**BC/NW 2021№ 2 (38):6.1**

**ИССЛЕДОВАНИЕ ЗАВИСИМОСТИ ВРЕМЕНИ ПЕРЕДАЧИ ДАННЫХ ПО СЕТИ ОТ ИСПОЛЬЗУЕМЫХ ПРОТОКОЛОВ И ПРОГРАММНЫХ ИНТЕРФЕЙСОВ**

Абросимов Л.И., Стеклов В.В.

В настоящее время огромное количество устройств общаются между собой, используя сеть передачи данных. Участники сетевого взаимодействия обязаны согласовать друг с другом множество правил, которые объединяются в протоколы передачи данных. Протоколы нижних уровней сетевой модели OSI часто реализуются с использованием программных и аппаратных средств.[1] Все протоколы, начиная с транспортного уровня и выше, реализуются программными средствами.[2]

В наше время актуальна проблема быстрой и надежной передачи информации.

Повысить эффективность транспорта данных достаточно сложно. В зависимости от используемых транспортных протоколов время и качество доставки информации может сильно отличаться.

В настоящей статье определяются зависимости времени передачи данных от используемых протоколов транспортного и прикладного уровней. Эффективность передачи информации по сети во многом зависит от класса передаваемой информации. В статье рассматривается передача графических изображений между двумя компьютерами, находящимися в одной локальной сети.

 В разрабатываемой модели передачи данных для определениях качества обслуживания определяются качественные оценки обслуживания. Чаще всего используются следующие показатели работы сети:[1]

— скорость передачи данных;

— задержки передачи пакетов;

— уровень потерь и искажений пакетов.

На основании этих показателей осуществляется выбор наилучшего метода передачи информации.

Простейшая модель передачи данных реализуется на языке программирования C# с использованием технологий библиотек .NET. В работе также исследуется роль сжатия графической информации для её эффективной передачи по каналам связи.

В процессе работы были рассмотрены основные протоколы передачи данных по сети, входящие в библиотеку .

В настоящей статье рассмотрено взаимодействие двух компьютеров, находящихся в одной локальной сети. Работа программы не зависит от типа передаваемых файлов. Интернет – взаимодействие строится с использованием реализаций TCP и UDP протоколов в .NET.

Таким образом, в результате на данном этапе мы получаем программу, которая реализует взаимодействие нескольких компьютеров, используя протоколы TCP и/или UDP, позволяющую передавать разнотипные файлы по локальной сети. При тестировании программы будем передавать разные файлы и фиксировать время их доставки.

1. **Разработка модели передачи данных**

В процессе разработки модели передачи данных создаются две программы, которые располагаются на разных компьютерах. Программы создаются в виде консольного приложения, причем при запуске программ появляется специальное меню, в котором можно выбрать тип подключения. Для правильного взаимодействия между устройствами требуется выбирать согласованный тип протокола: или UDP, или TCP.

Программа клиента осуществляет считывание файла из файловой системы и отправку его в сеть. Сервер ожидает подключение от клиента, после этого принимает файл и сохраняет его у себя в дисковом пространстве. Алгоритмы методов приема и передачи файлов у клиента и сервера представлены ниже.

Программы разрабатываются на языке C#. В программе клиента определим базовый абстрактный класс Client, который определяет интерфейс для будущих наследников, собственных реализаций TcpClient и UdpClient. Создание базового класса вместо интерфейса в данном случае оправдано тем, что сокращает дублирование повторяющегося кода.

using System;

using System.Net;

namespace Client.Clients

{

 public abstract class Client

 {

 protected readonly IPAddress serverAddress;

 protected readonly int serverPort;

 protected int bufferSize = 512;

 public Client(string serverAddress, int serverPort)

 {

 if (!IPAddress.TryParse(serverAddress, out var iPAddress))

 {

 throw new InvalidOperationException(

 $"String {serverAddress} is not ip address");

 }

 this.serverAddress = iPAddress;

 this.serverPort = serverPort;

 }

 public void SetBufferSize(int bufferSize) =>

 this.bufferSize = bufferSize;

 public abstract void SendFile(string fileName);

 }

}



Рис 1. Алгоритм передачи файла для клиента

Клиент обязательно должен иметь адрес и номер порта сервера, к которому он хочет подключиться.[1] Эти поля устанавливаются в конструкторе базового класса клиента. Предложенный алгоритм передачи данных подразумевает разбиение исходного файла на части. Программа считывает часть файла в специальный буфер, размер которого определяется переменной bufferSize. Далее происходит отправка этого буфера в сеть. Данный процесс происходит в цикле, пока весь файл не будет полностью считан. Соответствующий алгоритм считывания и передачи файла в программе клиента приведен выше. Причем данный алгоритм разрабатывается и для TcpClient, и для UdpClient.



Рис 2. Алгоритм приема файла для сервера

Алгоритм приема файла для сервера диаметрально противоположен с алгоритмом клиента. Вначале происходит считывание потока байтов в класс FileInfo. Этот класс содержит в себе информацию о передаваемом файле (его имя и размер). Далее происходит поблочное считывание файла и загрузка блоков данных в файловый поток, при этом файловый поток создает новый файл на диске сервера.

***1.1 Реализация UDP - взаимодействия***

UDP — это протокол эффективной передачи данных в удаленный узел. UDP не предусматривает установление соединения. Доставка пакетов, отправляемых в удаленную конечную точку, не гарантируется так же, как не гарантируется их очередность при приёме. Обычно приложения, использующие протокол UDP, должны самостоятельно обрабатывать отсутствующие, повторяющиеся и идущие не по порядку пакеты, т.к. реализация протокола не подразумевает использование сигналов подтверждения как в TCP.[2] Но в разрабатываемой программе мы не будем реализовывать данную функциональность.

Класс UdpClient предоставляет простые методы для отправки и получения датаграмм UDP без подключения в блокирующем синхронном режиме. Так как UDP является транспортным протоколом без подключения, устанавливать подключение к удаленному узлу перед отправкой и получением данных не требуется.[3] То есть вызов метода Connect для UDP не происходит, как это делается, например, в TCP. Приведем пример реализации метода SendFile. Реализация метода удовлетворяет алгоритму клиента, который был представлен выше.

public override void SendFile(string fileName)

{

 using (var inputFileStream = new FileStream(fileName, FileMode.Open))

 {

 SendFileInfo(inputFileStream);

 SendFile(inputFileStream);

 }

 udpClient.Close();

}

private void SendFileInfo(FileStream fileStream)

{

 var binaryFormatter = new BinaryFormatter();

 var fileInfo = new WebProtocolsModel.FileInfo

 {

 FileName = fileStream.Name,

 FileSize = fileStream.Length

 };

 var stream = new MemoryStream();

 binaryFormatter.Serialize(stream, fileInfo);

 stream.Position = 0;

 var bytes = new byte[stream.Length];

 stream.Read(bytes, 0, Convert.ToInt32(stream.Length));

 udpClient.Send(bytes, bytes.Length, iPEndPoint);

}

private void SendFile(FileStream fileStream)

{

 using (var binaryReader = new BinaryReader(fileStream))

 {

 byte[] buffer = new byte[bufferSize];

 int count;

 while ((count = binaryReader.Read(buffer, 0, bufferSize)) > 0)

 {

 udpClient.Send(buffer, count, iPEndPoint);

 }

 }

}

В начале метода SendFile происходит получение читающего потока для файла с полным именем filename, которое передается в параметрах. Далее происходит отправка служебной информации о файле. Информация о файле сериализуется в промежуточный поток с помощью двоичного преобразователя. Для отправки данных в сеть используется метод Send, который принимает массив байтов, полученный из промежуточного потока.

После передачи служебной информации о файле происходит отправка самого файла. Для этого из файла с помощью экземпляра класса BinaryReader происходит последовательное считывание блоков байтов и передача их в сеть.

На стороне сервера используется также экземпляр класса UdpClient. Считывание данных осуществляется с помощью метода Receive, который блокирует поток программы. Блокировка происходит пока следующий блок данных не придет на сервер. Реализация класса UdpServer представлена в приложении А.

***1.2 Реализация TCP - взаимодействия***

Протокол TCP – это сетевой протокол транспортного уровня, ориентированный на соединение между двумя хостами. Для установки соединения TCP использует механизм “тройного рукопожатия”, который заключается в последовательном обмене между устройствами специальных пакетов. Протокол TCP гарантирует целостность передаваемых данных, устраняя дублирование пакетов и сохраняя порядок их отправки.[2]

Чтобы передать данные по сети с использованием протокола TCP, в .NET используется класс TcpClient. Данный класс также требует адрес сетевого устройства, в котором размещается нужная служба, и порт, который служба использует для обмена данными.[3]

Сервер в свою очередь использует класс TcpListener. Данный класс используется для отслеживания входящих запросов. Метод  Start  включает прослушивание порта, а метод  Stop  отключает его. Метод  AcceptTcpClient  блокирует поток исполнения, принимает входящие запросы на подключение и создает экземпляр TcpClient для их обработки.

private WebProtocolsModel.FileInfo GetFileInfo(

 NetworkStream networkStream)

{

 var binaryFormatter = new BinaryFormatter();

 var fileInfo = (WebProtocolsModel.FileInfo)binaryFormatter.Deserialize(networkStream);

 fileInfo.FileName = fileInfo.FileName.Replace(

 oldValue: "Client",

 newValue: "Server");

 return fileInfo;

}

private void SaveFile(

 NetworkStream networkStream,

 WebProtocolsModel.FileInfo fileInfo)

{

 using (var binaryWritter = new BinaryWriter(

 newFileStream(fileInfo.FileName, FileMode.Create)))

 {

 byte[] buffer = new byte[bufferSize];

 for (int i = 0; i < fileInfo.FileSize; i += bufferSize)

 {

 networkStream.Read(buffer, 0, bufferSize);

 binaryWritter.Write(buffer);

 }

 }

}

Как видно из примера, принцип работы с протоколом TCP несколько отличается от UDP в .NET. Реализация TCP подразумевает работу с непрерывным потоком байтов. Для TcpClient мы получаем соответствующий ему сетевой поток (в коде это объект класса NetworkStream) и с помощью методов Read и Send, характерных для потоков, осуществляем чтение и запись данных.

**2. Результаты работы и тестирование программы**

Программы клиента и сервера выполнены в виде консольного приложения. Интерфейсы этих программ представлены на рисунке 3 и 4 соответственно. Как было сказано ранее, для правильного взаимодействия требуется выбирать один и тот же тип подключения у сервера и у клиента.



Рис 3. Пользовательский интерфейс программы клиента



Рис 4. Пользовательский интерфейс программы сервера

В предложенном алгоритме данные считываются в виде массива байт. Работа с файлами и с сетью осуществляется с помощью вспомогательных классов-преобразователей, которые форматируют исходные данные в байты информации. Таким образом, программа не зависит от типа передаваемого файла. Одинаковым образом будут передаваться как изображения, так и pdf, doc – документы.

В ходе тестирования будем передавать разные типы файлов: текстовые документы в форматах txt, doc, а также изображения в форматах png и jpg. Будем фиксировать время передачи данных у клиента и у сервера. На стороне сервера также будем проверять целостность передаваемых данных. Результаты тестирования представлены в таблице 1.

Таблица 1. Результаты тестирования программы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Тип файла | Размер файла(байты) | Тип подключения | Время клиента(мс) | Время сервера(мс) |
| Текстовый документ в формате txt | 3 605 | TCP | 6 | 12 |
| Текстовый документ в формате txt | 3 605 | UDP | 3 | 7 |
| Текстовый документ в формате doc | 147 773  | TCP | 22 | 30 |
| Текстовый документ в формате doc | 147 773  | UDP | 6 | - |
| Изображение png | 621 004  | TCP | 45 | 48 |
| Изображение png | 621 004  | UDP | 20 | - |
| Изображение jpg | 755 637  | TCP | 51 | 60 |
| Изображение jpg | 755 637  | UDP | 33 | - |
| Изображение jpg | 1 102 805  | TCP | 58 | 65 |
| Изображение jpg | 1 102 805  | UDP | 32 | - |

**3. Заключение**

Таким образом, в данной разработке была создана программа, реализующая сетевое взаимодействие между двумя устройствами через протоколы TCP и UDP. Для сервера и клиента были предложены и реализованы алгоритмы передачи данных, которые подразумевают чтение, запись, передачу байтовых блоков.

Были показаны сходства и различия в реализации TCP и UDP протоколов в .NET. Например, передача данных по TCP протоколу осуществляется с помощью объекта сетевого потока. Для UDP взаимодействия в .NET отсутствуют многие классы и методы, которые есть в TCP (например, класс TcpListener).

В ходе тестирования был получен ожидаемый результат – с увеличением размера передаваемого файла, происходит увеличение времени его передачи. Передача данных через протокол UDP осуществляется значительно быстрее, чем через TCP.

При тестировании также проверялась целостность переданных файлов на сервере. Было замечено, что целостность файла при его передаче через протокол UDP теряется. Ниже представлено изображение исходного файла, изначально находящегося на клиенте, который впоследствии передается на сервер.



Рис 5. Пример исходного передаваемого файла.

Использование TCP гарантирует целостность передаваемых данных. Полученный результат на сервере соответствует исходному файлу. Но, как можно заметить, переданный файл через UDP протокол является искаженным. Таким образом, несмотря на скорость передачи, для UDP соединения требуется дополнительные методы, позволяющие достигать целостности передаваемых данных.[3] Из таблицы 1 также видно, что время работы сервера для UDP соединения не определено для больших файлов. Это связано с тем, что сервер со временем блокируется в ожидании потерянных датаграмм.



Рис 6. Переданные изображения (слева для TCP, справа для UDP подключений).

Можно заметить, что при увеличении размера считывающего/записывающего буфера время работы алгоритмов может быть изменено. Для больших файлов ожидается меньшее количество итераций считывания данных, вследствие чего время алгоритма будет уменьшаться. Для малых файлов изменения не будут заметны. Однако создание слишком большого буфера может повлечь за собой то, что избыточные байты в пакете будут отбрасываться.

Таким образом, несмотря на то, что целостность небольших файлов при передаче через UDP протокол не терялась, для описанных типов файлов рекомендуется использовать все же протокол TCP.

**Список использованных источников**

1. Руководство по сетевому программированию в С# и .NET. [Электронный ресурс]. – Режим доступа: https://metanit.com/sharp/net/, свободный – (10.06.2020).

2. Эндрю Кровчик, Винод Кумар. .Net. Сетевое программирование для профессионалов. : Пер. с англ. — СПб. : ООО “Издательство Лори”, 2007 — 417с.

3. Сетевое программирование в .NET Framework. [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/framework/network-programming/, свободный – (10.06.2020).

4. Олифер В. Г., Олифер Н. А. Компьютерные сети. Принципы, технологии, протоколы: Учебник для вузов. 4-е изд. — СПб.: Питер, 2010. — 944 е.: ил.

5. Руденков Н.А., Долинер Л.И. Основы сетевых технологий: Учебник для вузов. Екатеринбург: Изд-во Уральского. Федерального ун-та, 2011. – 300 с.

6. Руководство по сетевому программированию в С# и .NET. [Электронный ресурс]. – Режим доступа: https://metanit.com/sharp/net/,  свободный – (10.06.2021).

7. Эндрю Кровчик, Винод Кумар. .Net. Сетевое программирование для профессионалов. : Пер. с англ. — СПб. : ООО “Издательство Лори”, 2007 — 417с.

8. Сетевое программирование в .NET Framework. [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/framework/network-programming/,  свободный – (10.06.2021).

9. Реализация Reliable Udp протокола для .Net. [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/250227/,  свободный – (20.12.2021).