Лабораторная работа

Тема: Создание сетевого приложения на основе TCP сокетов и перехват сетевого трафика

Теоретическая часть

Сокет — конечная точка связи двустороннего канала между 2 компьютерами.

Если мы соединим 2 сокета, то получим канал, через который можно передавать данные в обе стороны. Одна сторона канала называется сервером, другая — клиентом.

Для передачи/приема данных нужно открыть канал. В конце всех операций — закрыть.

Типы сокетов

Существует 2 вида сокетов: потоковые, дейтаграммные.

Потоковый сокет — это сокет, который состоит из потока байтов, который может быть двунапрямленным (в обе стороны). Он берет на себя всю ответственность о доставке данных и исправлении ошибок. Особенностью есть возможность передачи больших объемов данных. Использует протокол TCP (Transmission Control Protocol), именно который обеспечивает поступление данных на другую сторону в нужной последовательности и без ошибок.

Дейтаграммный сокет — в отличие от потокового, имеет ограничения по размеру. Реализован через протокол UDP (User Datagram Protocol), который не отвечает за приход в конечную точку всех данных. Одним из плюсов — не нужно создавать соединения между 2 сторонами. Это очень важно, когда затраты времени недопустимы.

Более подробная информация

http://msdn.microsoft.com/ru-ru/library/system.net.sockets.socket(v=vs.110).aspx

Практическая часть

Для примера создадим приложение-чат. Приложение будет состоять из двух частей: серверное приложение и клиентское приложение.

Реализация серверной части

Создадим новый проект Приложение Windows Forms на языке C#.

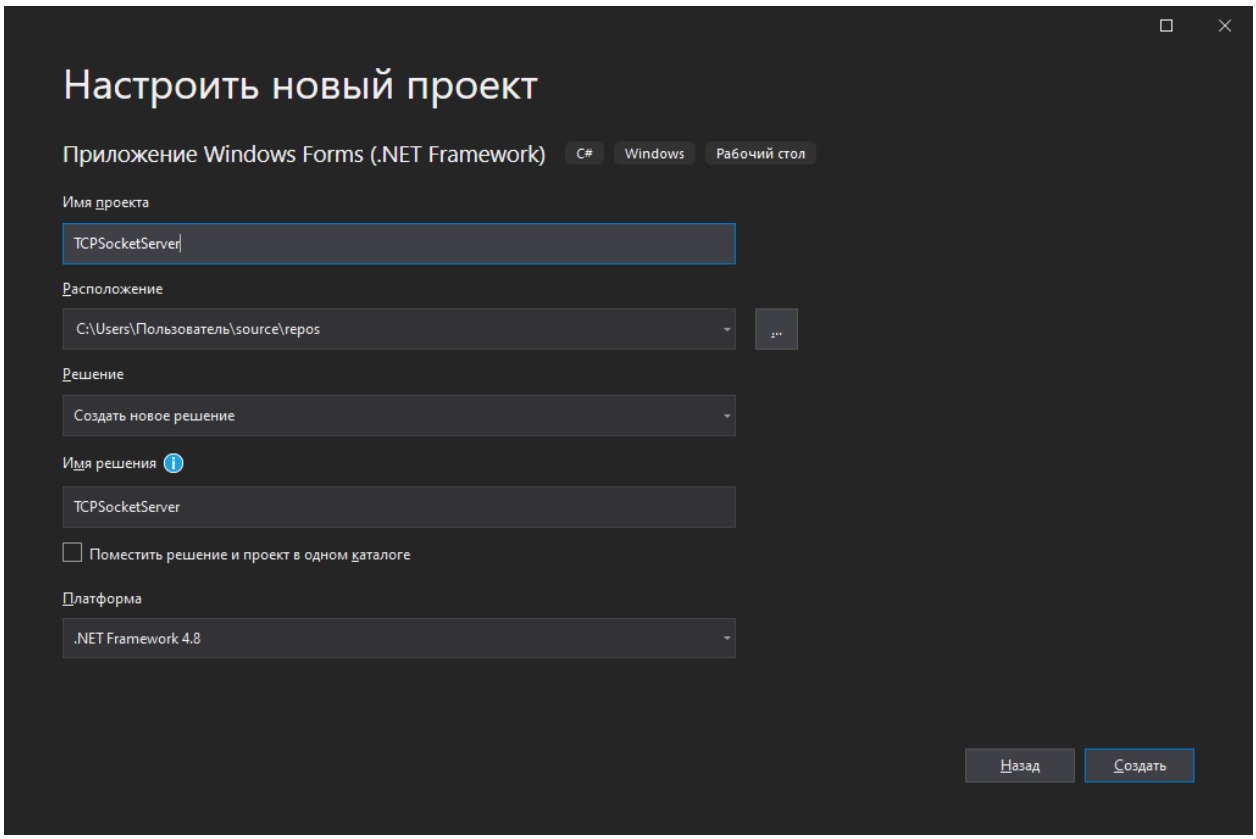

Рис.1. Создание проекта приложения

Добавим в проект пространства имен, методы которых будут использованы в проекте:

```
// Добавляем пространства имен для работы сокетов
using System.Net.Sockets;
using System.Net;
using System.Threading;
```

Объявим необходимые глобальные переменные:

```
// Раздел глобальных переменных
private static Socket Server; // Создаем объект сокета-сервера
private static Socket Handler; // Создаем объект вспомогательного сокета

private static IPHostEntry ipHost; // Класс для сведений об адресе веб-узла
private static IPAddress ipAddr; // Предоставляет IP-адрес
private static IPEndPoint ipEndPoint; // Локальная конечная точка

private static Thread socketThread; // Создаем поток для поддержки потока
private static Thread WaitingForMessage; // Создаем поток для приёма сообщений
```

Реализуем функцию startSocket для запуска сокета с целью прослушивания и ожидания подключения клиента:

```
// Функция запуска сокета
 private void startSocket()
 {
     // IP-адрес сервера, для подключения
     string HostName = "0.0.0.0";
     // Порт подключения
     string Port = tbPort.Text;
```

```
            // Разрешает DNS-имя узла или IP-адрес в экземпляр IPHostEntry.
            ipHost = Dns.Resolve(HostName);
            // Получаем из списка адресов первый (адресов может быть много)
            ipAddr = ipHost.AddressList[0];
            // Создаем конечную локальную точку подключения на каком-то порту
            ipEndPoint = new IPEndPoint(ipAddr, int.Parse(Port));

            try
            {
                // Создаем сокет сервера на текущей машине
                Server = new Socket(AddressFamily.InterNetwork,
                SocketType.Stream, ProtocolType.Tcp);
            }
            catch (SocketException error)
            {
                // 10061 – порт подключения занят/закрыт
                if (error.ErrorCode == 10061)
                {
                    MessageBox.Show("Порт подключения закрыт!");
                    Application.Exit();
                }
            }

            // Ждем подключений
            try
            {
                // Связываем удаленную точку с сокетом
                Server.Bind(ipEndPoint);
                // Не более 10 подключения на сокет
                Server.Listen(10);

                // Начинаем "прослушивать" подключения
                while (true)
                {
                    Handler = Server.Accept();
                    if (Handler.Connected)
                    {
                        // Позеленим кнопочку для красоты, чтобы пользователь знал, что
                        соединение установлено
                        bConnect.Invoke(new Action(() => bConnect.Text = "Клиент
                        подключен"));
                        bConnect.Invoke(new Action(() => bConnect.BackColor =
                        Color.Green));
                        // Создаем новый поток, указываем на ф-цию получения сообщений в
                        классе Worker
                        WaitingForMessage = new System.Threading.Thread(new
                        System.Threading.ParameterizedThreadStart(GetMessages));
                        // Запускаем, в параметрах передаем листбокс (история чата)
                        WaitingForMessage.Start(new Object[] { lbHistory });
                    }
                    break;
                }
            }
            catch (Exception e)
            {
                throw new Exception("Проблемы с подключением");
            }

    }
```

Запуск прослушивающего сокета производится по нажатию кнопки в отдельном потоке. Это необходимо для сохранения функциональности формы приложения:

```
socketThread = new Thread(new ThreadStart(startSocket));
socketThread.IsBackground = true;
socketThread.Start();
bConnect.Enabled = false;
bConnect.Text = "Ожидание подключения";
bConnect.BackColor = Color.Yellow;
```

В рамках метода startSocket при подключении клиента происходит запуск потока WaitingForMessage для организации принятия сообщений от клиента. В рамках потока исполняется метод GetMessages:

```
// Ф-ция, работающая в новом потоке: получение новых сообщений ———
 public static void GetMessages(Object obj)
 {
     // Получаем объект истории чата (лист бокс)
     Object[] Temp = (Object[])obj;
     System.Windows.Forms.ListBox ChatListBox =
(System.Windows.Forms.ListBox)Temp[0];

     // В бесконечном цикле получаем сообщения
     while (true)
     {
         // Ставим паузу, чтобы на время освобождать порт для отправки сообщений
         Thread.Sleep(50);

             try
             {
                 // Получаем сообщение от клиента
                 string Message = GetDataFromClient();
             // Добавляем в историю + текущее время
             ChatListBox.Invoke(new Action(() =>
ChatListBox.Items.Add(DateTime.Now.ToShortTimeString() + " Client:  " + Message)));
         }
             catch { }

     }
 }
```

В свою очередь метод GetMessages использует метод GetDataFromClient для принятия и обработки сообщений от клиента:

```
// Получение информации от клиента
public static string GetDataFromClient()
{
    string GetInformation = "";

    byte[] GetBytes = new byte[1024];
    int BytesRec = Handler.Receive(GetBytes);

    GetInformation += Encoding.Unicode.GetString(GetBytes, 0, BytesRec);

    return GetInformation;
}
```

Отправка сообщений от сервера к клиенту производится с использованием кода:

```
            // Посылаем клиенту новое сообщение
            SendDataToClient(tbMessage.Text);
            // Добавляем в историю свое же сообщение + ник + время написания
            lbHistory.Items.Add(DateTime.Now.ToShortTimeString() + " Server: " +
tbMessage.Text.ToString());
            // Автопрокрутка истории чата
            lbHistory.TopIndex = lbHistory.Items.Count - 1;
            // Убираем текст из поля ввода
            tbMessage.Text = "";
```

При этом используется метод SendDataToClient:

```
// Отправка информации на клиент
 public static void SendDataToClient(string Data)
 {
     byte[] SendMsg = Encoding.Unicode.GetBytes(Data);
     Handler.Send(SendMsg);
 }
```

Внешний вид приложения приведен на рисунке 2.

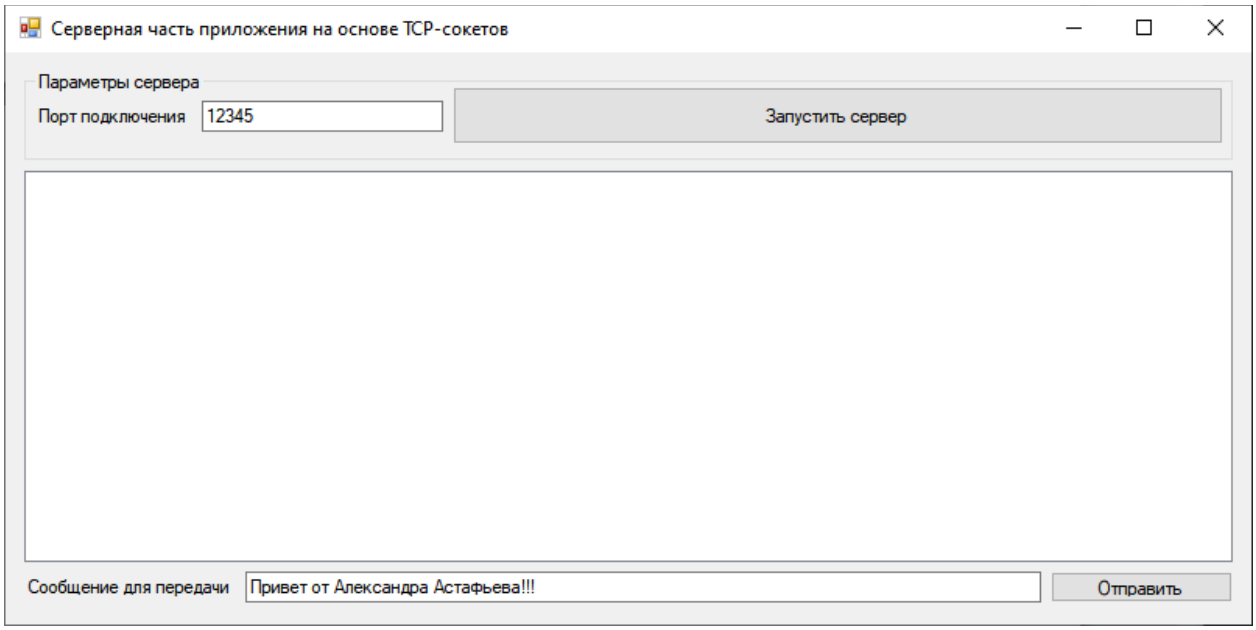

Рис. 2. Внешний вид приложения

Реализация клиентской части

Программная реализация клиентской части очень схожа с серверной. На первом этапе подключаем пространства имен:

```
// Добавляем пространства имен для работы сокетов
using System.Net.Sockets;
using System.Net;
using System.Threading;
```

Объявим необходимые глобальные переменные:

```
private static Socket Client; // Создаем объект сокета-сервера

private static IPHostEntry ipHost; // Класс для сведений об адресе веб-узла
private static IPAddress ipAddr; // Предоставляет IP-адрес
private static IPEndPoint ipEndPoint; // Локальная конечная точка

private static Thread socketThread; // Создаем поток для поддержки потока
private static Thread WaitingForMessage; // Создаем поток для приёма сообщений
```

Вид метода startSocket выглядит следующим образом:

```
private void startSocket()
 {
     // IP-адрес сервера, для подключения
     string HostName = tbAddress.Text;
     // Порт подключения
     string Port = tbPort.Text;

     // Разрешает DNS-имя узла или IP-адрес в экземпляр IPHostEntry.
     ipHost = Dns.Resolve(HostName);
     // Получаем из списка адресов первый (адресов может быть много)
     ipAddr = ipHost.AddressList[0];
     // Создаем конечную локальную точку подключения на каком-то порту
     ipEndPoint = new IPEndPoint(ipAddr, int.Parse(Port));

     try
     {
         // Создаем сокет на текущей машине
         Client = new Socket(AddressFamily.InterNetwork,
         SocketType.Stream, ProtocolType.Tcp);
         while (true)
         {
             // Пытаемся подключиться к удаленной точке
             Client.Connect(ipEndPoint);
             if (Client.Connected) // Если клиент подключился
             {
                 // Позеленим кнопочку для красоты, чтобы пользователь знал, что
                  соединение установлено
                 bConnect.Invoke(new Action(() => bConnect.Text = "Подключение
                  установлено"));
                 bConnect.Invoke(new Action(() => bConnect.BackColor =
                  Color.Green));
                 // Создаем новый поток, указываем на ф-цию получения сообщений в
                  классе Worker
                 WaitingForMessage = new System.Threading.Thread(new
                  System.Threading.ParameterizedThreadStart(GetMessages));
                 // Запускаем, в параметрах передаем листбокс (история чата)
                 WaitingForMessage.Start(new Object[] { lbHistory });
             }
             break;
         }
     }
     catch (SocketException error)
     {
         // 10061 – порт подключения занят/закрыт
         if (error.ErrorCode == 10061)
         {
             MessageBox.Show("Порт подключения закрыт!");
             Application.Exit();
         }
     }
 }
```

Метод GetMessages:

```
        // Ф-ция, работающая в новом потоке: получение новых сообщенй ———
         public static void GetMessages(Object obj)
         {
             // Получаем объект истории чата (лист бокс)
             Object[] Temp = (Object[])obj;
             System.Windows.Forms.ListBox ChatListBox =
(System.Windows.Forms.ListBox)Temp[0];

             // В бесконечном цикле получаем сообщения
             while (true)
             {
                 // Ставим паузу, чтобы на время освобождать порт для отправки сообщений
                 Thread.Sleep(50);
                     try
                     {
                         string Message = GetDataFromServer();
                         ChatListBox.Invoke(new Action(() =>
ChatListBox.Items.Add(DateTime.Now.ToShortTimeString() + " Server: " + Message)));
                     }
                     catch { }
             }
         }
```

Метод GetDataFromServer:

```
         // Получение данных от сервера
         public static string GetDataFromServer()
         {
             string GetInformation = "";

             // Создаем пустое “хранилище” байтов, куда будем получать информацию
             byte[] GetBytes = new byte[1024];
             int BytesRec = Client.Receive(GetBytes);

             // Переводим из массива битов в строку
             GetInformation += Encoding.Unicode.GetString(GetBytes, 0, BytesRec);

             return GetInformation;
         }
```

Метод SendDataToServer:

```
        // Отправка информации на сервер
         public static void SendDataToServer(string Data)
         {
             // Используем unicode, чтобы не было проблем с кодировкой, при приеме
информации
             byte[] SendMsg = Encoding.Unicode.GetBytes(Data);
             Client.Send(SendMsg);
         }
```

Код запуска сокета:

```
        socketThread = new Thread(new ThreadStart(startSocket));
        socketThread.IsBackground = true;
        socketThread.Start();
        bConnect.Enabled = false;
```

Код отправки сообщения на сервер:

```
        private void button2_Click(object sender, EventArgs e)
         {
             // Посылаем клиенту новое сообщение
             SendDataToServer(tbMessage.Text);
             // Добавляем в историю свое же сообщение + ник + время написания
             lbHistory.Items.Add(DateTime.Now.ToShortTimeString() + " Client:  " +
tbMessage.Text.ToString());
             // Автопрокрутка истории чата
             lbHistory.TopIndex = lbHistory.Items.Count - 1;
             // Убираем текст из поля ввода
             tbMessage.Text = "";
         }
```

Внешний вид клиентской части приведен на рисунке 3.

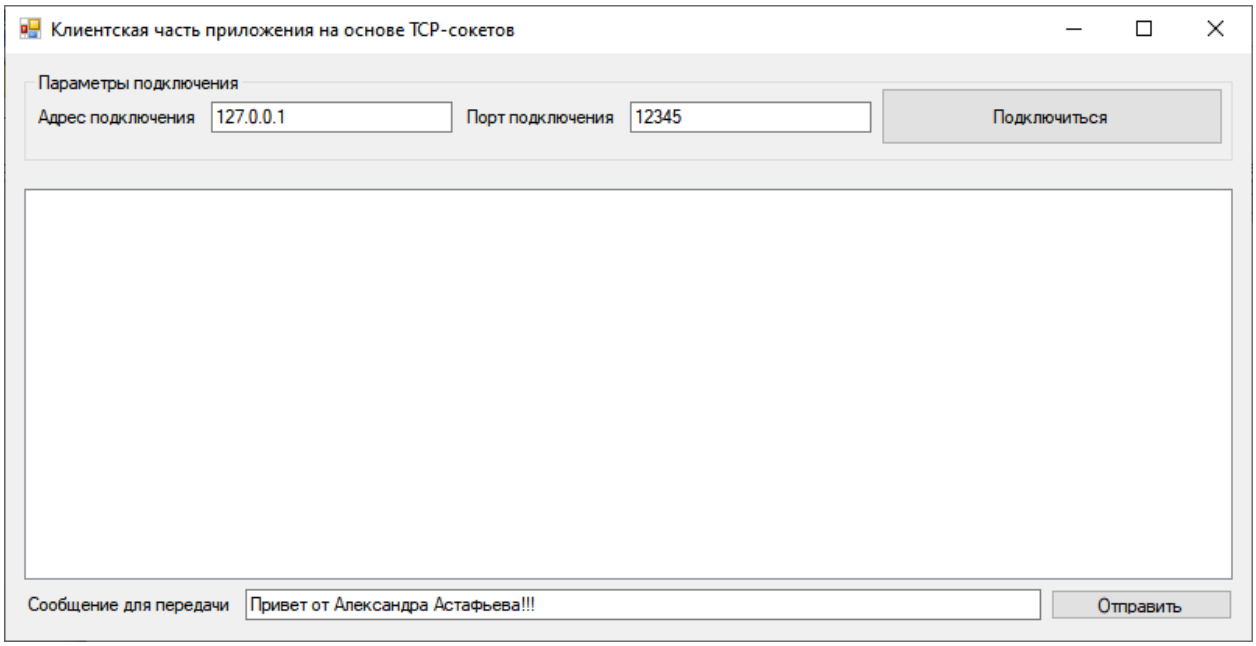

Рис. 3. Внешний вид клиентской формы

Процесс работы приложения:

1. Запускаем приложение-сервер.
2. Запускаем приложение-клиент.
3. На серверной части запускаем сокет с использованием кнопки «Запустить сервер». При этом необходимо указать порт подключения.
4. На клиентской части необходимо запустить сокет путем нажатия кнопки «Подключиться». При этом необходимо указать адрес подключения и порт подключения.
5. После успешной связи кнопки приложений окрасятся в зеленый цвет.
6. Для дальнейшей работы необходимо передавать сообщения с использованием кнопки «Отправить».

Пример работы клиент-серверного приложения приведен на рисунке 4.

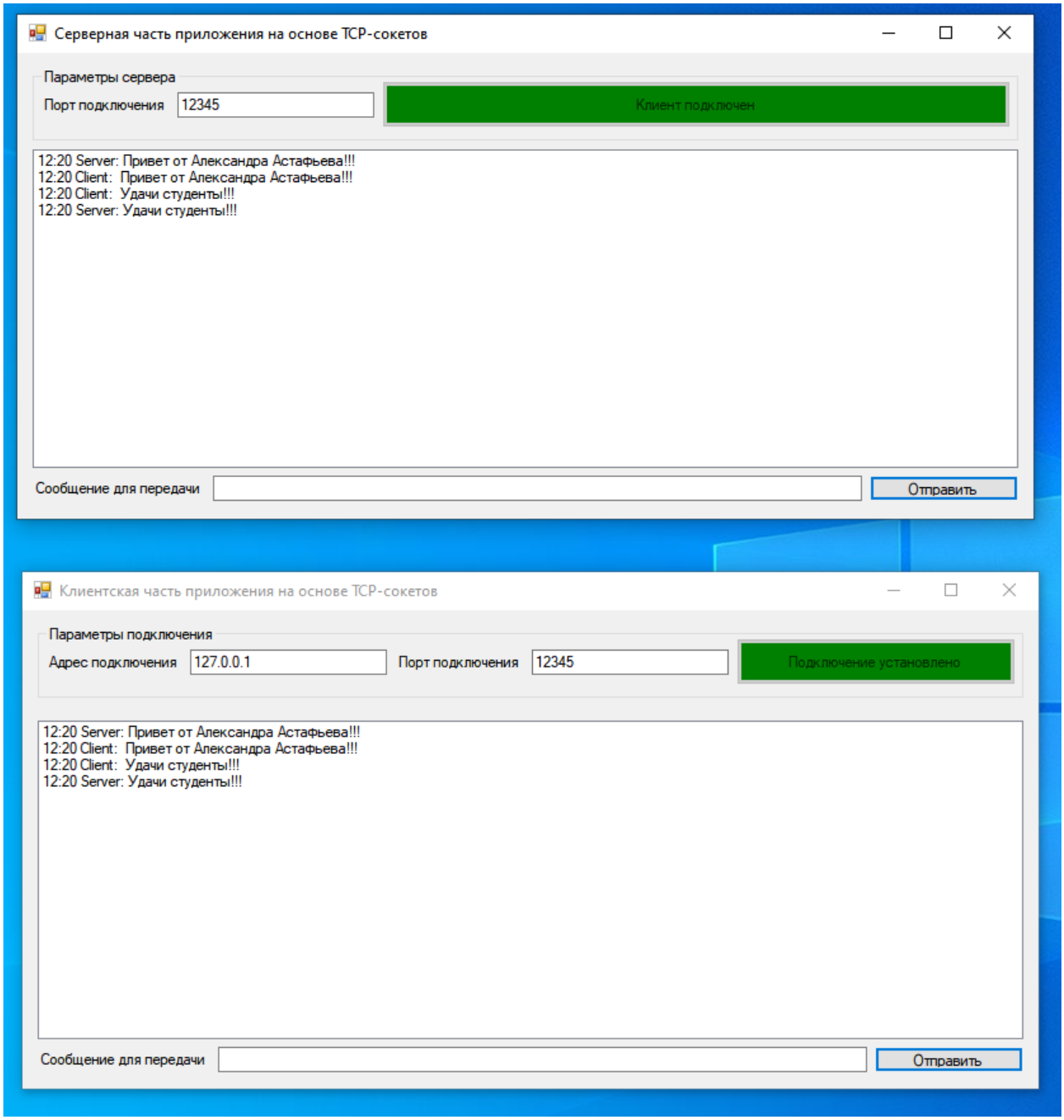

Рис. 4. Пример работы клиент-серверного приложения

Перехват сетевого трафика

WireShark - программа-анализатор трафика для компьютерных сетей Ethernet и некоторых других.

Для начала получения сетевого трафика будем использовать программу WireShark. Интерфейс программы приведен на рисунке 5.

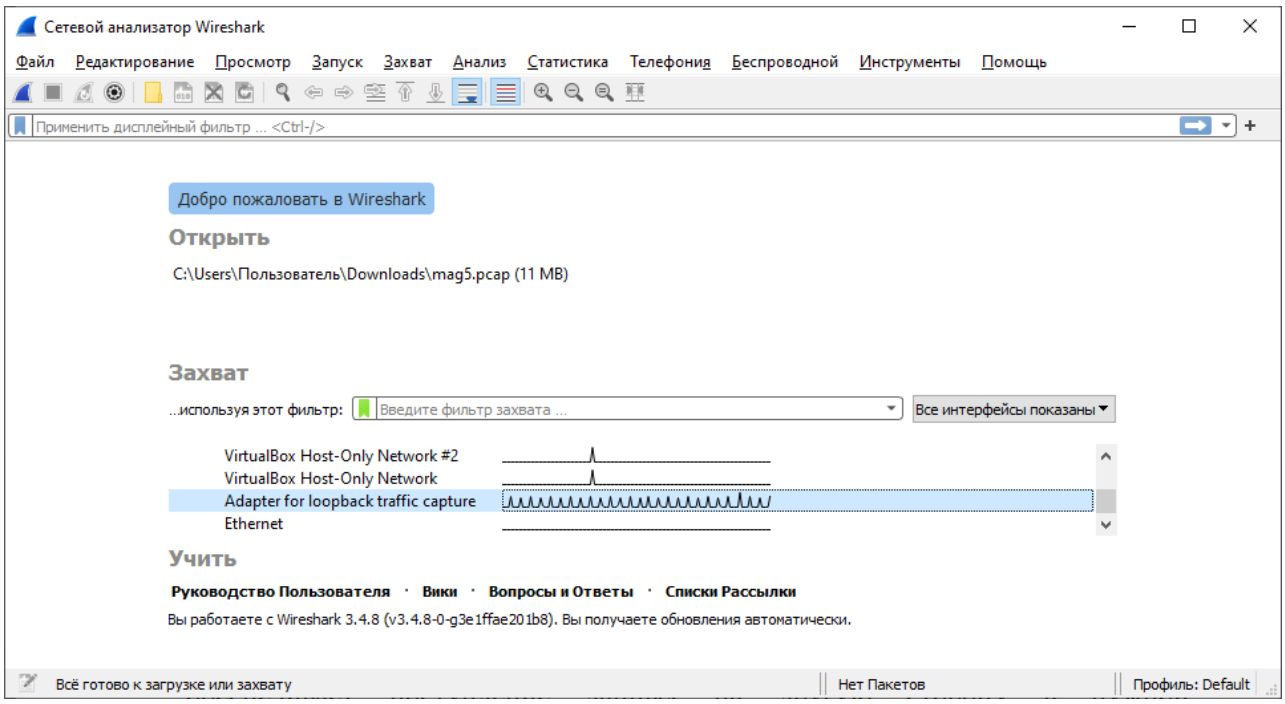

Рис.5. Интерфейс программы WireShark

В части интерфейса «Захват» в программе WireShark выводится список сетевых интерфейсов, с помощью которых возможно произведение захвата сетевого трафика. Список интерфейсов должен совпадать со списком сетевых подключений операционной системы и специализированными интерфейсами. Список сетевых подключений ОС приведен на рисунке 6.

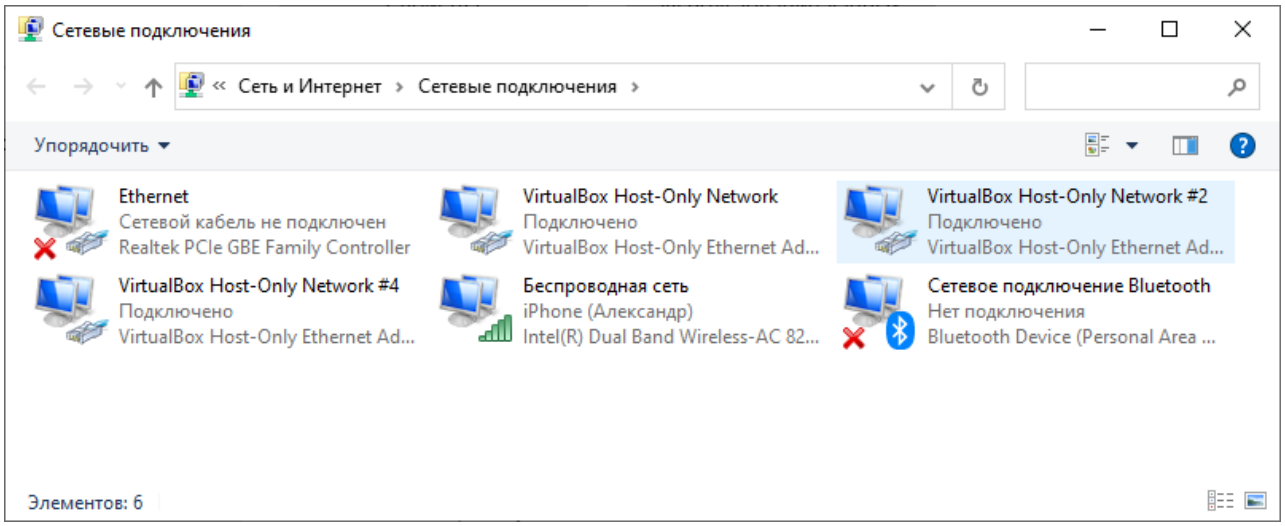

Рис.6. Список сетевых интерфейсов ОС

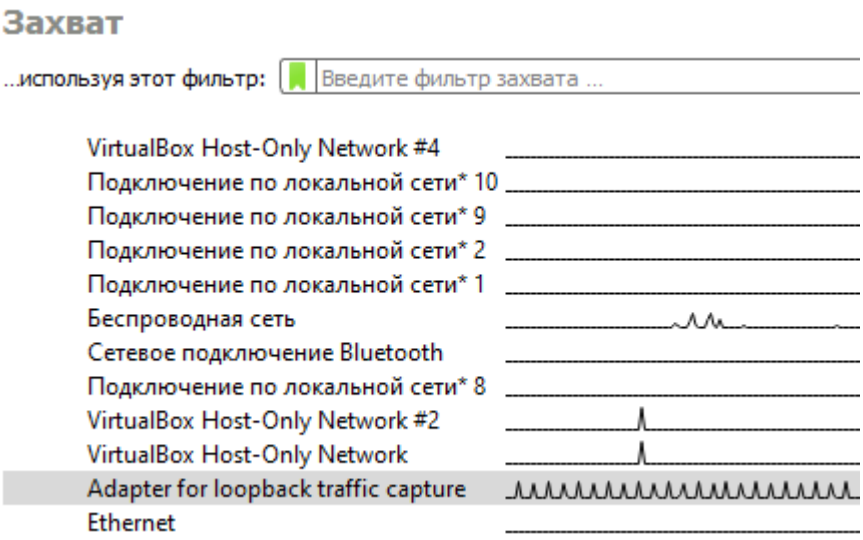

Рис.7. Список интерфейсов захвата

Помимо интерфейсов ОС в списке присутствует такой интерфейс, как «Adapter for loopback traffic capture». Если вы пытаетесь перехватить трафик с компьютера на себя, этот трафик не будет передаваться по реальному сетевому интерфейсу, даже если он отправляется по адресу на одном из сетевых адаптеров компьютера. Это означает, что вы не увидите его, если попытаетесь захватить, например, интерфейсное устройство адаптера, которому назначен адрес назначения. Вы увидите его только в том случае, если сделаете захват трафика на "Adapter for loopback traffic capture", если такой интерфейс существует и на нем можно осуществить захват. Проще говоря, весь трафик, направленный на адреса 127.0.0.1 и localhost будут видны именно на этом интерфейсе.

Для того, чтобы начать захват сетевого трафика необходимо выбрать интерфейс и кликнуть на него два раза. После этого программа переключится в режим захвата трафика. Пример начала захвата трафика приведен на рисунке 8.

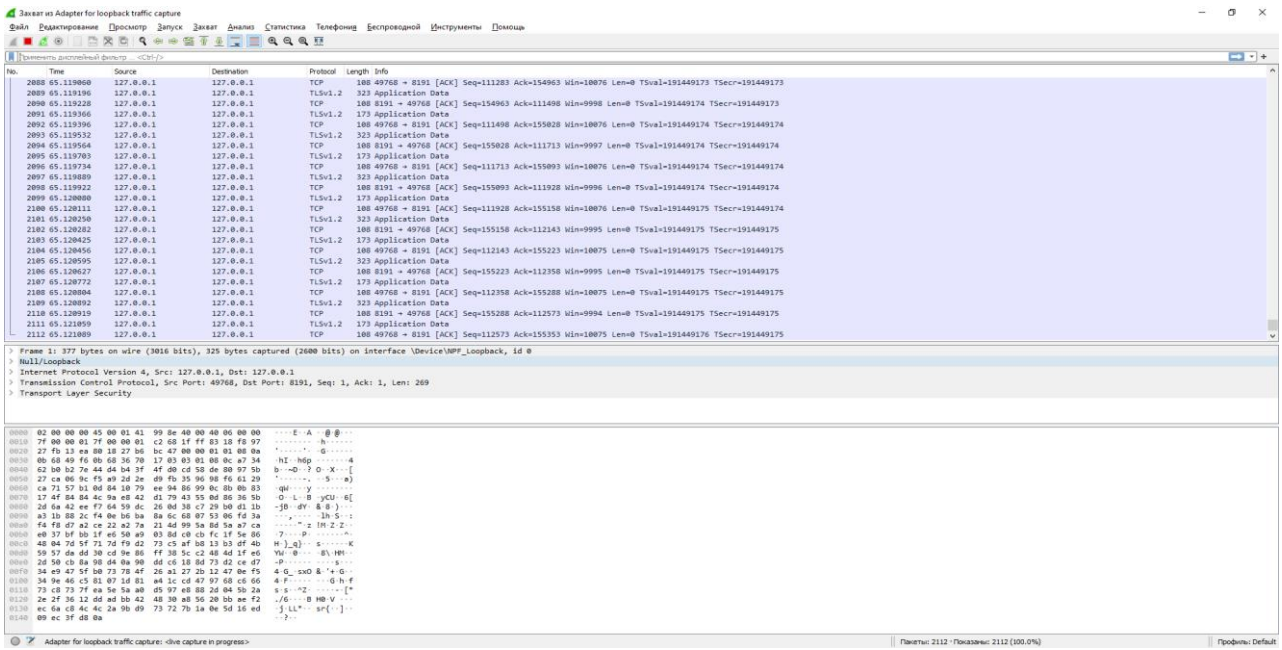

Рис. 8. Пример начала захвата трафика

Окно программы разделяется на 3 части. Верхняя часть визуализируется список захваченных пакетов. Средняя часть выводит выбранный захваченный пакет согласно его структуре. Нижняя часть показывает состав передаваемых данных.

Для первого эксперимента запустим приложение, разработанное на предыдущем занятии два раза: в режиме клиента и сервера. Создадим соединение по адресу 127.0.0.1. Таким образом, весь трафик, передаваемый между этими приложениями, будет захвачен на интерфейсе «Adapter for loopback traffic capture». Пример захвата трафика приведен на рисунке 9.

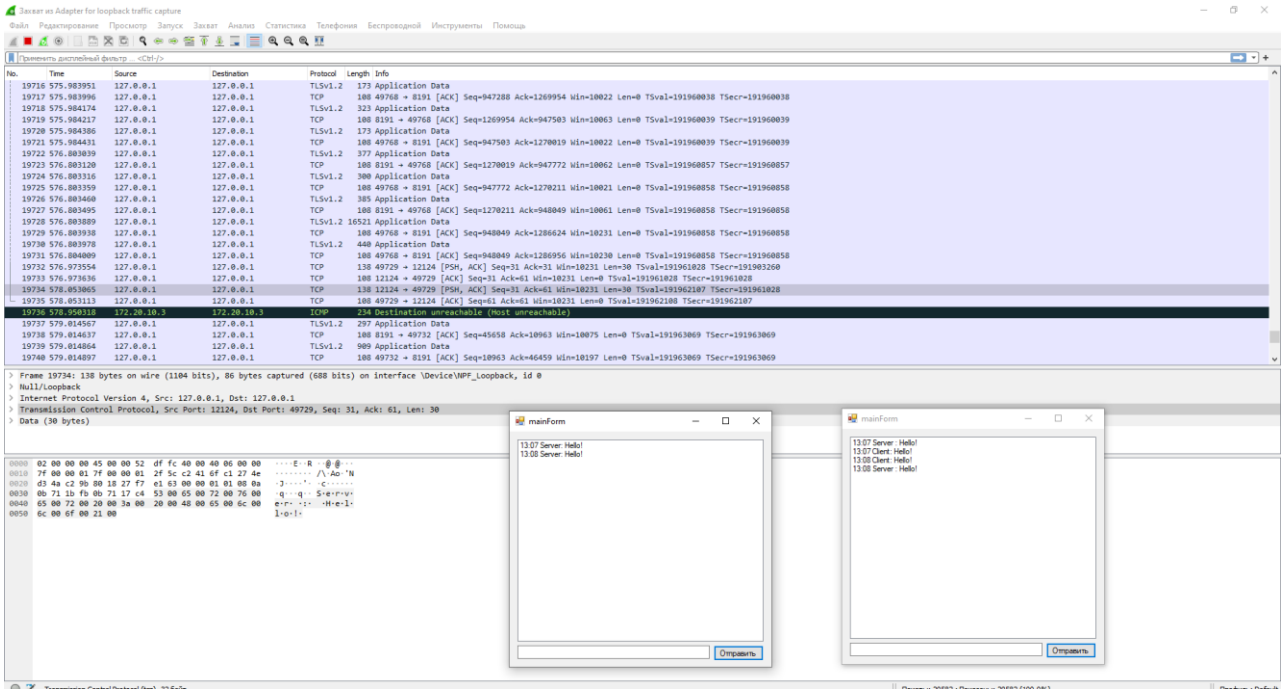

Рис.9. Пример захвата трафика

Таким образом, после передачи сообщений с сервера на клиент и в

обратную сторону в программе WireShark появляются соответствующие записи, как это показано на рисунке 10.

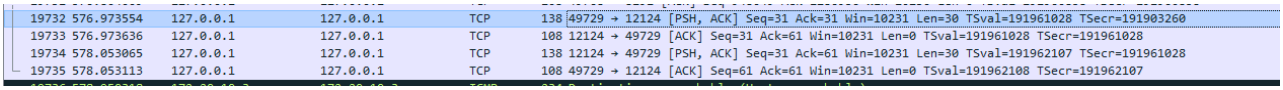

Рис.10. Пакеты трафика

Верхние 2 строки соответствуют передаче сообщения Hello! От клиента к серверу. Состав пакета приведен на рисунке 15.

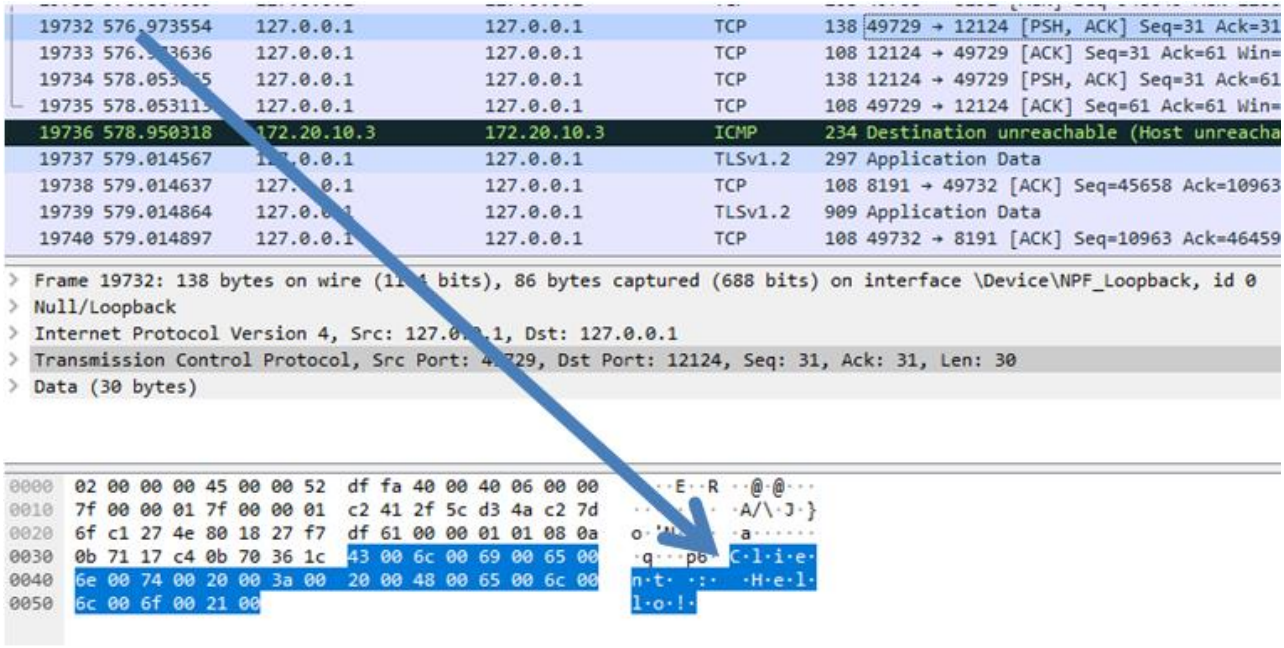

Рис. 11. Сообщение от клиента к серверу

Нижние 2 строки соответствуют сообщению Hello! От сервера к клиенту. Состав пакета приведен на рисунке 12.

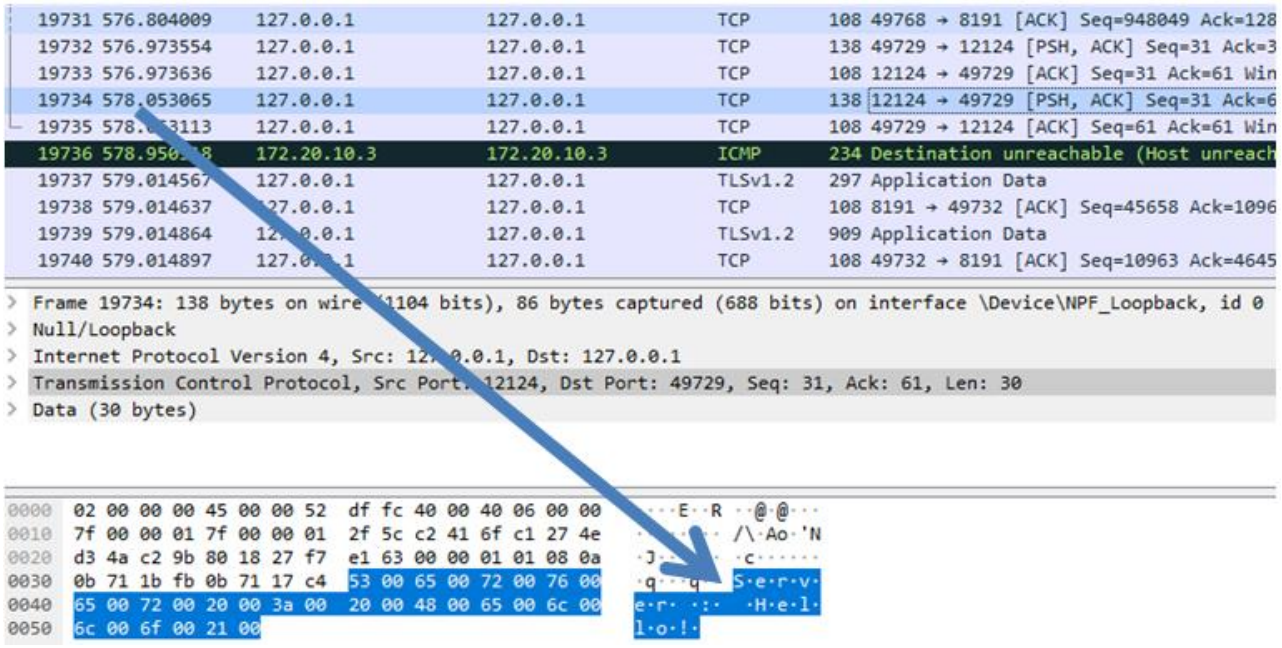

Рис. 12. Сообщение от сервера к клиенту

Так как в ходе работы приложений на базе сокетов происходит постоянный обмен информацией, то записей при захвате трафика, появляется

огромное количество. Для того, чтобы производить поиск необходимых пакетов, их необходимо проанализировать.

Исходя из того, что приложение для передачи данных было создано в рамках предыдущей работы можно описать характер передаваемой информации. Передаваемая информация включает в себя текстовую информацию. Исходя из того, что при захвате трафика были определены записи, содержащие данную информацию можно рассмотреть их более детально. На первый взгляд, записи отличаются наличием записи [PSH, ACK] (рисунок 13).

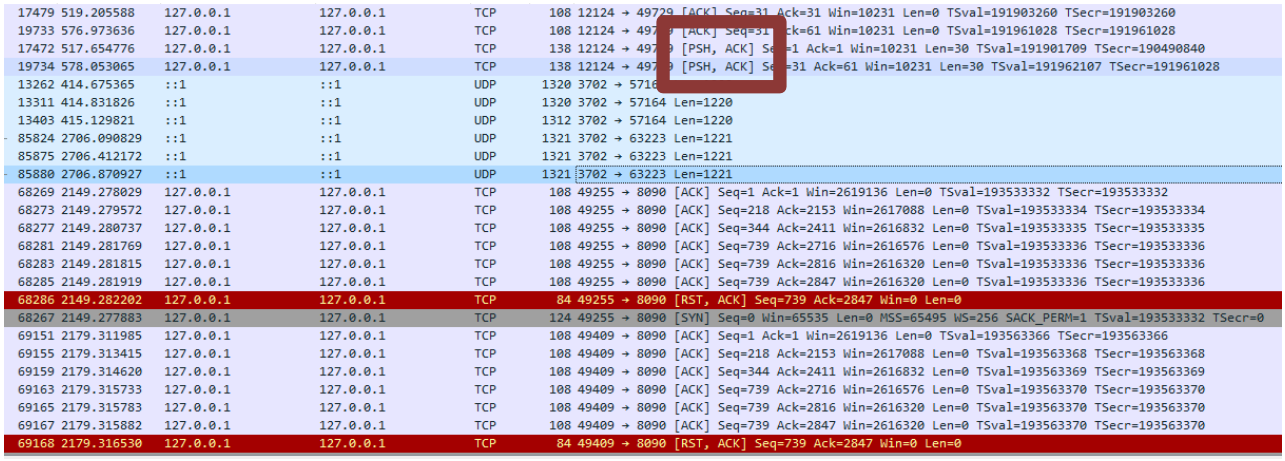

Рис. 13. Флаги PSH, ACK в записях трафика

Если обратиться к документации WireShark, то можно найти информацию, что

Флаг ACK означает, что машина, отправляющая пакет с ACK, подтверждает данные, которые она получила от другой машины. В TCP, как только соединение установлено, все пакеты, отправленные любой стороной, будут содержать ACK, даже если это просто повторное подтверждение данных, которые уже были подтверждены.

Флаг PSH - это указание отправителя на то, что, если реализация TCP принимающей машины еще не предоставила полученные данные коду, считывающему данные (программе или библиотеке, используемой программой), он должен сделать это в этот момент. Процитируем RFC 793, официальную спецификацию для TCP:

Данные, которые передаются по соединению, можно рассматривать как поток октетов. Отправляющий пользователь указывает в каждом вызове ОТПРАВКИ, следует ли немедленно передавать данные этого вызова (и любых предшествующих вызовов) получающему пользователю с помощью установки флага PUSH.

Для анализа данных сетевого трафика в программе WireShark используются «Дисплейные фильтры». Работа с дисплейными фильтрами производится либо через специальный интерфейс, который находится Панель

управления – Анализ – Дисплейные фильтры (Рисунок 14). Либо применять их напрямую из строки поиска, как показано на рисунке 15.

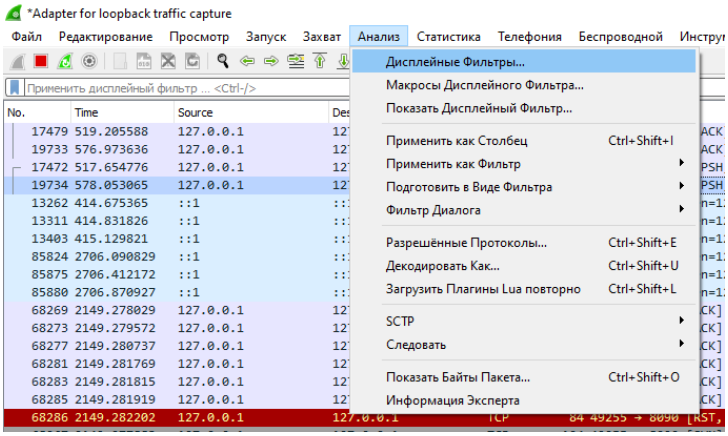

Рис. 14. Дисплейные фильтры

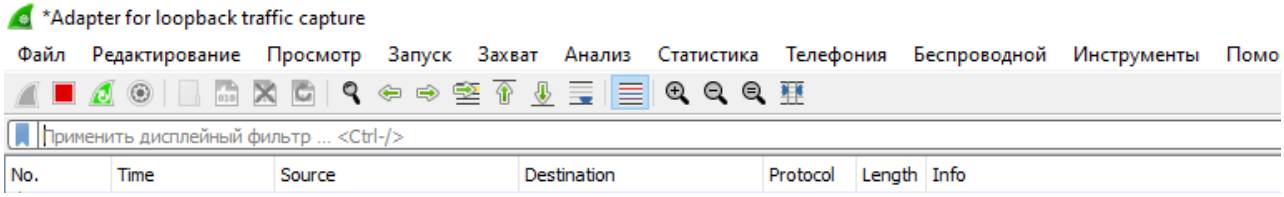

Рис. 15. Строка поиска для работы с дисплейными фильтрами

Программа WireShark уже имеет дисплейные фильтры по умолчанию. Их список приведен на рисунке 16.

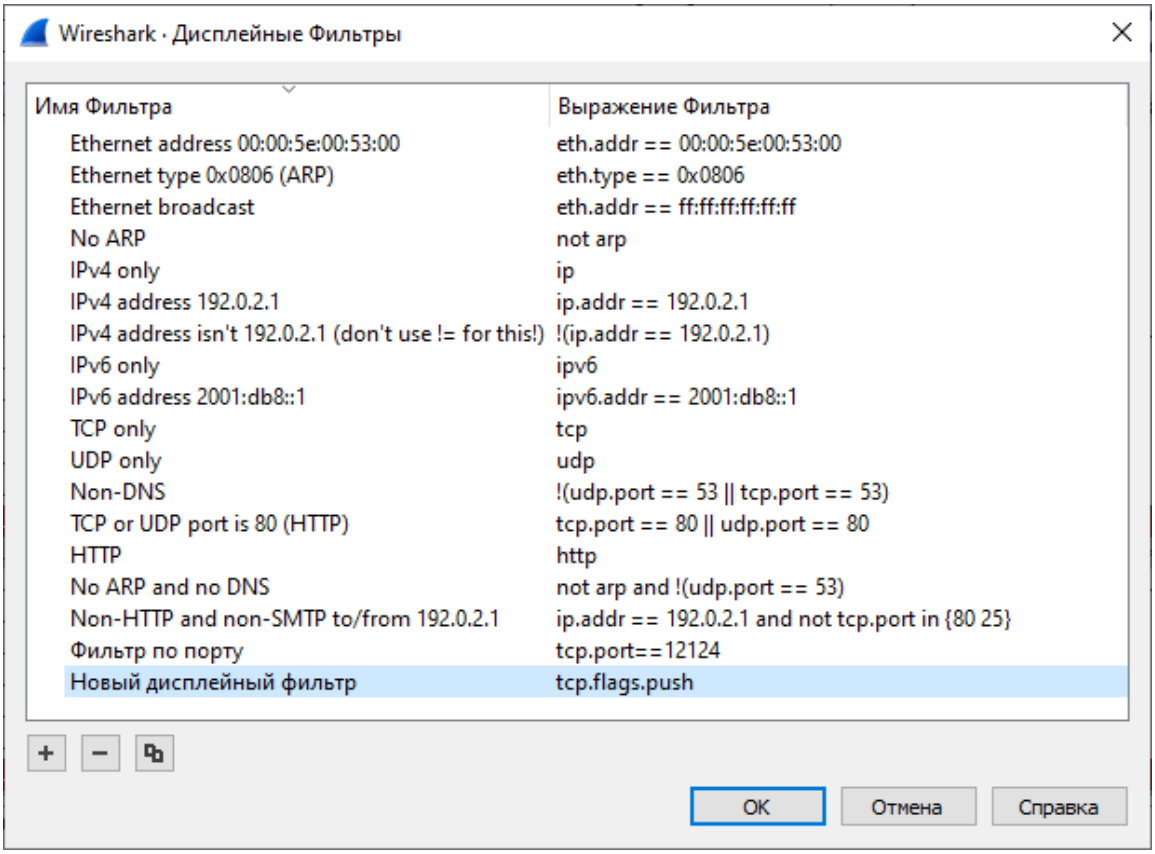

Рис. 16. Дисплейные фильтры по умолчанию

Помимо фильтров по умолчанию можно добавлять собственные фильтры, для их дальнейшего использования. Для создания фильтров можно

использовать команды, приведенные в таблице 1.

Таблица 1. Команды для создания дисплейных фильтров.

Команда	**Значение**	**Пример использования**
==	равенство	ip.dst == 193.168.3.10
!=	Не равно	udp.dst != 53
<	меньше чем	ip.ttl < 24
>	больше чем	frame.len > 10
<=	меньше или равно	frame.len <= 0x20
>=	больше или равно	tcp.analysis.bytes_in_flight >= 1000
matches	регулярные выражения	frame matches "[Pp][Aa][Ss][Ss]"
contains	содержит	dns.resp.name contains google

В нашем случае, для поиска записей трафика с сообщениями чата достаточно двух фильтров:

1. tcp.flags.push – наличие флага PSH.
2. tcp.port - номер порта сокета, в примере это 12345.

Таким образом, результирующий фильтр будет иметь следующий вид:

tcp.flags.push and tcp.port == 12345

Результат фильтрации по этому фильтру приведен на рисунке 17.

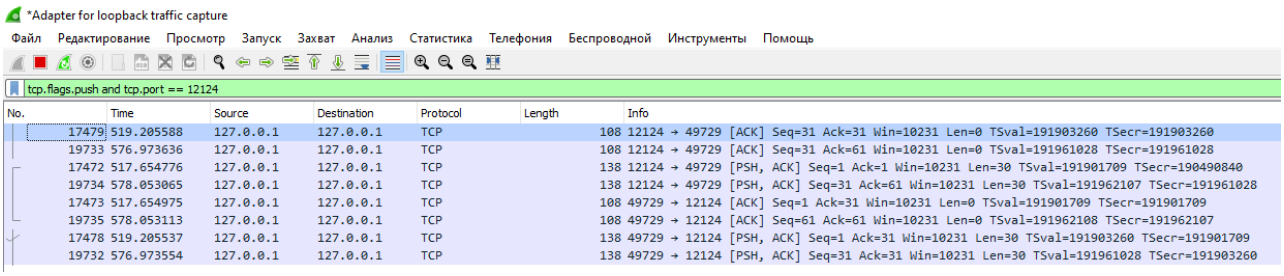

Рис. 17. Результат фильтрации

Задание на лабораторную работу

Используя в качестве механизма сетевого взаимодействия приложение, реализованное в прошлой работе, произвести сбор и анализ сетевого трафика:

1. Реализовать формат передачи данных, по следующему шаблону: *<ФИО_студента>Передаваемые данные</ФИО_студента>*. Например: *<AstafievAV>Hello World!</AstafievAV>*.
2. Произвести захват сетевого трафика с использованием программы WireShark по локальному интерфейсу «Adapter for loopback traffic capture».
3. Создать дисплейный фильтр для фильтрации сообщений, передаваемых приложениями на базе сокетов.
4. Оформить отчёт по проделанной работе с фиксацией основных моментов работы. В отчёте должны быть представлены сообщения и первого и второго абонентов.