CSC3002 – Networks Assignment 1 – 2025
Socket programming project

Department of Computer Science
University of Cape Town, South Africa

February 24, 2025

This assignment is on networked applications, where you will learn the basics of protocol design and socket programming with TCP and UDP connections: how to create a socket, bind it to a specific address and port, and send and receive messages/files. In groups of three students, you will design and implement a simplified torrent-like file-sharing system that enables peer-to-peer (P2P) file distribution. You will use TCP for reliable data transfer and UDP for tracker communication and peer discovery. This document describes the context/requirements (Section 1), what to submit (Section 3), and some basic information about socket programming (Section A).

1. Application Description

Peer-to-peer (P2P) file-sharing systems enable distributed content sharing without relying on centralized servers. BitTorrent, one of the most well-known P2P protocols, allows users to download and upload file chunks efficiently from multiple peers.

In this assignment, you will design and implement a mini-BitTorrent-like system that enables decentralized file sharing. This system will consist of three main components:

1. A **tracker**, which coordinates peer discovery.
2. A **seeder**, which provides file chunks for download.
3. A **leecher**, which downloads chunks from multiple peers and assembles them into a complete file.

Your implementation will use UDP for tracker communication (lightweight and fast) and TCP for reliable file transfer.

The tracker (a UDP server) serves as a lightweight directory responsible for maintaining a list of available peers participating in the file-sharing system. It relies on UDP sockets to facilitate communication with both seeders and leechers. When a leecher

requests information about available sources for a particular file, the tracker responds with a list of active seeders. To ensure accuracy, the tracker periodically removes inactive peers from its records, preventing stale or outdated information from being distributed.

The seeder (a TCP server) acts as a file provider, hosting a file and dividing it into fixed-size chunks, such as 512 KB per chunk. The seeder registers with the tracker via UDP. When a leecher requests a file, the seeder transmits the necessary chunks using TCP, ensuring reliable and ordered data delivery. To remain discoverable within the network, the seeder periodically notifies the tracker of its presence via UDP, allowing leechers to locate it when searching for the desired file.

The leecher (a TCP client) functions as a file downloader, initiating contact with the tracker through UDP to obtain a list of peers currently hosting the file. Once it has identified available seeders, the leecher establishes TCP connections with multiple sources, enabling parallel downloads of different chunks to maximize efficiency. As it receives and stores file chunks, it reconstructs the complete file on its system. After successfully downloading the entire file, the leecher may optionally transition into a seeder, making the file available for others to download and contributing to the peer-to-peer network.

Additional features to include:

- Parallel Downloads: Implement chunked downloads from multiple seeders simultaneously.
- File Integrity Verification: Use hashing (e.g., SHA-256) to validate chunks. (**optional for extra marks**)
- Re-Seeding: Leechers automatically transition to seeders after downloading.
- Graphical UI: Create a simple interface showing download progress. (**optional for extra marks**)

2. Protocol Design and Specification

Protocol specification involves defining the types and structure of messages. Three types of messages can be defined; commands, data transfer, and control. Command messages define the different stages of communication between parties, such as the initiation or termination of communication. Data transfer messages are used to carry the data that is exchanged between parties, and such data could be fragmented into several messages. Control messages manage the dialogue between parties, including such aspects as message acknowledgment, and retransmission requests.

The protocol message structure constitutes at least the header and body. The header, whose structure must be known to both sides of a communication link, may contain fields that describe the actual data in the message. Some of the fields/information contained in the header might be the message type, the command, and the recipient. The header generally has a fixed size and contains clues that should help the receiver understand the rest of the message.

The last aspect of the protocol design will be the communication rules that specify the sequence of messages at every stage of communication. For example, a peer can be in an 'available', 'connected', or 'away' state. How do peers respond to messages while in these different states, and how do they transition between states. You need to specify messages and reactions for different states. You will need to represent such rules with sequence diagrams.

3. What you need to submit

You will be required to submit the following:

1. You are to submit the source code (Python, Java, or C++), with proper inline documentation (comments), and a README file explaining how to set up and run the tracker, seeder, and leecher.

2. A report (max 10 pages) on the design and functionality of your application.

3. In addition, the report needs to include:
   a) A list of features with a brief explanation for their inclusion
   b) A protocol specification, detailing the message formats and structure. You are required to include sequence diagram(s).
   c) Screenshots of the application revealing its features.

4. Oral presentation to be scheduled with the TAs and Tutors (oral to be done on day after submission deadline)

A. Multi-threaded Client/Server Applications—Sockets Programming

A.1. What is a socket?

A socket is the one end-point of a two-way communication link between two programs running over the network. Running over the network means that the programs run on different computers, usually referred as the local and the remote computers. However one can run the two programs on the same computer. Such communicating programs constitutes a client/server application. The server implements a dedicated logic, called **service**. The clients connect to the server to get served, for example, to obtain some data or to ask for the computation of some data. Different client/server applications implement different kind of services.

To distinguish different services, a numbering convention was proposed. This convention uses integer numbers, called port numbers, to denote the services. A server implementing a service assigns a specific port number to the entry point of the service. There are no specific physical entry points for the services in a computer. The port numbers for services are stored in configuration files and are used by the computer software to create network connections.

A socked is a complex data structure that contains an internet address and a port number. A socket, however, is referenced by its descriptor, like a file which is referenced by a file descriptor. That is why, the sockets are accessed via an application programming interface (API) similar to the file input/output API. This makes the programming of network applications very simple. The two-way communication link between the two programs running on different computers is done by reading from and writing to the sockets created on these computers. The data read from a socked is the data wrote into the other socket of the link. And reciprocally, the the data wrote into a socket in the data read from the other socket of the link. These two sockets are created and linked during the connection creation phase. The link between two sockets is like a pipe that is implemented using a stack of protocols. This linking of the sockets involves that internally a socket has a much more complex data structure, or more precisely, a collaboration of data structures. Thus, a socket data structure is more than just an internet address and a port number. You have to imagine a socket as a data structure that contains at least the internet address and the port number on the local computer, and the internet address and the port number on the remote computer.

A.2. How is a network connection created?

A network connection is initiated by a client program when it creates a socket for the communication with the server. To create the socket, the client calls the Socket constructor and passes the server address and the the specific server port number to it. At this stage the server must be started on the machine having the specified address and listening for connections on its specific port number.

The server uses a specific port dedicated only to listening for connection requests from

clients. It can not use this specific port for data communication with the clients because the server must be able to accept the client connection at any instant. So, its specific port is dedicated only to listening for new connection requests. The server side socket associated with specific port is called server socket. When a connection request arrives on this socket from the client side, the client and the server establish a connection. This connection is established as follows:

1. When the server receives a connection request on its specific server port, it creates a new socket for it and binds a port number to it.

2. It sends the new port number to the client to inform it that the connection is established.

3. The server goes on now by listening on two ports:
   - it waits for new incoming connection requests on its specific port, and
   - it reads and writes messages on established connection (on new port) with the accepted client.

The server communicates with the client by reading from and writing to the new port. If other connection requests arrive, the server accepts them in the similar way creating a new port for each new connection. Thus, at any instant, the server must be able to communicate simultaneously with many clients and to wait on the same time for incoming requests on its specific server port. The communication with each client is done via the sockets created for each communication.

TCP is a connection-oriented protocol. In order to communicate over the TCP protocol, a connection must first be established between two sockets. While one of the sockets listens for a connection request (server), the other asks for a connection (client). Once the two sockets are connected, they can be used to transmit and/or to receive data. When we say "two sockets are connected" we mean the fact that the server accepted a connection. As it was explained above the server creates a new local socket for the new connection. The process of the new local socket creation, however, is transparent for the client.

The datagram communication protocol, known as UDP (user datagram protocol), is a connectionless protocol. No connection is established before sending the data. The data are sent in a packet called datagram. The datagram is sent like a request for establishing a connection. However, the datagram contains not only the addresses, it contains the user data also. Once it arrives to the destination the user data are read by the remote application and no connection is established. This protocol requires that each time a datagram is sent, the local socket and the remote socket addresses must also be sent in the datagram. These addresses are sent in each datagram.