МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное автономное   
образовательное учреждение высшего образования  
«Самарский национальный исследовательский университет   
имени академика С.П. Королева»

(Самарский университет)

Институт информатики, математики и электроники

Факультет информатики  
Кафедра суперкомпьютеров и общей информатики

**Отчет по лабораторной работе №1**

Дисциплина: «Проектирование распределенных защищенных приложений»

Выполнил: Неженский М.С.

Группа: 6133-010402D

Самара 2023

**Задание**

1. Разработать клиент серверный механизм передачи цифрового изображения.

2. Учесть возможность возникновения помех в канале передачи.

3. Разработать блок восстановления полученных изображений на стороне сервера.

4. Для передачи данных между клиентом и сервером использовать сокеты Беркли.

5. Оценить потери данных при наличии блока восстановления ошибок и при его отсутствии.

**Код приложения**

**Класс FinalServer:**

import socket

import cv2

import numpy as np

import matplotlib.pyplot as plt

HOST = "127.0.0.1"

class FinalServer:

def \_\_init\_\_(self, server\_port, block\_size=1024):

self.server\_port = server\_port

self.block\_size = block\_size

self.socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

self.socket.bind((HOST, server\_port))

self.socket.listen()

def run(self):

with self.socket as s:

print(f'Ожидание соединения на порту {self.server\_port}.')

conn, addr = s.accept()

print(f'Соединение установлено. Порт:{self.server\_port}.')

# Получение изображения с наложенным шумом

self.img\_noise = self.recv\_image(conn)

# Получение оригинального изображения

self.img\_orig = self.recv\_image(conn)

# Получение изображения с шумоподавлением

self.img\_denoise = self.denoise(self.img\_noise)

def recv\_image(self, conn):

# Получение размеров изображения

h\_bytes = conn.recv(4)

h = int.from\_bytes(h\_bytes, byteorder='big')

w\_bytes = conn.recv(4)

w = int.from\_bytes(w\_bytes, byteorder='big')

print(f'Размер изображения h = {h} w = {w}.')

# Получение изображения

n = h \* w \* 3

n\_iters = n // self.block\_size

n\_remainder = n - n\_iters \* self.block\_size

img\_bytes = []

for i in range(n\_iters):

block = conn.recv(self.block\_size)

img\_bytes.append(block)

if n\_remainder > 0:

block = conn.recv(n\_remainder)

img\_bytes.append(block)

print('Изображение получено.')

img\_bytes\_b = b''.join(img\_bytes)

img\_np = np.frombuffer(img\_bytes\_b, dtype='uint8')

img = img\_np.reshape(h, w, 3)

return img

def denoise(self, image):

return cv2.medianBlur(image, ksize=5)

finalserver = FinalServer(server\_port=5066)

finalserver.run()

img\_orig = finalserver.img\_orig

img\_noise = finalserver.img\_noise

img\_denoise = finalserver.img\_denoise

# Просмотр изображений (оригинал)

print(img\_orig.shape)

plt.imshow(cv2.cvtColor(img\_orig, cv2.COLOR\_BGR2RGB))

# Просмотр изображения (с шумом)

print(img\_noise.shape)

plt.imshow(cv2.cvtColor(img\_noise, cv2.COLOR\_BGR2RGB))

# Просмотр изображений (с шумоподавлением)

print(img\_denoise.shape)

plt.imshow(cv2.cvtColor(img\_denoise, cv2.COLOR\_BGR2RGB))

def eval\_dataloss(img1, img2, img3):

mse = np.square((img1 - img2).astype('float32')).mean()

print('Среднеквадратическая ошибка для изображения с шумом:', mse)

mse = np.square((img1 - img3).astype('float32')).mean()

print('Среднеквадратическая ошибка для изображения с шумоподавлением:', mse)

eval\_dataloss(img\_orig,img\_noise,img\_denoise)

**Класс NoiseServer:**

import socket

import cv2

import numpy as np

import matplotlib.pyplot as plt

HOST = "127.0.0.1"

class NoiseServer:

def \_\_init\_\_(self, noise\_server\_port, final\_server\_port, block\_size=1024):

self.final\_server\_port = final\_server\_port

self.noise\_server\_port = noise\_server\_port

self.block\_size = block\_size

self.socket\_server = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

self.socket\_server.bind((HOST, noise\_server\_port))

self.socket\_server.listen()

# Sends image

self.socket\_client = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

def run(self):

# Получение оригинального изображения

img\_orig = self.recv\_image()

# Добавление шума на оригинальное изображение

img\_noise = self.add\_noise(img\_orig)

with self.socket\_client as s:

print(f'Waiting for connetion with port {self.final\_server\_port}.')

s.connect((HOST, self.final\_server\_port))

print(f'Connected. Port:{self.final\_server\_port}. Sending images.')

# Отправка изображения с шумом

self.send\_image(img\_noise, s)

# Отправка оригинального изображения

self.send\_image(img\_orig, s)

def recv\_image(self):

with self.socket\_server as s:

print(f'Waiting for connection on port {self.noise\_server\_port}.')

conn, addr = s.accept()

print(f'Connected. Port:{self.noise\_server\_port}.')

# Получение размера изображения

h\_bytes = conn.recv(4)

h = int.from\_bytes(h\_bytes, byteorder='big')

w\_bytes = conn.recv(4)

w = int.from\_bytes(w\_bytes, byteorder='big')

print(f'Размер изображения h = {h} w = {w}.')

# Получение изображения

n = h \* w \* 3

n\_iters = n // self.block\_size

n\_remainder = n - n\_iters \* self.block\_size

img\_bytes = []

for i in range(n\_iters):

block = conn.recv(self.block\_size)

img\_bytes.append(block)

if n\_remainder > 0:

block = conn.recv(n\_remainder)

img\_bytes.append(block)

print("Image was received.")

img\_bytes\_b = b''.join(img\_bytes)

img\_np = np.frombuffer(img\_bytes\_b, dtype='uint8')

img = img\_np.reshape(h, w, 3)

return img

def send\_image(self, img, s):

# Отправка размера изображения

h, w, \_ = img.shape

n = h \* w \* 3 # size of bytes array

h\_bytes = h.to\_bytes(length=4, byteorder='big')

w\_bytes = w.to\_bytes(length=4, byteorder='big')

img\_bytes = img.tobytes()

assert s.sendall(h\_bytes) is None, print("Ошибка отправки h\_bytes: %s", e)

assert s.sendall(w\_bytes) is None, print("Ошибка отправки w\_bytes: %s", e)

# Отправка изображения

n\_iters = n // self.block\_size

n\_remainder = n - n\_iters \* self.block\_size

for i in range(n\_iters):

block = img\_bytes[self.block\_size\*i:(i+1)\*self.block\_size]

assert s.sendall(block) is None, print("Ошибка отправки изображения: %s", e)

if n\_remainder > 0:

block = img\_bytes[-n\_remainder:]

assert s.sendall(block) is None, print("Ошибка отправки изображения: %s", e)

print("Изображение успешно отправлено.")

def add\_noise(self, image):

img = image.copy()

h, w, c = img.shape

# Генерация шума

gaussian = np.random.normal(0, 1, img.size)

gaussian = gaussian.reshape(h, w, c).astype('uint8')

# Добавление шума на изображение

img\_gaussian = cv2.add(img, gaussian)

return img\_gaussian

noiseserver = NoiseServer(noise\_server\_port=5065, final\_server\_port=5066)

img = noiseserver.run()

**Класс Client:**

import socket

import cv2

import numpy as np

import matplotlib.pyplot as plt

import sys

HOST = "127.0.0.1"

class Client:

def \_\_init\_\_(self, server\_port, image, block\_size=1024):

self.server\_port = server\_port

self.block\_size = block\_size

self.img = image

self.socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

def send\_image(self):

# Отправка размера изображения

h, w, \_ = img.shape

n = h \* w \* 3

h\_bytes = h.to\_bytes(length=4, byteorder='big')

w\_bytes = w.to\_bytes(length=4, byteorder='big')

img\_bytes = img.tobytes()

with self.socket as s:

print(f"Ожидание соединения с портом {self.server\_port}.")

s.connect((HOST, self.server\_port))

print("Connected to server.")

assert s.sendall(h\_bytes) is None, print("Ошибка отправки h\_bytes: %s", e)

assert s.sendall(w\_bytes) is None, print("Ошибка отправки w\_bytes: %s", e)

# отправка изображения

n\_iters = n // self.block\_size

n\_remainder = n - n\_iters \* self.block\_size

for i in range(n\_iters):

block = img\_bytes[self.block\_size\*i:(i+1)\*self.block\_size]

assert s.sendall(block) is None, print("Ошибка отправки изображения: %s", e)

if n\_remainder > 0:

block = img\_bytes[-n\_remainder:]

assert s.sendall(block) is None, print("Ошибка отправки изображения: %s", e)

print("Изображение успешно отправлено.")

# Загрузка изображения и преобразование изображения в массив numpy

img = cv2.imread("./data/jpg.jpg")

# Просмотр изображения перед отправкой

print(img.shape)

plt.imshow(cv2.cvtColor(img, cv2.COLOR\_BGR2RGB))

client = Client(server\_port=5065,image=img)

client.send\_image()

**Вывод приложения**

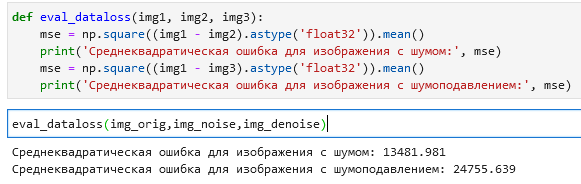


Рисунок 1 – Исходное изображение



Рисунок 2 – Изображение с шумом



Рисунок 3 – Восстановленное изображение  
  
Рисунок 4 – Результаты анализа потерь