0, 72]

    >>> df = [1, 0, 72]
    >>> p = 0
    >>> print(corrupt(p, df))
    [1, 0, 72]
    """

    if np.random.random() < p:
        return [0, df[1], df[2]]
    else:
        return df

def main(n = 4,
         M_size = 4,
         p1 = 0.2,
         p2 = 0.2,
         linear = False):
    """
```

```
Description
-----------
Main function to loop through the message M
This message is chosen randomly with size M_size and the
 ↪  integers are in range [0, 2^n - 1]

If linear = True the message is linear, e.g. M = [1, 2, 3, 4,
 ↪  5]

Inputs
------
n : integer
    Range of M: [0, 2^n - 1]
M_size : integer
    Size of M
p1 : float
    Probability of an error occuring when transmitting a
     ↪  message from sender to receiver
p2 : float
    Probability of an error occuring when transmitting an
     ↪  acknowledgement from receiver to sender

Outputs
-------
R : np.array of integers
    Values that the receiver has received, -1 indicates that an
     ↪  errors has occured
M : np.array of integers
    Message
"""

if linear == False:
    M = np.random.randint(0, 2**n-1, size=M_size)
else:
    M = np.int32(np.linspace(0, 2**n-1, M_size))

R = []

ACK_frame = [1, 0, 0] # = [error, frame, message]
timer_value = 0
nS = 0
nR = 0
i = 0

while i < len(M):
```

```
        dataFrame, timer_value, nS, i = sender(ACK_frame =
         ↪  ACK_frame, timer_value = timer_value, nS = nS,
         ↪  next_integer = M[i], buffered_integer = M[i-1], i=i)
        dataFrame = corrupt(p1, dataFrame)
        ACK_frame, nR = receiver(arrived_frame = dataFrame, nR =
         ↪  nR)
        ACK_frame = corrupt(p2, ACK_frame)

        R.append(ACK_frame[2])
    return R, M

def sender(ACK_frame, timer_value, nS, next_integer,
 ↪  buffered_integer, i):
    """
    Description
    -----------
    Sender code
    Send a new message if no error has occured and the frames align
    Send previous message if this is not the case
    """
    timer_value += 1
    if (ACK_frame[0] == 1 and ACK_frame[1] == nS) or i == 0:
        # If there is no corruption and the n align, sent the next
         ↪  integer (with its corresponding n)
        # and update nS for the next integer
        timer_value = 0
        i += 1
        df = [1, nS, next_integer]
        nS = int(not(nS))
    else:
        # If there is corruption or the n do not align, sent the
         ↪  previous integer again (with its corresponding n)
        df = [1, int(not(nS)), buffered_integer]
    return df, timer_value, nS, i

def receiver(arrived_frame, nR):
    """
    Description
    -----------
    Receiver code
    Confirm if the frames align and no error has occured
    Otherwise simulate not sending anything by sending an error
    """
    if arrived_frame[0] == 1:
        if arrived_frame[1] == nR:
            # If there is no corruption and the n align, confirm
             ↪  this
```

```
            # and updata nR to expect the following frame
            nR = int(not(nR))
            return [1, nR, arrived_frame[2]], nR
        else:
            # If there is no corruption, but the n don't align,
             ↪  don't sent anything (or in this simplified case,
             ↪  sent that there is corruption)
            # and updata nR to expect the corrupted frame again
            nR = int(not(nR))
            return [0, nR, -1], nR
    else:
        # If there is corruption, don't sent anything (or in this
         ↪  simplified case, sent that there is corruption)
        return [0, nR, -1], nR

# Toggle experiments
give_an_example = True
do_experiment_1 = True
do_experiment_2 = True
do_experiment_3 = True

if give_an_example == True:
    print("Creating an example")

    n = 3
    p = 0.3

    fig, ax = plt.subplots(1, 2, figsize=(7,3))

    R, M = main(n=n, M_size = 2**n, p1 = p, p2 = 0, linear = True)
    ax[0].stem(np.linspace(1,len(R),len(R)), R)
    ax[0].set_title(f"Receiver for $p_1={p}$")
    ax[0].set_xlabel("Iteration")
    ax[0].set_ylabel("Value")

    R, M = main(n=n, M_size = 2**n, p1 = 0, p2 = p, linear = True)
    ax[1].stem(np.linspace(1,len(R),len(R)), R)
    ax[1].set_title(f"Receiver for $p_2={p}$")
    ax[1].set_xlabel("Iteration")
    ax[1].set_ylabel("Value")

    plt.suptitle("Received value at receiver for a linear message")
    plt.tight_layout()
    plt.savefig(os.path.join(dir_path, "example_SandW.svg"))

if do_experiment_1 == True:
    print("Running experiment 1:")
```

```python
    print("Number of loop iterations for different message sizes")

    n = 5
    loop_iterations = []
    M_sizes = np.arange(0,1000, 20)
    for M_size in tqdm.tqdm(M_sizes):
        R, M = main(n = n, M_size = M_size, p1 =0, p2 =0)
        loop_iterations.append(len(R))

    print("Plotting experiment 1")
    fig, ax = plt.subplots()

    ax.plot(M_sizes, loop_iterations)
    ax.set_title("Loop iterations necessary for a certain size
     ↪  without errors")
    ax.set_xlabel("Size $S$")
    ax.set_ylabel("Loop iterations $T$")
    plt.savefig(os.path.join(dir_path, "size_experiment.svg"))

if do_experiment_2 == True:
    print("Running experiment 2:")
    print("Number of loop iterations for different values of p1")

    n = 5
    M_size = 200
    p1_values = np.linspace(0,1,100, endpoint=False)
    loop_iterations_tot = []
    for i in tqdm.tqdm(range(0,10)):
        # Do experiment several times (10) and take average to deal
         ↪  with randomness
        loop_iterations = []
        for p1 in p1_values:
            R, M = main(n = n, M_size = M_size, p1 = p1, p2 = 0)
            loop_iterations.append(len(R))
        loop_iterations_tot.append(loop_iterations)
    loop_iterations_tot = np.array(loop_iterations_tot).T
    loop_iterations_averages = np.average(loop_iterations_tot,
     ↪  axis=1)

    print("Plotting experiment 2")
    fig, ax = plt.subplots()

    ax.plot(p1_values, loop_iterations_averages)
    ax.set_title("Averaged loop iterations for $S=200$ and varying
     ↪  $p_1$")
    ax.set_xlabel("$p_1$ values")
    ax.set_yscale('log')
```

```
    ax.set_ylabel("Loop iterations $T$")
    plt.savefig(os.path.join(dir_path, "p1_experiment.svg"))

if do_experiment_3 == True:
    print("Running experiment 3:")
    print("Number of loop iterations for different values of
     ↪  p1=p2=p")

    n = 5
    M_size = 100
    p_values = np.linspace(0,1,100, endpoint=False)
    loop_iterations_tot = []
    for i in tqdm.tqdm(range(0,10)):
        # Do experiment several times (10) and take average to deal
         ↪  with randomness
        loop_iterations = []
        for p in p_values:
            R, M = main(n = n, M_size = M_size, p1 = p, p2 = p)
            loop_iterations.append(len(R))
        loop_iterations_tot.append(loop_iterations)
    loop_iterations_tot = np.array(loop_iterations_tot).T
    loop_iterations_averages = np.average(loop_iterations_tot,
     ↪  axis=1)

    print("Plotting experiment 3")
    fig, ax = plt.subplots()

    ax.plot(p_values, loop_iterations_averages)
    ax.set_title("Averaged loop iterations for $S=100$ and varying
     ↪  $p_1=p_2=p$")
    ax.set_xlabel("$p$ values")
    ax.set_yscale('log')
    ax.set_ylabel("Loop iterations $T$")
    plt.savefig(os.path.join(dir_path, "p_experiment.svg"))
```