UDPConnection

Software Documentation

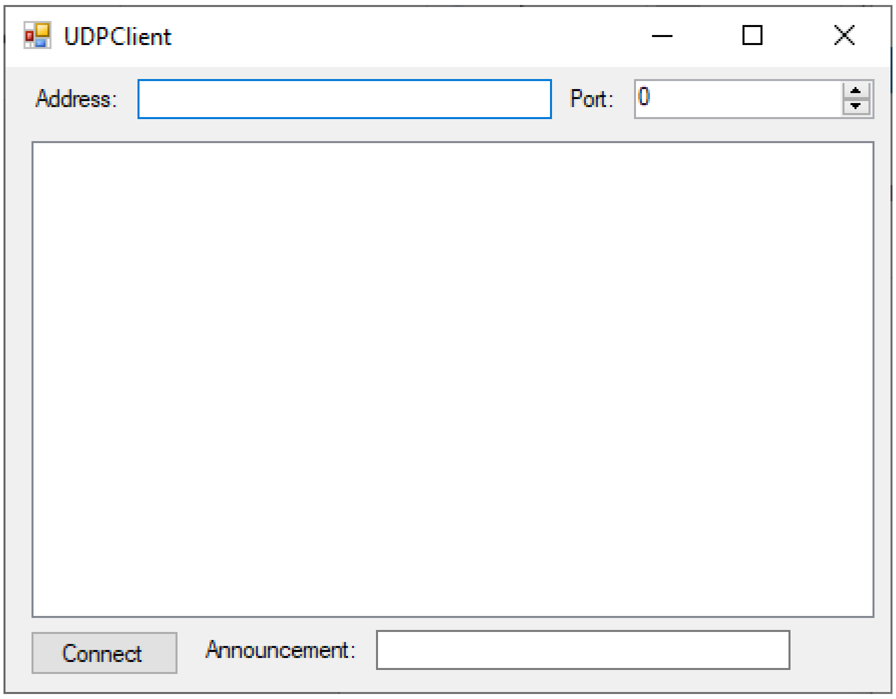
Author: matiwa

Table of contents

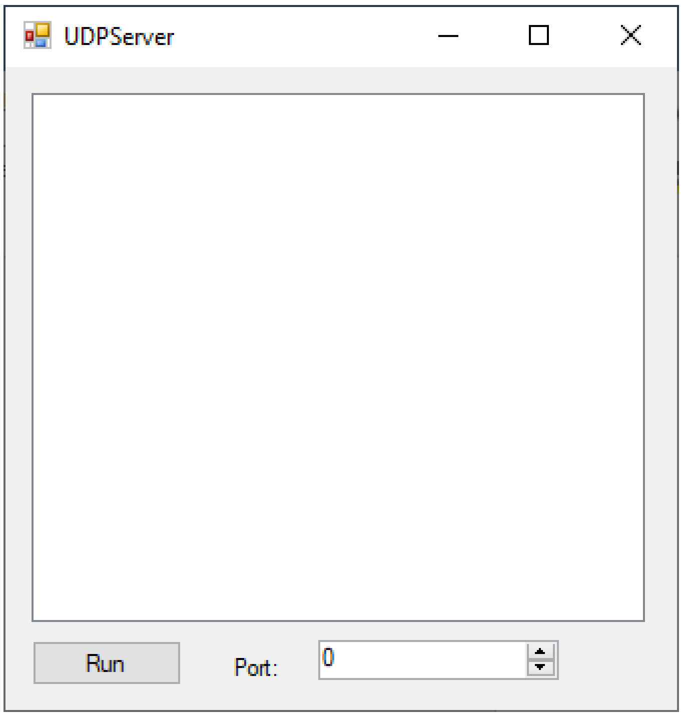
Introduction

This software documentation includes: description of the application's operation, what is needed for use, algorithms used, interface description and source code description. These applications are used to establish UDP connections with each other according to the principles of the client-server architecture. The concept consists of two projects, UDPClient and UDPServer.

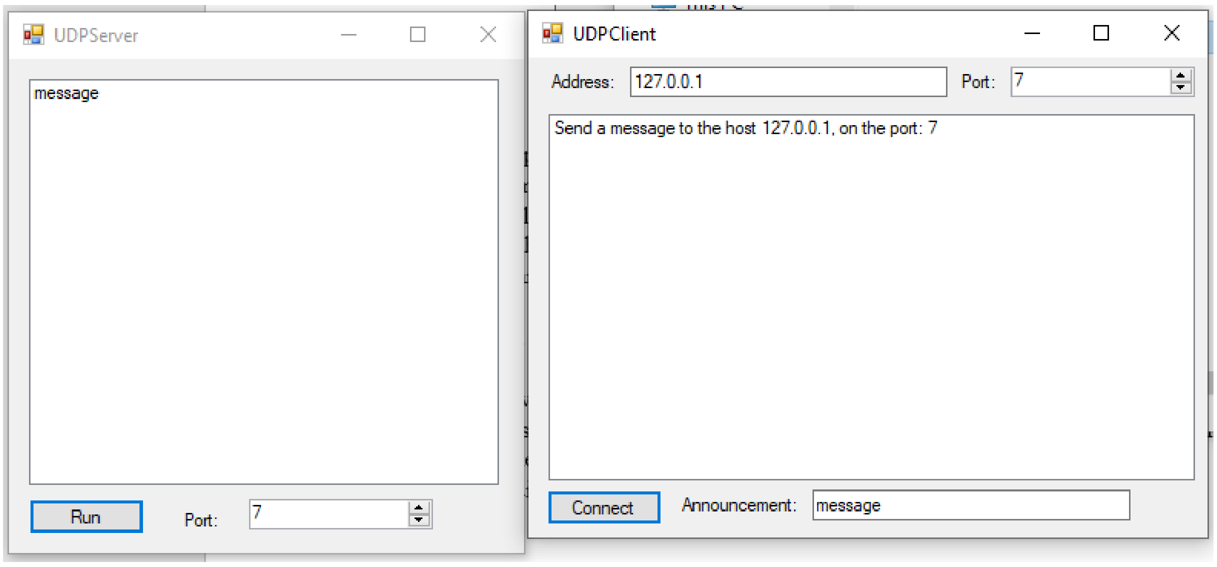
Describing of the application’s operation



Drawing 1: The beginning of the application's operation UDPClient [own study]



Drawing 2: The beginning of the application's operation UDPServer [own study]



message

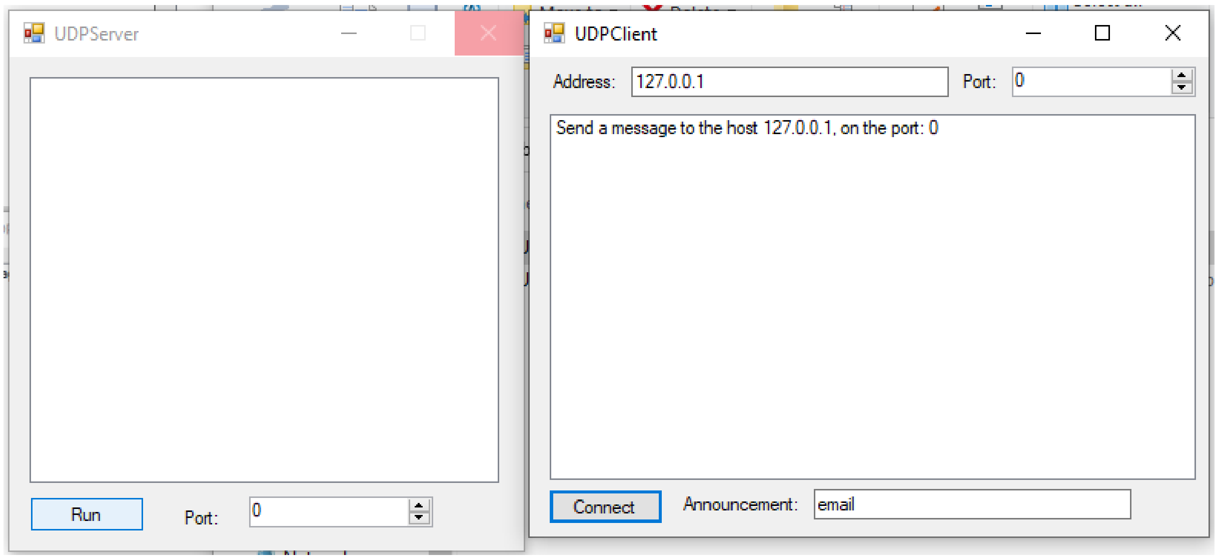
Address: 127.0.0.1

Port: 7

Send a message to the host 127.0.0.1, on the port: 7

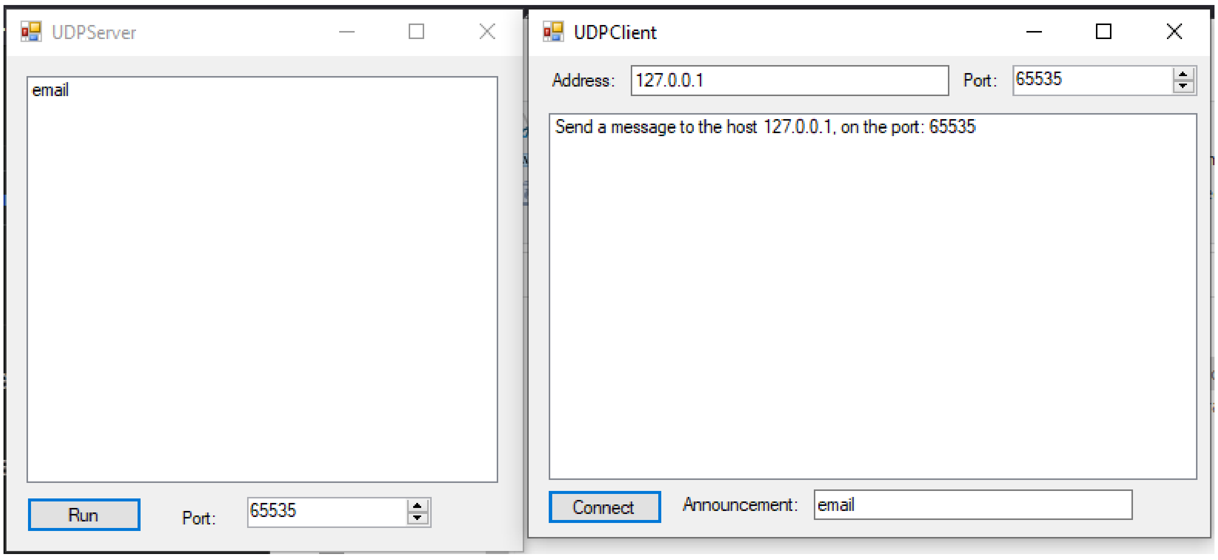
Announcement: message

Drawing 3: Operation of the port structure for network services [own study]



Send a message to the host 127.0.0.1, on the port: 0

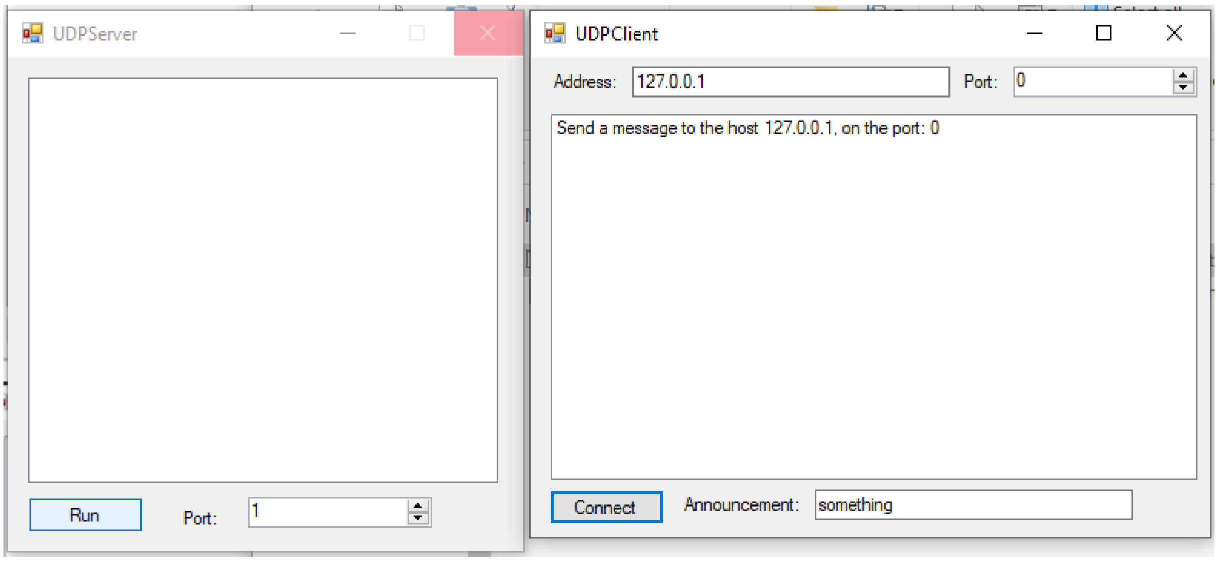
Drawing 4: The operation of the structure for port 0 [own study]



email

Send a message to the host 127.0.0.1, on the port: 65535

Drawing 5: Structure activity for a port in scope for the application [own study]



Send a message to the host 127.0.0.1, on the port: 0

Drawing 6: Structure behavior for incompatible ports [own study]

After starting the server program, the user sets the port on which to listen. Then presses the Run button. Then the user goes to the client application. He types in the IP address (127.0.0.1 for localhost), sets the same port as the server is listening to, and enters the message. Finally, it confirms the will to connect with the Connect button. If everything is fine, the message content appears on the server side, and the client displays a message that the message with the given content has been sent to the given server's IP address on the specified port. There are comments about the port. Although ports 0 through 1023 are reserved for network services and ports 1024 through 65535 are reserved for user applications, the connection is normally made for a port greater than 1. In the event of a port or port 0 conflict, the server application hangs. These shortcomings are for possible work, but are not necessary to show the essence of the guiding structure. There are certainly some bugs that the developer didn't discover while working on the apps.

What is needed for use?

The applications do not require installation. They only need the Windows operating system.

Algorithm used

The basic form of the algorithm can be deduced from the previous section. They only needs the Windows operating system. In summary, These applications are used to establish UDP connections with each other in accordance with the principles of the client-server architecture.

Client / server, client-server model - the architecture of the computer system, in particular software, enabling the division of tasks (roles). This is done by establishing that the server is providing services to clients that make service requests to the server.

The basic, most common servers operating on the basis of the client-server architecture are: e-mail server, web server, file server, application server. Typically multiple clients can use one server. A single client, in general, may use multiple servers simultaneously. According to the client-server scheme, most of the currently used database management systems also operate.

In some simplification, without going into technical details of the implementation, the method of communication according to the client-server architecture can be characterized by defining tasks (assigning roles) to both parties and defining their modes of operation.

Client side - this is the party requesting access to the given service or resource.

Client work mode:

- active,
- sends the request to the server,
- waiting for a response from the server.

Server side - this is the site that provides a service or resource.

Server operation mode:

- passive,
- waiting for requests from customers,
- upon receiving the request, it processes it and then sends a response.

Due to the division of performed tasks, the following types of client-server architecture are distinguished:

- two-tier architecture - data processing and storage takes place in one module
- three-tier architecture - data processing and storage takes place in two separate modules
- multi-tier architecture - processing, storage and other data operations take place in many separate modules.

The connection between the client and the server is described using specific communication protocols. The most common is TCP / IP. In most cases, communication is based on a pattern where the client connects to the server. It then sends a request in the specified format to the server and waits for a response. The server is waiting for the clients all the time and when it receives the request, it processes them and sends a response. In the OSI model, communication between the parties takes place at the application layer.

P2P is a type of different architecture where each host can act as both a client and a server.

Benefits:

- All information is stored on the server, so better data security is possible. The server can decide who has the right to read and change the data.
- There are many advanced technologies supporting the operation, safety and usability of this type of solution.

Disadvantages:

- A large number of clients trying to receive data from one server causes various types of problems related to the bandwidth of the link and the technical ability to process customer requests.
- While the server is down, data access is completely impossible.
- To run a server unit capable of serving a large number of clients, special software and hardware are required, which are not found in most home computers.

The closest example is the organization of access to Internet resources, where:

- the role of the server is played by a web server,
- the role of the client is played by a web browser.

When browsing websites, the user's computer is the client, and the computers that run databases and other applications needed to handle the connection are the server. When the browser requests a page, the server searches for the relevant information in the database, converts it into a website, and then sends it to the client. [1]

UDP (User Datagram Protocol) - one of the Internet protocols. UDP is used in the transport layer of the OSI model. It does not guarantee delivery of the datagram.

It is a connectionless protocol, so there is no overhead for connecting and session tracking (unlike TCP). There are also no flow control and retransmission mechanisms. The advantage of such simplification of construction is faster data transmission and the lack of additional tasks that must be dealt with by the host using this protocol. For these reasons, UDP is often used in applications such as video conferencing, Internet audio streaming and network games, where data must be transferred as quickly as possible and other layers of the OSI model deal with correcting errors. Examples include VoIP or DNS.

UDP provides a mechanism for identifying various endpoints (e.g. running applications, services or services) on one host thanks to ports (compare: socket). UDP deals with the delivery of individual packets, shared over the IP on which it is based. Another feature that distinguishes UDP from TCP is the ability to transmit to several destination addresses at once (multicast).

UDP packets, also called datagrams, contain a UDP header in addition to lower-level headers. It consists of fields containing the checksum, packet length and ports: source and destination.

Like TCP, UDP ports are written on two bytes (sixteen bits), so each IP address can be assigned 65,535 different endings. For historical reasons, ports 0-1023 are reserved for well-known network services - ports from 1024 are allocated to user applications.

Sender's port - identifies the port from which the message was sent, when the signifier indicates the port of the sending process and can be taken as the port to which the reply message should be returned in the absence of other information. The sender's port is an optional field. When this field is not used, it is set to zero.

Recipient's port - identifies the recipient port and is a required field.

Length - the 16-bit fields specify the length in bytes of the entire datagram: header and data. The minimum length is 8 bytes, which is the length of the header. The field size provides a theoretical limit of 65,527 bytes for data carried over a single UDP datagram.

Checksum - A 16 bit field that is used to validate the header and data. The field is optional. Since IP does not check sum for the data, the UDP checksum is the only guarantee that the data has not been corrupted.

When UDP is running on IPv4, the method used to compute the checksum is specified in RFC 768 ↓

Whole 16-bit words are summed together using the one's complement code (the checksum field is set to zero). The final value is inserted into the checksum field.

The difference is in the data used to create the checksum.

The source and destination addresses are included in the IPv4 header. The UDP field length consists of its header and data. If the checksum is computed and is 0, it should be sent as an alternate representation of zero in the unity's complement code (all 1). If the checksum is not used, it should be sent as a "regular" zero (all 0), indicating to the recipient that it is not being used.

The user interface should allow:

- create new ports for receiving data
- accept operations on these ports, return data objects, and point to the source port and data source address
- the operations that allow the datagram to send, data, source and destination ports, or their addresses. RFC 768 ↓

The UDP module must be able to determine source and destination Internet addresses, and distinguish between the protocol field and the header. One possible UDP / IP interface would return the entire datagram including the internet header in response to the received operation. A UDP interface would also allow the complete datagram and header to be sent over the IP protocol. The IP would check some fields to match and calculate the header checksum.

When UDP is running on IPv6, the checksum is no longer optional. The checksum calculation method is described in RFC 2460 ↓.

Any transport or other higher protocol layer that includes the addresses from the IP header in its checksum must be modified to be used. IPv6 must contain 128-bit addresses instead of the 32-bit ones used by IPv4.

There is one data source address in the IPv6 header. The destination address is the end address; if the IPv6 packet does not contain a routing header, the destination address will be the address contained in the IPv6 header, otherwise, at the emerging node it will be the address of the last routing header element and at the receiving node it will be the destination address in the IPv6 header. The next header value is the value for the UDP protocol. The length of the UDP field consists of its header and data. See art. Datagram Protocol source.

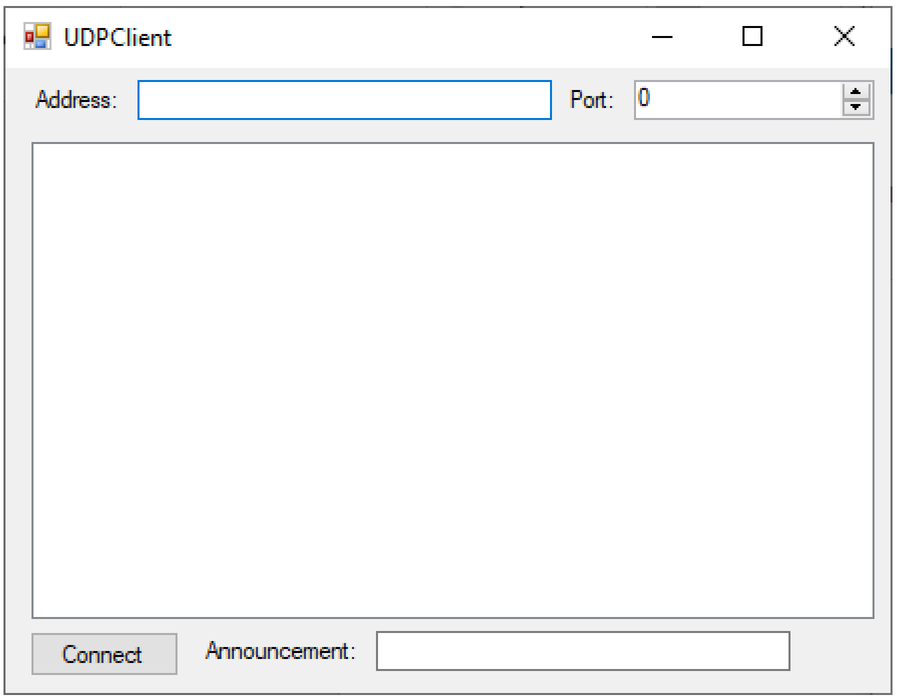
The user interface should allow:

- create new ports for receiving data
- accept operations on these ports, return data objects, and point to the source port and data source address
- the operations that allow the datagram to send, data, source and destination ports, or their addresses. RFC 768 ↓

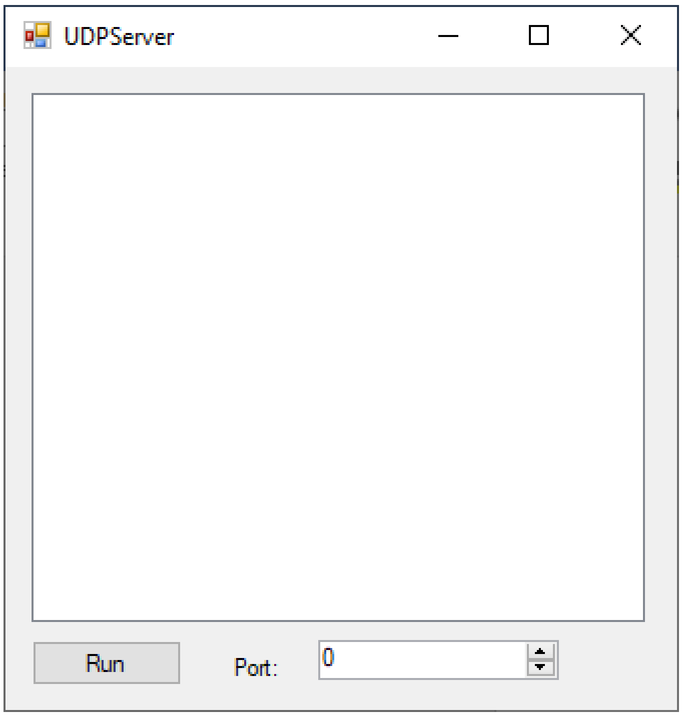
Encapsulating is embedding higher layer data inside a lower layer message. Before a message on the sender is sent, it is forwarded down the stack of the layer; each subsequent layer, after receiving this message from the layer above, adds its own header and footer to it. The UDP datagram is thus encapsulated in an IP datagram before being sent out to the network. The IP header identifies the source and destination machine, UDP - identifies the sender and receiver ports. The opposite process takes place for the recipient. The message is passed up the layer stack and each subsequent layer interprets and then removes the header previously added by the same layer at the sender.

So at the receiver, the packet reaches the lowest layer of the network software and travels to higher and higher layers. Each of them removes one header, the waiting process receives a message without headers. The UDP datagram received from the IP on the target machine is identical to the one UDP forwarded to the IP on the source machine.[2]

Interface description



Drawing 7: UDPClient graphical interface [own study]



Drawing 3: UDPServer graphical interface [own study]

The interfaces are typical for a Windows Forms Application. There are essential components: listbox, textboxes, labels, numericupdowns and buttons.

Source code description

The projects were made in the C# programming language, in the Visual Studio Community 2017 programming environment. All work was done on the Windows 10 operating system. The UDPClient application's source code looks like this.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Net.Sockets;

namespace UDPClient
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void BConnect_Click(object sender, EventArgs e)
        {
            listBox1.Items.Clear();
            string host = textBox1.Text;
            int port = (int)numericUpDown1.Value;
            try
            {
                UdpClient klient = new UdpClient(host, port);
                Byte[] dane = Encoding.ASCII.GetBytes(textBox2.Text);
                klient.Send(dane, dane.Length);
                listBox1.Items.Add("Send a message to the host " + host + ", on the
port: " + port);
                klient.Close();
            }
            catch (Exception ex)
            {
                listBox1.Items.Add("Error: The message could not be sent!");
                MessageBox.Show(ex.ToString(), "Error");
            }
        }
    }
}
```

Listing 1: UDPClient source code [own study]

The UDPServer application's source code looks like this.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Net.Sockets;
using System.Net;

namespace UDPServer
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void BRun_Click(object sender, EventArgs e)
        {
            int port = (int)numericUpDown1.Value;
            IPEndPoint zdalnyIP = new IPEndPoint(IPAddress.Any, 0);
            try
            {
                UdpClient serwer = new UdpClient(port);
                Byte[] odczyt = serwer.Receive(ref zdalnyIP);
                string dane = Encoding.ASCII.GetString(odczyt);
                listBox1.Items.Add(dane);
                serwer.Close();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Error");
            }
        }
    }
}
```
Listing 2: UDPServer source code [own study]

List of drawings

List of listings

Bibliography

[1] https://pl.wikipedia.org/wiki/Klient-serwer
[2] https://pl.wikipedia.org/wiki/User_Datagram_Protocol