Communication Protocols Research Report

Mureşeanu,Gabriel A.G.

fhict sem3

Table of Contents

[0](#_Toc62147473)

[Introduction 2](#_Toc62147474)

[Methods 2](#_Toc62147475)

[Questions 3](#_Toc62147476)

[Results of Research 4](#_Toc62147477)

[The Connection 4](#_Toc62147478)

[How a connection works 4](#_Toc62147479)

[Packets 4](#_Toc62147480)

[Datagrams 5](#_Toc62147481)

[Streams 5](#_Toc62147482)

[Sockets 5](#_Toc62147483)

[Connection Types 5](#_Toc62147484)

[Transmission Control Protocol (TCP) 5](#_Toc62147485)

[User Datagram Protocol (UDP) 6](#_Toc62147486)

[Advantages and Disadvantages 6](#_Toc62147487)

[Best uses for TCP and UDP 8](#_Toc62147488)

[Best Protocol for Me 8](#_Toc62147489)

[Implementing the Connection 9](#_Toc62147490)

[Server socket 9](#_Toc62147491)

[Client Socket 10](#_Toc62147492)

[Transferring data over a connection 10](#_Toc62147493)

[Recommendations 11](#_Toc62147494)

[References 11](#_Toc62147495)

# 

# Introduction

In this paper I will showcase the results of the research that I have conducted on Application Communication.

Communication Protocols are the key factor to the online landscape that we live in, we use them on a daily basis for basic actions such as sending an E-mail, communication through social apps, and most recently for online courses and university.

My first time working with Communication Protocols was a simple connection between two computers on the same network, however in this semester I have decided to take my knowledge to the next level and take the next big step – file transferring over two computers on different networks. Although the full implementation was unsuccessful, I think that the knowledge I have gathered over the four weeks I have been researching the topic is valuable enough to showcase in this paper.

# Methods

The methods that I used to gather data for this paper are the following:

* Self-Research Based on Documentation
* Scientific articles on the subject (See References for links)
* Studying and applying communication principles and practices.

# Questions

At the start of my research I have stumbled into many questions and problems that needed answers.

1. What is a connection?
   1. How does it work?
   2. What is a packet?
   3. What is a datagram?
   4. What is a stream?
   5. What is a socket?
2. What types of connection exist?
   1. What are the advantages and disadvantages?
   2. What are the best usages for them?
   3. Which one is the best for me?
3. How to implement the connection?
   1. How to implement a server socket?
   2. How to implement a client socket?
4. How to transfer data over a connection?

# Results of Research

## The Connection

The basic definition of a connection is to have a path of communication between two different computers, on the same or separate networks.

A basic connection can be achieved in two ways:  
-Using a cable (Ex: USB)  
-Remotely (Ex: Wi-fi, Bluetooth, Internet)

In the following points I will only talk about the remote type connection, specifically the same-network and separate networks ones.

### How a connection works

A same-network connection is based on ports, which are 16-bit unsigned integers ranging from 0 to 65535 that are used as a communication endpoint. Information is sent and received through them, both participants have to agree on said ports. (Note that in some cases the IP is also needed for a same-network connection, as multiple devices might be listening for data on the same port)

When it comes to a connection over different networks you will also need the IP address, which is a label assigned to each device connected to a network that uses the Internet Protocol to communicate. This form of communication also uses ports for sending and receiving data.

### Packets

IP packets have two components:

* Header

The header contains control information which is used to deliver the contents of the packet.

It contains the source and destination of the packet, error detection codes and in case of multiple packets – sequence information.

* Payload

The payload is the actual content of the packet, which is delivered based on the header control information. This information is delivered in bytes.

A packet guarantees the delivery, the arrival time, and the order of arrival of the data.

### Datagrams

The datagram is the more basic version of the packet. Having the same two components, both having the same functions, it does not guarantee the delivery, the arrival time, and the order of the data.

Datagram is also the name of the connection that uses the datagrams for sending data and the protocol used for it is called [UDP](#_User_Datagram_Protocol)

### Streams

A stream is a sequence of data that is processed as it is delivered instead of in batch.

In computing science, a stream is considered “codata”, meaning that it is potentially unlimited in size.  
Streams use [TCP](#TCP) to transfer data.

### Sockets

A socket is a software structure within a network that is used as an endpoint for sending and receiving data as bytes.

Sockets require the following data to function:

* IP address (or the localhost address in case of a same-network connection)
* Port number
* Data Transfer Protocol ([TCP](#TCP) or [UDP](#_User_Datagram_Protocol))

Sockets that use TCP are called Stream Sockets .

Sockets that use UDP are called Datagram Sockets.

## Connection Types

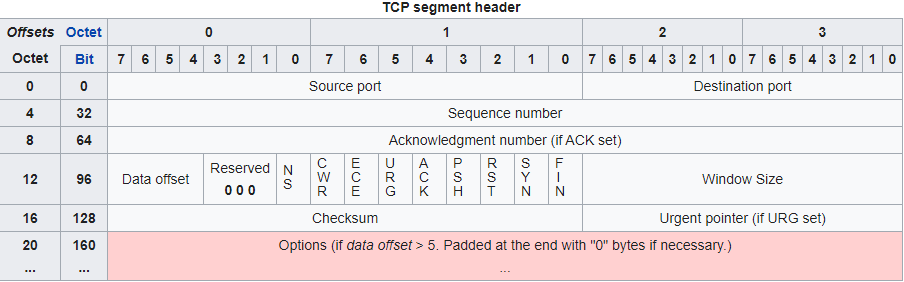
### Transmission Control Protocol (TCP)

Commonly referred to as TCP/IP, it provides a reliable connection that is ordered and error-checked using the Three-way Handshake. Because it checks all the data sent and received, it creates latency in the data sent.

TCP delivers data in streams of bytes called [packets](#_Packets)

TCP is a connection-oriented protocol, which requires a connection between the client and the server before being able to send data. The receiving end must be listening passively for a connection request before the connection is established

TCP is used by many internet applications, such as: P2P connections, World Wide Web, and media streams.

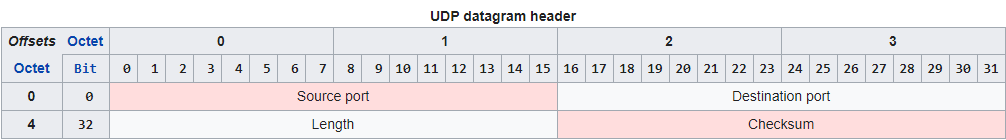


(The Byte information of a TCP [header](#_Packets) – credit: Wikipedia)

### User Datagram Protocol (UDP)

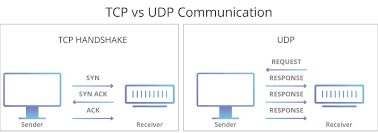
UDP provides no guarantee for the message delivery, order or errors, trading reliability for speed. The latency of the data sent is very low. UDP does not use any type of handshake.

UDP delivers data using datagrams of bytes called also referred to as datagrams.

UDP is a transaction-oriented protocol, suitable for simple communication where no checking is needed, especially when there are multiple clients. The lack of the handshake makes it suitable for real-time application such as voice-calls and online game servers.  
  


(The Byte information of a UDP [header](#_Packets) – credit:Wikipedia)

### Advantages and Disadvantages

 (Image credit: Cloudflare)

TCP is a connection-oriented protocol and requires handshaking to set up end-to-end communications. Once a connection is set up, user data may be sent bi-directionally over the connection.

* *Reliable* – TCP manages message acknowledgment, retransmission, and timeouts. Multiple attempts to deliver the message are made. If data gets lost along the way, data will be resent. In TCP, there is either no missing data, or, in case of multiple timeouts, the connection is dropped.
* *Ordered* – If two messages are sent over a connection in sequence, the first message will reach the receiving application first. When data segments arrive in the wrong order, TCP buffers the out-of-order data until all data can be properly re-ordered and delivered to the application.
* *Heavyweight* – TCP requires three packets to set up a socket connection, before any user data can be sent. TCP handles reliability and congestion control
* *Streaming* – Data is read as a byte stream; no distinguishing indications are transmitted to signal message (segment) boundaries.

UDP is a simpler message-based connectionless protocol. Connectionless protocols do not set up a dedicated end-to-end connection. Communication is achieved by transmitting information in one direction from source to destination without verifying the readiness or state of the receiver.

* *Unreliable* – When a UDP message is sent, it cannot be known if it will reach its destination; it could get lost along the way. There is no concept of acknowledgment, retransmission, or timeout.
* *Not ordered* – If two messages are sent to the same recipient, the order in which they arrive cannot be guaranteed.
* *Lightweight* – There is no ordering of messages, no tracking connections, etc. It is a very simple transport layer designed on top of IP.
* *Datagrams* – Packets are sent individually and are checked for integrity on arrival. Packets have definite boundaries which are honored upon receipt; a read operation at the receiver socket will yield an entire message as it was originally sent.
* *No congestion control* – UDP itself does not avoid congestion. Congestion control measures must be implemented at the application level or in the network.
* *Broadcasts* – being connectionless, UDP can broadcast - sent packets can be addressed to be receivable by all devices on the subnet.
* *Multicast* – a multicast mode of operation is supported whereby a single datagram packet can be automatically routed without duplication to a group of subscribers.

### Best uses for TCP and UDP

TCP is used for communication where data needs to be as safe as possible:

* Important file transfer (Ex: Download Servers, Online Storage)
* Web Browsing
* E-mails
* P2P connections (Ex: CO-OP Games)

UDP is used for communication where data does not have to be in a perfectly safe condition, but where the latency must be as low as possible:

* Audio and video live-streaming
* Online games (MMORPG, MoBA, RTS)
* Online Chatrooms (Facebook Messenger, WhatsApp)

### Best Protocol for Me

For the application that I was working on – a server that streams music to an application- , according to the prior point, UDP would seem like the best choice, however due to the small scale of the application I chose to go with TCP due to the following points:

* Datagram order control

Since UDP does not control data order, putting together the music file after transfer would require a complicated algorithm that would slow down the data transfer more than the TCP packets would have.

* No confirmation of received data

Since UDP does not confirm the data received, it means that in case of an error, the client has to do all of the error handling, which in this case means that a new call is required in order to get the missing data. While this is very hard to implement on such a volatile connection, it will also slow down the data transfer to the point where the latency would be bigger than a TCP connection.

* Small scale application with small file size

In a music streaming application, the files are kept as Mp3, which is very size efficient. The server only needs to send about 10Mb per each file. In this case TCP is better than UDP even for streaming.

## Implementing the Connection

Now that the basics have been explained, implementing a connection is very simple:

* TCP

For a TCP connection implement a socket on both devices, acquire the IP of both machines using whatever method you prefer, bind the sockets together and your connection is done.

* UDP

For a UDP connection implement a socket on your sender, and a listener on your receiver. Depending on what type of data and how you want to transfer it, an IP might be required.

### Server socket

For the next part of implementing a Client/Server socket I will assume the server is trying to send data over to the client.

* TCP

Create a socket that is going to listen to any connection calls. This can be done on an exact port or on the 11000 port (any incoming call will be randomly assigned a port) In case of a connection bind the socket to the received call. Now you can send data to the listening client.

In case of a disconnect from the client, you must handle the error and remove the client connection.

* UDP

Create a socket that is going to send data over a chosen port (or 11000); there is no need for binding the socket.

You can begin sending data straight away, however it is recommended to implement fail-safe methods and error handling due to the unreliability of the protocol.

### Client Socket

* TCP

For the client socket you must also create a socket, but instead of listening for an incoming call you need to call the IP of the server on the right port (or any in case the server is listening on 11000), after the server has successfully bonded the sockets, you can receive data through it.

In case of the server shutting down or a disconnect from the server, you must handle the error and close/reopen the connection.

* UDP

For an UDP socket receiving data is more straight forward, you just need a listener on the port on which the server sends data.

It is highly recommended to create an algorithm to sort the data and check for missing datagrams to make a call for receiving them again.

Since UDP does not use any type of connection, you do not need to handle disconnects.

## Transferring data over a connection

As mentioned in [Sockets](#_Sockets), they can only transfer data as bytes, so in order to transfer anything you first need to convert it into binary. This is simple for basic data, such as Strings, integers, and small text files.

After the conversion of the desired data, sending the data through the socket is done with a method in most languages. (Consult the documentation of the programming language that you use)

On the client side, the socket must be listening for the data. Once it has been received it must be saved in a byte array and converted back to the data/file it was before. Usually the information for the type of file it was are already in the byte array.

Sending files over a connection follows the same principle and simple data, however the size of the file needs to be considered.

For large files you must split them into smaller byte chunks and send them one at a time. For TCP this is very simple as all the data will be put back together automatically, but for UDP it requires more data manipulation on the receiving side.

# Recommendations

* For your project, always investigate what the best protocol is for you, while TCP and UDP are the base ones there are many edited versions of them that might do the trick for you.
* Remember to take into account what type of data you need to transfer when choosing a protocol, for example if you’re making a game think of the number of players that you’re going to have connected at one time. The netcode will have to prioritize getting the data to the players as fast as possible with minimal loses – in this case, UDP would be better for a large number of customers but TCP will do just as good with a smaller customer base.
* Always encrypt your data, you do not want files or important data being stolen.
* Make sure that code cannot be injected over a socket, always use tokens or another form of validation for data sent over sockets.
* Always check where the data is coming from, so you don’t receive false data or injected code.

# References

* <https://en.wikipedia.org/wiki/User_Datagram_Protocol>
* TCP/IP Sockets in C#: Practical Guide for Programmers (The Practical Guides)
* Microsoft Official Documentation for Communication Protocols (<https://docs.microsoft.com/en-us/>)
* <https://www.guru99.com/tcp-vs-udp-understanding-the-difference.html>
* <https://www.cloudflare.com/learning/ddos/glossary/user-datagram-protocol-udp/>
* <https://www.cloudflare.com/learning/ddos/glossary/tcp-ip/>
* https://www.otago.ac.nz/distance/otago021395.pdf