# Programmer friendly communication framework

1st Lilo Zobl
*Botball Team FrenchBakery*
*HTL Anichstraße*
Innsbruck, Austria
lzobl@tsn.at

2nd Matteo Reiter
*Botball Team FrenchBakery*
*HTL Anichstraße*
Innsbruck, Austria
mareiter@tsn.at

***Abstract*—This document will focus on an important topic in software development: Network communication. It will provide an insight into why what network communication is needed for and how it is typically realized. Additionally it will cover problems and annoyances with currently available communication frameworks for user-level applications as well as introduce concepts and a possible solution for the before-mentioned.**

## I. Introduction

For network to function, there needs to be at least two network capable devices.

When it comes to computer science and robotics, sooner or later, multiple computers need to communicate with each other. In many cases this communication is achieved over a computer network. Nowadays this is typically implemented using the internet protocol (IP).

### A. IP Networks

Without going into detail of how the Internet Protocol or any of it's underlying technologies work (out of the scope of this paper), this section provides a quick overview of IP.

In an IP network, every participating communication party (so called "host",such as a computer) is assigned an IP Address (either manually by the Administrator or using automated systems such as DHCP which will not be explained further). The IP address is the lowest level of addressing most user-level applications have to deal with. It is a 32-Bit number typically expressed by four octets (bytes) in dot notation: **192.168.1.1**

The IP address is split into two parts: the network address bits (some number of bits on the left of the address) and the host address bits (the rest of the bits on the right side). How many of the bits are part of the network and host area is variable and defined by a so-called netmask, which will not be discussed in more detail. All host with the same network-address-bits are considered to be on the same network and can therefore communicate with each other. Each host is uniquely identified in the network by the host-address-bits.

The Internet Protocol provides the foundations for transmitting data between hosts on this network (or possibly multiple networks): IP network packets. These contain Among other details, the sender and receiver addressing information and a block of data to be transmitted between the host. The operating system's network stack and other networking hardware then switch and route the packet from the sender to the desired destination host.

When developing applications however, the programmer almost never has to worry about the working of these underlying systems. Instead, they use higher level protocols build on top of IP, which abstract abstract IP packets and allow programmers to more conveniently achieve the desired functionality.

### B. TCP and UDP protocols

There are two main protocols on top of IP for user applications - TCP and UDP.

TCP stands for **Transmission Control Protocol** and is a connection based protocol, meaning there will be a standing connection between two hosts. Before any data can be sent over TCP, a connection has to be established during an initial handshake between both communication parties. This is done automatically in the background. Once established, a TCP connection provides a pipe-like environment for transmitting a continuous stream of data bytes. The stream is of undefined length and data sent by one host is guaranteed to be received in the same order as sent by the other (without missing bytes in between). However, since TCP "streams" bytes, it does guarantee that any block of bytes send is received in one piece.

UDP stands for **User Datagram Protocol** and is . The UDP protocol finds the destination device and sends the data inform of a datagram. There is nothing set in place to ensure that the datagram reaches the destination. There is also no process set in place that tells the receiver end in which order the packets were send. A datagram has a size restriction (65.535 bytes) which the packets gets split into. (Achtung!!!!!!!!!!!!!!!!!!!!!! wenn man mehrere geräte hat verwendet man oft eigentlich UDP da das schneller geht)

UDP is less reliable than TCP. Before choosing a respective protocol one has to decide if they need speed or reliability more. A common uses for UDP are videos whereas a common use for TCP are websites.

From here on, this paper will focus on point-to-point communication between two hosts, although many of the concepts described later are also applicable to multicast communication. For this reason, it will focus higher-level protocols and frameworks based on TCP from here-on out.

### C. Communication frameworks

While TCP communication is the basis of many point-to-point networking applications requiring data integrity, it is still rather tedious for a programmer to implement in high-level applications. The programmer still has to know how to use the OS APIs to create and manage TCP sockets and since only bytes can be sent over the socket, the programmer needs to manually serialize and deserialize internal program structures and variables to transmit them. Besides this, raw TCP is only really suited for continuous data streams, while many applications require event-based bidirectional communication where messages (blocks of data with clear boundaries) are sent back and fourth.

Providing solutions for these annoyances/problems is the job of communication frameworks or library. A communication framework does not only implement a higher level protocol to support the additional functionality, but also provides programming language functions to integrate the functionality as seamlessly as possible with language features.

In summary, they may provide the following services to the user:

- Abstraction of low-level socket APIs, exposing the most important functionality and automatically handle the rest in the background (according to common use-cases)
- Automatic management of connection state, handling disconnects and reconnects
- Provide a way to exchange messages, which are blocks of data of a known (but still possibly dynamic) length, guaranteeing arrival in same grouping and order as sent
- Provide facilities to serialize and deserialize language-native data structures to easily send them over the network (with the least code possible)
- Provide data validation for sent and received messages according to programmer defined schemas (in language native format as far as possible), so the user code can trust received data to be in valid format.
- Provide additional, commonly used communication schemes that further simplify common use cases of network communication for the developer such as Remote Procedure Calls (RPCs)

## II. State of the Art

- TCP & UDP (mostly TCP) for low-level, very performant systems
- WebSockets: Implements an additional protocol on top of TCP re-introducing the concept of messages. For Web applications already using HTTP(s)
- SocketIO: Builds on top of WebSocket, allows sending javascript events over the network, adding event listeners/emitters and allow sending arbitrary data.

## III. Problems with current offer

This section builds upon the last mentioned protocol SocketIO.

No reassurance of data validity. before using incoming data, one ideally has to check that the structure and type of the received data matches what is expected and handle the error if that isn't the case. This is tedious to do manually for every single event. Additionally since SocketIO allows sending arbitrary data, it is easy to accidentally send data in an incorrect structure, such as accidentally making a typo in an attribute name.

## IV. Solution and Design Overview

## V. Implementation

## VI. Results and Applications

## VII. Limitations

## VIII. Conclusion

## List of Figures

## References

[1] Proof for NordPass study, URL:https://tech.co/password-managers/how-many-passwords-average-person