University of Dublin,
Trinity College

Computer Networks (CS3D3)

Project 1 - Data Link Simulation

Author:
Edmond O'Flynn 12304742

Lecturer:
Dr. Hitesh Tewari

February 28, 2016

1 Documentation

1.1 Transmitter Reference Documentation

```
struct DATA_PACKET { ... };
```

Contains information of each part of transmission stage for data encapsulation within a preset structure that is concatenated together with the following members:

bitset<> header → sequence number of the data frame
bitset<> data → 4 character data encapsulated from ascii
bitset<> trailer → generated checksum that may or may not be subject to gremlin_checksum()
string data_frame → the concatenated frame consisting of a header, payload, and trailer

Where each bitset member is a cast to binary representations which are then concatenated into an 8 byte long binary string under data_frame. The struct also contains string representations of the binary members above which allow for debugging of information stored and passed into the struct itself.

```
char generate_random_ascii();
```

A function representing the pseudo-random choice of the predefined alphanumeric character array which generates seeded values for transmission.

@param null
@return an alphanumeric character from the statically defined choice array.

```
int generate_random_number(int low, int high);
```

Pseudo-random number generation for a range where the parameters passed are the upper and lower values for seeding ascii generation.

@param int low → the lower limit of the range selection
@param int high → the upper limit of the range selection
@return → an integer value pseudo-randomly chosen from between low and high.

```
string generate_random_string();
```

Utilises the generate_random_ascii() function to return 1024 seeded values which are stored in a temporary holder.

@param null
@return a 1024 long pseudo-randomly generated alphanumeric character selection.

```
string gremlin_checksum(bitset<TRAILER_SIZE_BITS> data);
```

Takes a parameter of ascii characters, converts the given characters to their binary equivalents, and returns the remainder under the CRC 32 checksum where the chance of generating bit flips is a seeded process which has a 20% chance of occurring on each subsequent encoding.

@param bitset<> data → the data from which the checksum can be seeded and generated
@return the CRC 32 checksum for the given parameter of data

```
string get_allocated_ip(struct sockaddr_in addr);
```

Takes the given socket address struct passed, and returns its IPv4 equivalent for the given machine.

@param struct sockaddr_in addr → AF_INET representation of the allocated socket address.

@return a character array of IPv4 type for the machine's local address

```
string get_data();
```

Helper function to open a given *.txt file and return its contents into a buffer for use.

@param null
@return a file's contents for use in transmission

```
bitset<HEADER_SIZE_BITS> generate_header(int sequenceNumber);
```

Takes the parameter of a sequence number for a given data packet and returns its representation in binary.

@param int sequenceNumber → the given packet number for the data packet undergoing transmission.
@return the binary representation of the data packet's sequence number in the binary base system.

```
bitset<DATA_SIZE_BITS> generate_data(string data);
```

Takes the parameter of ascii characters to be encoded and returns their representation under a given size in binary representation.

@param string data → the payload of alphanumeric character to be encoded in binary representation
@return the binary representation of the payload in its binary representation from ascii

```
bitset<TRAILER_SIZE_BITS> generate_checksum(string data, int length);
```

Takes the conditional parameters of data to be encoded as well as its length, and returns a checksum under the CRC 32 algorithm checksum.

@param string data → alphanumeric characters used to generate the appropriate checksum
@param int length → length of the given characters to be encoded to a checksum representation
@return the checksum of the given characters in the binary representation

```
bitset<TRAILER_SIZE_BITS> generate_trailer(string data);
```

A top level, cascading function which relies on the output of its child functions in order to act as a parent, encapsulating function which invokes trailer checksum generation through taking returned values as the parent function's parameter. Beforehand, the function prepares the data to be passed by reverse-encoding the data passed into the appropriate representation of binary from ascii.

@usage return bitset<TRAILER_SIZE_BITS>(gremlin_checksum(generate_checksum(
data_concat, data_concat.length())));
@param string data → the data to be packaged
@return the checksum given the output of the other parameters in a cascading format

```
void generate_output_file();
```

Helper function to generate a *.txt file that stores the output of the character generation functions for use in transmission.

@param null
@return null

```
void split_data(string data);
```

Helper function to split the given 1024 long character array into packets of length 4 such that for 1024 characters, 256 character packet elements each of length 4 are generated of IDs 0 to 255.

@usage [0:xxxx][1:xxxx]...[255:xxxx]
@param null
@return null

```
void open_socket();
```

Kills any live connections and opens a UDP connection for the client to connect to the server's IP and port.

@param null
@return null

```
void close_connection();
```

Kills any connections on the client's socket.

@param null
@return null

```
void display_connection_info();
```

Displays the connection information for the IP and server being connected to by the client.

@param null
@return null

```
void send_data(int sequenceNumber, string data\_packet);
```

Calls appropriate socket handling, and sends the given data packet to the server.

@param int sequenceNumber → debug information to display the current packet being sent on the UI
@param string data_packet → encapsulated packet of binary information being sent to the server
@throws exception on send errors and kills the connection
@return null

```
void receive_echo();
```

Opens a listener on a given IP and port in order to receive information on the integrity of the sent packet to the server, and act accordingly depending on its reception of good or bad for the packet sent.

@param null
@return null

```
void receive_ack(int sequenceNumber, string data);
```

Instantiation of handling if a packet's echo is good or bad. If bad, another packet is generated which has a chance of being corrupted by the gremlin function, else the next packet is generated and sent to the server until the last packet is reached.

@param int sequenceNumber → the debug information for the packet sequence number for UI
@param string data → the data packet regenerated and reencapsulated.
@return null

```
bool hasEnding (string const &data, string const &regex);
```

Boolean test to return whether or not the current string contains the given regex for the given string

@param string data → data to be analysed for string regex
@param string regex → regex for analysis

@return true given the regex exists at the end of the given string

```
string stuff_bits(string data);
```

Takes the parameter of a string to be stuffed and returns the string in its stuffed format ready for transmission with a given redundancy.

@usage "0111110" → "01111010"
@param data → raw data to be stuffed
@return the stuffed data in raw format

```
string pad(string data);
```

Given a packet less than the maximum packet length, the packet is padded with post-affixed 0s until the maximum packet length is achieved

@param data → the data to be padded
@return the padded data with respect to the maximum packet length

1.2 Receiver Reference Documentation

```
string get_allocated_ip(struct sockaddr_in addr);
```

Takes the given socket address struct passed, and returns its IPv4 equivalent for the given machine.

@param struct sockaddr_in addr → AF_INET representation of the allocated socket address.
@return a character array of IPv4 type for the machine's local address

```
bitset<TRAILER_SIZE_BITS> generate_trailer(string data);
```

A top level, cascading function which relies on the output of its child functions in order to act as a parent, encapsulating function which invokes trailer checksum generation through taking returned values as the parent function's parameter. Beforehand, the function prepares the data to be passed by reverse-encoding the data passed into the appropriate representation of binary from ascii.

@usage return bitset¡TRAILER_SIZE_BITS¿(gremlin_checksum(generate_checksum(data_concat, data_concat.le
@param string data → the data to be packaged
@return the checksum given the output of the other parameters in a cascading format

```
string analyse_integrity(string data, string trailer);
```

Returns whether or not the checksum matches the integrity for the given payload in the data packet with GOOD or BAD.

@param string data → data to be examined for its integrity
@param string trailer → given checksum parametrised for data integrity checking @return compares the checksum matching regex, if there is a match, then it returns GOOD, else BAD and requests a resend.

```
void start_listener();
```

Kills any open sockets and starts a UDP listener with a bind to a given server IP and port.

@throws exception on incorrect binding
@param null
@return null

```
void receive_data();
```

Initialises a listener that receives from a source over the UDP protocol, and accesses data received that is saved under header, data and trailer given certain lengths. Also displays information of given packet to the UI.

@param null
@return null

```
void echo(string result);
```

Calls an initialiseation of echo to the client of either GOOD or BAD integrity, invoking reinitialisaiton of transmission or incrementation of packet transmission.

@throws exception on send error
@param string result → result of GOOD or BAD for client reception
@return null

```
void close_connection();
```

Kills any connection on the given socket.

@param null
@return null

```
void start_echo();
```

Kills the current connection and reinitialises to an inverse client-server relationship where the server returns a message to the client of the packet's integrity.

@param null
@return null

```
void save_to_file(string data);
```

Saves the input from transmission via a buffer to file having displayed its contents to screen.

@throws exception on incorrect file saving
@param string data → the data to be saved to file
@return null

```
static int binary_to_dec(string sequenceNumber);
```

Helper to convert binary data to a decimal representation from the packet.

@param string sequenceNumber → binary sequence number to be converted to decimal representation
@return the decimal representation of the binary string passed via parametrisation

```
bitset<16> generate_checksum(string data, int length);
```

Generates a given checksum on the server side for comparison with the packaged trailer encapsulated in the data frame.

@param string data → alphanumeric characters to have a checksum generated for
@param int length → length of data
@return the checksum for the given character data

```
bool hasEnding (string const &data, string const &regex);
```

Boolean test to return whether or not the current string contains the given regex for the given string

@param string data → data to be analysed for string regex
@param string regex → regex for analysis
@return true given the regex exists at the end of the given string

```
string unstuff_bits(string data);
```

Takes the parameter of a string to be unstuffed and returns the string in its unstuffed format ready for interpretation with a given redundancy.

@usage "01111010" → "0111110"
@param data → stuffed data to be unstuffed
@return the unstuffed data in raw format

2 Source Code

2.1 Transmitter.cpp

```
//data and bit manipulation
#include <iostream>
#include <cstring>
#include <ctime>
#include <cstdlib>
#include <fstream>
#include <sstream>
#include <bitset>

//TCP connection handling
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <arpa/inet.h>

using namespace std;

//start of consts ----->
static const char ascii_chars[] =
"!$%^&*()_+-={}[];'#:@~,./<>?"
"abcdefghijklmnopqrstuvwxyz"
"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
"0123456789";

static const int RAW_FILE_SIZE = 1024;
static const int ARRAY_SIZE = RAW_FILE_SIZE / 4;
string dataStream[ARRAY_SIZE];

//size of payload parts in bytes
static const int HEADER_SIZE_BYTES = 1;
static const int HEADER_SIZE_BITS = HEADER_SIZE_BYTES * 8;
static const int DATA_SIZE_BYTES = 4;
static const int DATA_SIZE_BITS = DATA_SIZE_BYTES * 8;
static const int TRAILER_SIZE_BYTES = 2;
static const int TRAILER_SIZE_BITS = TRAILER_SIZE_BYTES * 8;
static const int MAX_DATA_PACKET_SIZE_BYTES = 8;
static const int MAX_DATA_PACKET_SIZE_BITS = MAX_DATA_PACKET_SIZE_BYTES * 8;

static const int SERVER_PORT = 2016;
static const int CLIENT_PORT = 2015;
static const char* IP_CLIENT = "127.0.0.1";
static const char* IP_HOST = "127.0.0.1";

static const string regex = "011111";

struct sockaddr_in client_addr, server_addr;
```

```
int client_len = sizeof(client_addr);
int server_len = sizeof(server_addr);
socklen_t addr_len = sizeof(struct sockaddr);
int client_socket;
int req_send, req_recv;

const static int BUFFER_SIZE = 1024;
char buffer[BUFFER_SIZE];
const int MAX_PACKETS = 255;

struct DATA_PACKET
{
   //data to be sent
   bitset<HEADER_SIZE_BITS> header;
   bitset<DATA_SIZE_BITS> data;
   bitset<TRAILER_SIZE_BITS> trailer;
   string data_frame;

   //reference value for debug
   string header_string;
   string data_string;
   string trailer_string;
};

char generate_random_ascii();
int generate_random_number(int low, int high);

string generate_random_string();
string gremlin_checksum(bitset<TRAILER_SIZE_BITS> data);
string get_allocated_ip(struct sockaddr_in addr);
string get_data();

bitset<HEADER_SIZE_BITS> generate_header(int sequenceNumber);
bitset<DATA_SIZE_BITS> generate_data(string data);
bitset<TRAILER_SIZE_BITS> generate_checksum(string data, int length);
bitset<TRAILER_SIZE_BITS> generate_trailer(string data);

void generate_output_file();
void split_data(string data);
void open_socket();
void close_connection();
void display_connection_info();
void send_data(int sequenceNumber, string data_packet);
void receive_echo();
void receive_ack(int sequenceNumber, string data);

bool hasEnding (string const &data, string const &regex);
string stuff_bits(string data);
string pad(string data);

//<----- end of data transmission

int main()
{
   //generate and parse data
```

```
    generate_output_file();
    split_data(get_data());

    //parse data to packets if successful in binary from ascii chars
    DATA_PACKET dataPacket;
    for(int i = 0; i <= MAX_PACKETS; i++)
    {
        //set up sockets
        cout << "Packet #" << i << endl;
        open_socket();
        display_connection_info();

        //data processing
        dataPacket.header_string = to_string(i);
        dataPacket.data_string = dataStream[i];
        dataPacket.trailer_string = dataStream[i];

        dataPacket.header = generate_header(i);
        dataPacket.data = generate_data(dataStream[i]);
        dataPacket.trailer = generate_trailer(dataStream[i]);

        cout << "HEADER: " << dataPacket.header_string << " parsed to " << dataPacket.header <<
        " (" << dataPacket.header.size() << " bits)" << endl;
        cout << "DATA: " << dataPacket.data_string << " parsed to " << dataPacket.data <<
        " (" << dataPacket.data.size() << " bits)" << endl;
        cout << "TRAILER: checksum for " << dataPacket.trailer_string << " parsed to " <<
            dataPacket.trailer <<
        " (" << dataPacket.trailer.size() << " bits)" << endl << endl;

        dataPacket.data_frame = dataPacket.header.to_string() + dataPacket.data.to_string() +
            dataPacket.trailer.to_string();
        cout << "dataframe for packet #" << dataPacket.header_string << endl;
        cout << dataPacket.data_frame << endl << endl;

        //send the ith packet
        send_data(i, dataPacket.data_frame);

        //wait and receive ack, if echo has integrity, follow through with next packet
        receive_ack(i, dataPacket.data_string);
        usleep(2000);

        //reset
        close_connection();

        cout << endl << "************************************************************" << endl
            << endl;
    }

  cout << "Original message: " << endl;
  cout << get_data() << endl << endl;

    return 0;
}
```

```
//begin random file generation ----->

//generate a random character from the selection
char generate_random_ascii()
{
   return ascii_chars[rand() % (sizeof(ascii_chars) - 1)];
}

//generate a 1024 alphanumeric string
string generate_random_string()
{
   string holder;
   srand(time(0));

   for(int i = 0; i < 1024; i++)
   {
      holder += generate_random_ascii();
   }
   return holder;
}

//write the generated string to a txt file
void generate_output_file()
{
   string message = generate_random_string();

   cout << "String generated: " << message << endl;
   cout << "Chars generated: " << message.size() << endl;

   ofstream out;
   out.open("generated.txt");

   if(out.fail()){
      cout << "Error in opening file" << endl;
   } else {
      cout << "File created successfully" << endl;
      out << message;
      out.close();
   }
}

//<---- end file generation

//start of data encapsulation ----->
string get_data()
{
   string output;
   ifstream in;
   in.open("generated.txt");

   while(!in.eof()){
      getline(in, output);
   }

   in.close();
```

```
   return output;
}

void split_data(string data)
{
   //split each 4 characters into array elements
   //[0:xxxx][1:xxxx]...[255:xxxx]
   for(int i = 0; i < RAW_FILE_SIZE / 4; i++){
      dataStream[i] = data.substr(i * 4, 4);
   }
   dataStream[MAX_PACKETS] = data.substr(1020, 4);
}

bitset<HEADER_SIZE_BITS> generate_header(int sequenceNumber)
{
   return bitset<HEADER_SIZE_BITS>(sequenceNumber);
}

bitset<DATA_SIZE_BITS> generate_data(string data)
{
   char charArray[data.length()];
   data.copy(charArray, data.length());

   string bits;

   for(int i = 0; i < data.length(); i++)
   {
      bitset<8> x(charArray[i]);
      bits += x.to_string();
   }

   return bitset<DATA_SIZE_BITS>(bits);
}

bitset<TRAILER_SIZE_BITS> generate_checksum(string data, int length){
   bitset<16> polynomial(65535);
   bitset<16> remainder(data);
   bitset<16> divisor(128);

   for(unsigned int bit = 8; bit > 0; bit--)
   {
      //check if uppermost bit is 1
      if((remainder & divisor) == 1)
      {
         //XOR remainder with divisor
         remainder ^= polynomial;
      }
      //shift bit into remainder
      remainder = (remainder << 1);
   }

   cout << "Poly " << polynomial << ", Divisor " << divisor << ", CRC " << (remainder >> 4)
       << endl;
   return (remainder >> 4);
}
```

```
int generate_random_number(int low, int high) {
  if (low > high)
     return high;
  else return low + (rand() % (high - low + 1));
}

string gremlin_checksum(bitset<TRAILER_SIZE_BITS> data)
{
   bitset<16> remainder(data);
   int random = generate_random_number(0,99);
   if(random < 10)
   {
      //invoked -- flip checksum bits
      remainder = ~remainder;
      cout << "Gremlin function invoked! CRC is now " << remainder.to_string() << endl;
   }
   return remainder.to_string();
}

bitset<TRAILER_SIZE_BITS> generate_trailer(string data)
{
   char charArray[data.length()];
   data.copy(charArray, data.length());

   string bits;

   for(int i = 0; i < data.length(); i++)
   {
      bitset<8> x(charArray[i]);
      bits += x.to_string();
   }

   string data_concat = bits.substr(0, 8);
   return bitset<TRAILER_SIZE_BITS>(gremlin_checksum(generate_checksum(data_concat,
       data_concat.length())));
}

bool hasEnding(string const &data, string const &regex) {
    return data.size() >= regex.size() &&
           data.compare(data.size() - regex.size(), regex.size(), regex) == 0;
}

string stuff_bits(string data)
{
  //0111111... -> 01111101...
  cout << "Stuffing bits..." << endl;

  char stuffedArray[data.length()];
  data.copy(stuffedArray, data.length());

  string currSequence = "", stuffedSequence = "";

  for(int i = 0; i < data.length(); i++)
  {
```

```
    currSequence += stuffedArray[i];
    stuffedSequence += stuffedArray[i];

    if(hasEnding(currSequence, regex))
    {
      currSequence = "";
      stuffedSequence += "0";
    }
  }
  return stuffedSequence;
}

string pad(string data)
{
  cout << "Padding data from " << data.length() << " to " << MAX_DATA_PACKET_SIZE_BITS << "
     bits..." << endl;
  if(data.length() < MAX_DATA_PACKET_SIZE_BITS)
  {
    while(data.length() < MAX_DATA_PACKET_SIZE_BITS)
      data+="0";
  }
  return data;
}

//<----- end of data encapsulation

//start of data transmission ----->

string get_allocated_ip(struct sockaddr_in addr)
{
   char address[INET_ADDRSTRLEN];
  return inet_ntop(AF_INET, &(addr.sin_addr.s_addr), address, INET_ADDRSTRLEN);
}

void close_connection()
{
   close(client_socket);
}

void open_socket()
{
  close_connection();
   client_socket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
   server_addr.sin_family = AF_INET;
   server_addr.sin_port = htons(SERVER_PORT);
   inet_aton(IP_HOST, &server_addr.sin_addr);
}

void display_connection_info()
{
   cout << endl;
   cout << "Opening connection on " << get_allocated_ip(server_addr) << ":" << SERVER_PORT <<
      "..." << endl;
}
```

```
void send_data(int sequenceNumber, string data_packet)
{
   open_socket();
   usleep(100);
   cout << "Sending packet #" << sequenceNumber << "..." << endl;
   const char* data = pad(stuff_bits(data_packet)).c_str();
   strcpy(buffer, data);

   req_send = sendto(client_socket, &buffer, BUFFER_SIZE, 0, (struct sockaddr*) &server_addr,
       sizeof(server_addr));
   if(req_send == -1)
   {
      cout << "Send error!" << endl;
   close_connection();
   }
   else
   {
      cout << "Packet sent successfully..." << endl << endl;
   }
}

void receive_echo()
{
   close(client_socket);
   client_socket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
   client_addr.sin_family = AF_INET;
   client_addr.sin_port = htons(CLIENT_PORT);
   client_addr.sin_addr.s_addr = inet_addr(IP_CLIENT);
   cout << "Listening for echo on " << get_allocated_ip(client_addr) << ":" << CLIENT_PORT <<
       endl;

   int result = bind(client_socket, (struct sockaddr*) &client_addr, sizeof(struct sockaddr));
   if(result == -1)
   {
      cout << "Error on binding!" << endl;
   close_connection();
   }
   else
   {
      cout << "Bind successful, listening on " << get_allocated_ip(client_addr) << ":" <<
          CLIENT_PORT << endl;
   }
}

void receive_ack(int sequenceNumber, string data)
{
   cout << "Waiting for ack..." << endl;
   receive_echo();
   //clear the buffer before proceding to continue as not to overflow it
   memset(buffer, 0, sizeof(buffer));
   req_recv = recvfrom(client_socket, &buffer, BUFFER_SIZE, 0, (struct sockaddr*)
       &client_addr, (socklen_t*) &client_addr);

   string ack = buffer;
   if(ack == "GOOD" && sequenceNumber != MAX_PACKETS)
```

```
    {
        cout << "Server echo: GOOD integrity in data, sending next packet..." << endl;
    }
    else if(ack == "BAD")
    {
        cout << "Server echo: BAD integrity in data, regenerate packet..." << endl;

        //regenerate datapacket as before with a chance of the gremlin function
        string header = generate_header(sequenceNumber).to_string();
        string data_packet = generate_data(dataStream[sequenceNumber]).to_string();
        string trailer = generate_trailer(dataStream[sequenceNumber]).to_string();
        string data_frame = header + data_packet + trailer;
        cout << endl << "dataframe for regenerated packet (" << sequenceNumber << ")" << endl;
        cout << data_frame << endl << endl;

        //send the ith packet
        cout << "Retry sending data..." << endl;
        send_data(sequenceNumber, data_frame);
    }
    else if(ack == "GOOD" && sequenceNumber == MAX_PACKETS)
    {
        cout << "Server echo: GOOD integrity in data, finishing transmission..." << endl;
    }
    close_connection();
}

//<----- end of data transmission
```

2.2 Receiver.cpp

```
//data handling
#include <iostream>
#include <cstring>
#include <ctime>
#include <cstdlib>
#include <fstream>
#include <sstream>
#include <bitset>

//UDP connection
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <arpa/inet.h>

using namespace std;

static const int SERVER_PORT = 2016;
static const int CLIENT_PORT = 2015;
static const int BUFFER_SIZE = 1024;
char buffer[BUFFER_SIZE];
const int MAX_PACKETS = 255;

static const char* IP_CLIENT = "127.0.0.1";
static const char* IP_HOST = "127.0.0.1";

static const string regex = "011111";

int server_socket, result;
int req_send, req_recv;
struct sockaddr_in server_addr, client_addr;

bool isEnd = false;
string message;

string get_allocated_ip(struct sockaddr_in addr);
string generate_trailer(string data);
string analyse_integrity(string data, string trailer);

void start_listener();
void receive_data();
void echo(string result);
void close_connection();
void start_echo();
void save_to_file(string data);
static int binary_to_dec(string sequenceNumber);
bitset<16> generate_checksum(string data, int length);
```

```
bool hasEnding (string const &data, string const &regex);
string unstuff_bits(string data);

int main()
{
   cout << endl << "************************************************************" << endl <<
      endl;
   cout << "Server started..." << endl;
   while(!isEnd){receive_data();}
  save_to_file(message);
  close_connection();
  return 0;
}

string get_allocated_ip(struct sockaddr_in addr)
{
   char address[INET_ADDRSTRLEN];
  return inet_ntop(AF_INET, &(addr.sin_addr.s_addr), address, INET_ADDRSTRLEN);
}

void close_connection()
{
   close(server_socket);
}

void start_listener()
{
  close(server_socket);
   server_socket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
   server_addr.sin_family = AF_INET;
   server_addr.sin_port = htons(SERVER_PORT);
   server_addr.sin_addr.s_addr = inet_addr(IP_HOST);

   result = bind(server_socket, (struct sockaddr*) &server_addr, sizeof(struct sockaddr));

   if(result == -1)
   {
     cout << "Bind error with status " << result << "..." << endl;
   }
   else
   {
     cout << "Bind successful, listening on " << get_allocated_ip(server_addr) << ":" <<
        SERVER_PORT << "..." << endl;
   }
}

void start_echo()
{
   close(server_socket);
   server_socket = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP);
   client_addr.sin_family = AF_INET;
   client_addr.sin_port = htons(CLIENT_PORT);
   client_addr.sin_addr.s_addr = inet_addr(IP_CLIENT);
}
```

```
static int binary_to_dec(string sequenceNumber)
{
  return stoi(sequenceNumber, nullptr, 2);
}

static string binary_to_ascii(string data)
{
  string output;
  stringstream sstream(data);
  while(sstream.good())
  {
    bitset<8> bits;
    sstream >> bits;
    char c = char(bits.to_ulong());
    output += c;
  }
  return output;
}

bitset<16> generate_checksum(string data, int length){
   bitset<16> polynomial(65535);
   bitset<16> remainder(data);
   bitset<16> divisor(128);

   for(unsigned int bit = 8; bit > 0; bit--)
   {
      //check if uppermost bit is 1
      if((remainder & divisor) == 1)
      {
         //XOR remainder with divisor
         remainder ^= polynomial;
      }
      //shift bit into remainder
      remainder = (remainder << 1);
   }

  return (remainder >> 4);
}

string generate_trailer(string data)
{
   char charArray[data.length()];
   data.copy(charArray, data.length());

   string bits;

   for(int i = 0; i < data.length(); i++)
   {
      bitset<8> x(charArray[i]);
      bits += x.to_string();
   }

   string data_concat = bits.substr(0, 8);

   return bitset<16>(generate_checksum(data_concat, data_concat.length())).to_string();
```

```
}

string analyse_integrity(string data, string trailer)
{
  if(generate_trailer(data) == trailer)
    return "GOOD";
  else return "BAD";
}

void echo(string result)
{
      //establish a new echo connection with the client from the receiving address
      start_echo();
      socklen_t server_addr_len = sizeof(server_addr);

      //parse result of integrity check for client packet handling
      cout << "Echoing " << result << " to client..." << endl;
      const char* reply = result.c_str();
      strcpy(buffer, reply);

      usleep(1000);

      req_send = sendto(server_socket, &buffer, BUFFER_SIZE, 0, (struct sockaddr*)
          &client_addr, sizeof(client_addr));

      if(req_send == -1)
      {
         cout << "Send error!" << endl;
      }
      else
      {
         cout << "Echo successful to client" << endl;
      }
      close_connection();
}

void receive_data()
{
   memset(buffer, 0, sizeof(buffer));
   start_listener();
   req_recv = recvfrom(server_socket, &buffer, BUFFER_SIZE, 0, (struct sockaddr*)
       &client_addr, (socklen_t*) &client_addr);

   cout << endl << "**********************************************************" << endl <<
       endl;
   cout << "Retrieved data: " << endl << buffer << endl;
   string header, data, trailer, received(buffer);
   received = unstuff_bits(received);
   header = received.substr(0, 8);
   data = received.substr(8, 32);
   trailer = received.substr(40, 16);

   if(binary_to_dec(header) > 0)
   {
      cout << "Reinitialising server listener..." << endl;
```

```
   }

   if(binary_to_dec(header) == MAX_PACKETS)
   {
      isEnd = true;
   }

   string decompiled_data = binary_to_ascii(data);

   cout << "Packet " << binary_to_dec(header) << "/" << MAX_PACKETS << endl;
   cout << "Data contained in payload: \"" << decompiled_data << "\"" << endl;
   cout << "Integrity: " << analyse_integrity(binary_to_ascii(data), trailer) << endl;

   echo(analyse_integrity(binary_to_ascii(data), trailer));

   if(analyse_integrity(binary_to_ascii(data), trailer) == "GOOD")
   {
      message += decompiled_data;
   }
   //usleep(1000);
}

void save_to_file(string data)
{
   cout << endl << "************************************************************" << endl <<
       endl;
   cout << "Transmission finished!" << endl;
   ofstream out;
   out.open("received.txt");

   cout << message << endl << endl;

   if(out.fail()){
      cout << "Error in opening file" << endl;
   } else {
      out << data;
      out.close();
      cout << "File created successfully" << endl << endl;
   }
}

bool hasEnding(string const &data, string const &regex) {
    return data.size() >= regex.size() &&
           data.compare(data.size() - regex.size(), regex.size(), regex) == 0;
}

string unstuff_bits(string data)
{
  //01111101... -> 0111111...
  cout << "Unstuffing bits..." << endl;

  char stuffedArray[data.length()];
  data.copy(stuffedArray, data.length());

  string currSequence = "", unstuffedSequence = "";
```

```
  for(int i = 0; i < data.length(); i++)
  {
    currSequence += stuffedArray[i];
    unstuffedSequence += stuffedArray[i];

    if(hasEnding(currSequence, regex))
    {
      currSequence = "";
      i++;
    }
  }

  return unstuffedSequence;
}
```