1. Implementation of Stop and Wait protocol.

calc.py

```
def xor(a, b):
    result = []
    for i in range(1, len(b)):
        if a[i] == b[i]:
            result.append('0')
        else:
            result.append('1')

    return ''.join(result)


def mod2div(dividend, divisor):
    pick = len(divisor)
    tmp = dividend[0: pick]

    while pick < len(dividend):

        if tmp[0] == '1':
            tmp = xor(divisor, tmp) + dividend[pick]
        else:
            tmp = xor('0'*pick, tmp) + dividend[pick]
        pick += 1

    if tmp[0] == '1':
        tmp = xor(divisor, tmp)
    else:
        tmp = xor('0'*pick, tmp)

    return tmp


if __name__ == "__main__":
    dividend = "10010101"
    divisor = "011010"
    print(dividend + " % " + divisor + " = " + mod2div(dividend, divisor))
```

configuration.py

```
# Number of characters to be read
# at once.
FRAME_SIZE = 10

# Buffer size of tcp socket
BUFFER_SIZE = 1024 * 10
```



```
# Constant to determine the end
# of file and transaction.
END_OF_FILE = "##**##**##**##**##"

# CRC generator key
CRC_GENERATOR = "10110100110101110011010101110100000101"

# Accept and reject acknowledgements
# from receiver to the sender.
REJECT = "NAK"
ACCEPT = "OK"
```

error_gen.py

```
import random

class err_gen() :

    def induce_err(self, in_str) :
        chk = (int)(random.random() * 1000) % 2

        if not chk :
            return in_str

        idx = (int)(random.random() * 1000) % len(in_str)
        f_bit = '*'
        if in_str[idx] == '0':
            f_bit = '1'
        else :
            f_bit = '0'

        out_str = in_str[ : idx] + f_bit + in_str[idx + 1 : ]
        return out_str

if __name__ == "__main__":
    data = "1001010"
    print("Initial : ", data)
    print("Final : ", err_gen().induce_err(data))
```

client.py

```
from calc import mod2div
from error_gen import err_gen
from configuration import *
import socket


class Client:
```



```
    def __init__(self, ipadd, portn):
        self.socket_ = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.socket_.connect((ipadd, portn))

    def asciiToBin(self, data):

        # ascii to bin.
        return bin(int.from_bytes(data.encode(), 'big'))

    def appendZero(self, message):

        # append n - 1 0's.
        message = (message.ljust(len(CRC_GENERATOR) - 1 + len(message), '0'))
        return message

    def encode(self, data):

        # convert ascii to bin
        message = self.asciiToBin(data)
        dividend = self.appendZero(message)

        # generate and append crc
        crc = mod2div(dividend, CRC_GENERATOR)
        curr_frame = (message + crc)

        return curr_frame

    def send_file(self, filename='file.txt'):

        f = open(filename)
        data = f.read(FRAME_SIZE)

        while len(data) > 0:

            # encode data
            curr_frame = self.encode(data)

            # induce error
            curr_frame = err_gen().induce_err(curr_frame)

            # send frame
            self.socket_.send(curr_frame.encode())

            # receive acknowledgement
            if self.socket_.recv(BUFFER_SIZE).decode() == 'OK':
                data = f.read(FRAME_SIZE)

        # Terminate session
        self.socket_.send(END_OF_FILE.encode())
        self.socket_.close()
```



```
        f.close()
        print("File sent")


newclient = Client(ipadd="127.0.0.1", portn=3241)
newclient.send_file(filename="file.txt")
```

server.py

```
import socket
from calc import mod2div
from configuration import *


class Server:

    def __init__(self, ipaddr, portn):

        self.socket_ = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.socket_.bind((ipaddr, portn))
        self.socket_.listen(5)

    def iszero(self, data):
        for x in data:
            if x != '0':
                return False
        return True

    def isCurrupted(self, message):
        return not self.iszero(mod2div(message, CRC_GENERATOR))

    def decode(self, message):
        message = message[: 1 - len(CRC_GENERATOR)]
        n = int(message, 2)
        return n.to_bytes((n.bit_length() + 7) // 8, 'big').decode()

    def log(self, loghandle, itr, received_frame, retrie_count):

        loghandle.write("Frame Number : " + str(itr) + "\n")
        loghandle.write("Frame Content : \"" +
                        self.decode(received_frame) + "\"\n")
        loghandle.write("Retries : " + str(retrie_count) + "\n\n")

    def receive_file(self, filepath, logpath):

        received_socket, addr = self.socket_.accept()

        f = open(filepath, 'w')
        l = open(logpath, 'w')
```

```
        itr = 1
        retrie_count = 0

        while 1:

            received_frame = received_socket.recv(BUFFER_SIZE).decode()

            if received_frame == END_OF_FILE:
                f.close()
                l.close()
                self.socket_.close()
                print("File Received")
                return

            if self.isCurrupted(received_frame):
                retrie_count += 1
                received_socket.send(REJECT.encode())

            else:
                # Received file
                f.write(self.decode(received_frame))

                # Log
                self.log(l, itr, received_frame, retrie_count)

                retrie_count = 0
                received_socket.send(ACCEPT.encode())

newServer = Server(ipaddr="127.0.0.1", portn=3241)
newServer.receive_file(filepath="received_data.txt", logpath="logfile.txt")
```

Output (logfile.txt):

```
Frame Number : 1
Frame Content : "Stop and w"
Retries : 0

Frame Number : 1
Frame Content : "ait protoc"
Retries : 1

Frame Number : 1
Frame Content : "ol is an e"
Retries : 0

Frame Number : 1
Frame Content : "rror contr"
Retries : 3

Frame Number : 1
Frame Content : "ol protoco"
Retries : 0

Frame Number : 1
Frame Content : "l."
Retries : 0
```


2. Implementation of Sliding-Window protocol.

server.py

```
import socket
import pickle

def receive_data():
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    server_socket.bind(('localhost', 12345))

    window_size = 3
    base = 0
    frames = {}

    while True:
        data, client_addr = server_socket.recvfrom(1024)
        frame = pickle.loads(data)

        if frame["seq_num"] == base:
            print(f"Received Frame {frame['seq_num']}: {frame['data']}")
            base += 1

            # Acknowledge the received frame
            ack_frame = {"ack_num": base - 1}
            server_socket.sendto(pickle.dumps(ack_frame), client_addr)

            # Slide the window
            for i in range(base, base + window_size):
                frames.pop(i, None)

def main():
    print("Server is listening...")
    receive_data()

if __name__ == "__main__":
    main()
```

client.py

```
import socket
import pickle
import time

class SlidingWindow:
    def __init__(self, window_size):
        self.window_size = window_size
        self.base = 0
        self.next_seq_num = 0
        self.frames = {}
```



```
        self.server_addr = ('localhost', 12345)
        self.client_socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    def send_frame(self, data):
        if self.next_seq_num < self.base + self.window_size:
            frame = {"seq_num": self.next_seq_num, "data": data}
            self.client_socket.sendto(pickle.dumps(frame), self.server_addr)
            print(f"Sent Frame {self.next_seq_num}: {data}")
            self.frames[self.next_seq_num] = data
            self.next_seq_num += 1
        else:
            print("Window is full. Cannot send more frames.")

    def receive_ack(self):
        ack_data, _ = self.client_socket.recvfrom(1024)
        ack_frame = pickle.loads(ack_data)
        ack_num = ack_frame["ack_num"]
        print(f"Received ACK {ack_num}")
        if ack_num >= self.base:
            self.base = ack_num + 1
            self.slide_window()

    def slide_window(self):
        print(f"Sliding window to base {self.base}")
        for seq_num in range(self.base, self.next_seq_num):
            print(f"   Removing Frame {seq_num} from window")
            self.frames.pop(seq_num, None)

    def simulate_protocol(self):
        for i in range(10):
            data = f"Data_{i}"
            self.send_frame(data)
            self.receive_ack()
            time.sleep(1)

def main():
    window_size = 3
    client = SlidingWindow(window_size)
    client.simulate_protocol()

if __name__ == "__main__":
    main()
```


Output (server):

```
Server is listening...
Received Frame 0: Data_0
Received Frame 1: Data_1
Received Frame 2: Data_2
Received Frame 3: Data_3
Received Frame 4: Data_4
Received Frame 5: Data_5
Received Frame 6: Data_6
Received Frame 7: Data_7
Received Frame 8: Data_8
Received Frame 9: Data_9
```

Output (Client):

```
Sent Frame 0: Data_0
Received ACK 0
Sliding window to base 1
Sent Frame 1: Data_1
Received ACK 1
Sliding window to base 2
Sent Frame 2: Data_2
Received ACK 2
Sliding window to base 3
Sent Frame 3: Data_3
Received ACK 3
Sliding window to base 4
Sent Frame 4: Data_4
Received ACK 4
Sliding window to base 5
Sent Frame 5: Data_5
Received ACK 5
Sliding window to base 6
Sent Frame 6: Data_6
Received ACK 6
Sliding window to base 7
Sent Frame 7: Data_7
Received ACK 7
Sliding window to base 8
Sent Frame 8: Data_8
Received ACK 8
Sliding window to base 9
Sent Frame 9: Data_9
Received ACK 9
Sliding window to base 10
```

3. Implementation of TCP client server model.

server.py

```
import socket

# Create a TCP/IP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Bind the socket to the port
server_address = ('localhost', 10000)
print('Starting up on {} port {}'.format(*server_address))
sock.bind(server_address)

# Listen for incoming connections
sock.listen(1)

while True:
    # Wait for a connection
    print('Waiting for a connection')
    connection, client_address = sock.accept()
    try:
        print('Connection from', client_address)

        # Receive the data in small chunks and retransmit it
        while True:
            data = connection.recv(16)
            print('Received {!r}'.format(data))
            if data:
                print('Sending data back to the client')
                connection.sendall(data)
            else:
                print('No data from', client_address)
                break

    finally:
        # Clean up the connection
        connection.close()
```

client.py

```
import socket

# Create a TCP/IP socket
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Connect the socket to the port where the server is listening
server_address = ('localhost', 10000)
print('Connecting to {} port {}'.format(*server_address))
```



```
sock.connect(server_address)

try:
    # Send data
    message = b'This is the message.  It will be repeated.'
    print('Sending {!r}'.format(message))
    sock.sendall(message)

    # Look for the response
    amount_received = 0
    amount_expected = len(message)

    while amount_received < amount_expected:
        data = sock.recv(16)
        amount_received += len(data)
        print('Received {!r}'.format(data))

finally:
    print('Closing socket')
    sock.close()
```

Output (Server):

```
Starting up on localhost port 10000
Waiting for a connection
Connection from ('127.0.0.1', 55540)
Received b'This is the mess'
Sending data back to the client
Received b'age.  It will be'
Sending data back to the client
Received b' repeated.'
Sending data back to the client
Received b''
No data from ('127.0.0.1', 55540)
```

Output (Client):

```
Connecting to localhost port 10000
Sending b'This is the message.  It will be repeated.'
Received b'This is the mess'
Received b'age.  It will be'
Received b' repeated.'
Closing socket
```


4. Implementation of UDP client server model.

server.py

```
# Again we import the necessary socket python module
import socket
# Here we define the UDP IP address as well as the port number that we have
# already defined in the client python script.
UDP_IP_ADDRESS = "127.0.0.1"
UDP_PORT_NO = 1234

# declare our serverSocket upon which
# we will be listening for UDP messages
serverSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# One difference is that we will have to bind our declared IP address
# and port number to our newly declared serverSock
serverSock.bind((UDP_IP_ADDRESS, UDP_PORT_NO))

while True:
    data, addr = serverSock.recvfrom(1024)
    print("Message: " + data.decode())
```

client.py

```
import socket
UDP_IP_ADDRESS = "127.0.0.1"
UDP_PORT_NO = 1234
Message = "Hello, Server".encode()

clientSock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

while True:
    msg = str(input("Enter your message: "))
    msg = msg.encode()
    clientSock.sendto(msg, (UDP_IP_ADDRESS, UDP_PORT_NO))
```

Output (Server):

```
Message: Hi
Message: Hello
Message: What is UDP?
Message: Bye
```

Output (Client):

```
Enter your message: Hi
Enter your message: Hello
Enter your message: What is UDP?
Enter your message: Bye
Enter your message:
```


5. Implementation of HTTP for webpage upload and download.

Server.java

```
import java.net.*;
import java.io.*;
import java.awt.image.*;
import javax.imageio.*;
import javax.swing.*;

class Server {
    public static void main(String args[]) throws Exception {
        ServerSocket server = null;
        Socket socket;
        server = new ServerSocket(4000);
        System.out.println("Server Waiting for image");

        socket = server.accept();
        System.out.println("Client connected.");

        InputStream in = socket.getInputStream();
        DataInputStream dis = new DataInputStream(in);

        int len = dis.readInt();
        System.out.println("Image Size: " + len / 1024 + "KB");

        byte[] data = new byte[len];
        dis.readFully(data);
        dis.close();
        in.close();

        InputStream ian = new ByteArrayInputStream(data);
        BufferedImage bImage = ImageIO.read(ian);

        JFrame f = new JFrame("Server");
        ImageIcon icon = new ImageIcon(bImage);
        JLabel l = new JLabel();

        l.setIcon(icon);
        f.add(l);
        f.pack();
        f.setVisible(true);

        server.close();
    }
}
```

Client.java

```
import java.net.*;
import java.io.*;
import java.awt.image.BufferedImage;
import java.io.ByteArrayOutputStream;
import java.io.File;
import javax.imageio.ImageIO;

public class Client {
    public static void main(String args[]) throws Exception {

        Socket soc;
        BufferedImage img = null;
        soc = new Socket("localhost", 4000);
        System.out.println("Client is running. ");

        try {
            System.out.println("Reading image from disk. ");
            img = ImageIO.read(new File("flower.JPG"));
            ByteArrayOutputStream baos = new ByteArrayOutputStream();

            ImageIO.write(img, "jpg", baos);
            baos.flush();

            byte[] bytes = baos.toByteArray();
            baos.close();

            System.out.println("Sending image to server. ");

            OutputStream out = soc.getOutputStream();
            DataOutputStream dos = new DataOutputStream(out);

            dos.writeInt(bytes.length);
            dos.write(bytes, 0, bytes.length);

            System.out.println("Image sent to server. ");

            dos.close();
            out.close();

        } catch (Exception e) {
            System.out.println("Exception: " + e.getMessage());
            soc.close();
        }
        soc.close();
    }
}
```


Output (Server):

```
Server Waiting for image
Client connected.
Image Size: 16KB
```

Output (Client):

```
Client is running.
Reading image from disk.
Sending image to server.
Image sent to server.
```


6. Implementation of Remote Procedure Call.

server.py

```
# Server side (server.py)
from xmlrpc.server import SimpleXMLRPCServer
from xmlrpc.server import SimpleXMLRPCRequestHandler

# Restrict to a particular path.
class RequestHandler(SimpleXMLRPCRequestHandler):
    rpc_paths = ('/RPC2',)

# Create server
with SimpleXMLRPCServer(('localhost', 8000),
                        requestHandler=RequestHandler) as server:
    def add(x, y):
        return x + y

    def multiply(x, y):
        return x * y

    # Register functions
    server.register_function(add, 'add')
    server.register_function(multiply, 'multiply')

    # Run the server
    print("Server listening on port 8000...")
    server.serve_forever()
```

client.py

```
# Client side (client.py)
import xmlrpc.client

# Create server proxy
proxy = xmlrpc.client.ServerProxy("http://localhost:8000/RPC2")

# Call remote functions
result_add = proxy.add(4, 5)
result_multiply = proxy.multiply(3, 7)

print("Result of add: ", result_add)
print("Result of multiply: ", result_multiply)
```

Output (Server):

```
Server listening on port 8000...
127.0.0.1 - - [04/Feb/2024 00:07:31] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [04/Feb/2024 00:07:33] "POST /RPC2 HTTP/1.1" 200 -
```

Output (Client):

```
Result of add:  9
Result of multiply:  21
```