**Coursework assignment:** Communication via computing technology

In our modern-day to day life, we actively use different online messengers and voice chats. To communicate with friends, I frequently use the software called “Discord” for instant messaging and voice chats. This program has a computer and mobile version: the desktop version is stored on the hard drive (mass storage device or hard disk drive [HDD]) of my PC and the mobile one is on the flash drive of my phone. These two types use different technology to store data – a magnetic disk for the HDD and electronic circuits for the flash drive – the main purpose is the same, to offer long-time storage. The HDD is slower because it requires a physical spinning of the disk but holds more data in comparison to the flash drive. When I initialize the program start (either by double click on PC or touch on the screen of the phone) operating system then load the code from the mass storage to the volatile memory (random access memory [RAM]). I use Windows OS on PC and Android OS on the phone, but both systems work in the same way. They have a “User layer” in which I work, that recognize my “clicks/touches” as the command to start the program that is then transferred to the “Kernel layer” which reserves the memory addresses for the program and executes binary instructions of the software to add it to a queue. Both OSs do this differently because they use different kernels that are suited for their architecture. The control unit of the CPU then sends the address of the first instruction of the program to the memory (memory and CPU are communicating through the address bus) and in response, the memory returns the next instruction to the CPU to execute, and the address of the next instruction and the process repeats itself. Program code itself has instructions for the OS that it uses to create the visual environment that we work with called “user interface”. This environment is different based on the device (PC, tablet, smartphone) and the OS that we use (Windows PC and Mac PC for example will look slightly different), but the program itself looks almost identical, that user will not experience any trouble shifting from one device or OS to another.

I wasn’t able to find the exact explanation about the inner working of this program, so this is my educated guess. While I manipulate the program through the interface I can send and receive text messages, images, and other media files. “Discord” doesn’t create its files but can be used to transfer files of any format. But this program is more suited for communication, so its main features are online messages and voice chats. There are two main ways to handle communication online: peer-to-peer (P2P) and client-server transmission. P2P connection implies a direct link between two users, without a mediator, while client-server implies that individuals work through centralized administration (servers). “Discord” is a client-server type messenger, which implies that all text, file, and voice exchanges are going through the server. When “Discord” loads it creates a request to login using a user Id and a password. This request is just another instruction from the CPU to the peripheral device – the network adapter. On a hardware level: the CPU transfers data to this peripheral device using network adapter databus. On a software level: OS communicates with the network adapter through drivers, the pieces of software that administer and manages data exchange between devices. There are a lot of different peripherals and drivers, but they all have the same purpose and for “average” users they indistinguishable. The “Discord” then creates a request for the network adapter to form a packet for a connection to the server (“handshake”) containing information about the source (address of my computer or phone), about the destination (address of the server) and login data. This messenger uses one of the commonly used protocols (TCP/IP) to establish the connection between a client and a server.

My PC is connected to the internet through the optic fiber cable, while my phone uses a Wi-Fi wireless connection. The wired connection is more stable compared to Wi-Fi because its data transferred directly through the cable, but the phone uses the built-in antenna to send information using radio waves. That’s why wireless connection quality is worse: there are a lot of different factors that create interference to the signal and cause the loss of data in exchange. When I chat using messages, the difference is mostly negligible, because this loses almost undetectable and looks like a small delay compared to the PC chat. But if I use voice chat from the phone, these packet losses will create problems – audio/video will break up a lot, and because the process is dynamic, the difference in quality will be more apparent. But despite the way I connect, the packets will then proceed to the next type of equipment – router. This hardware will choose the route that my packets will go to between the source and destination addresses.

The packet follows an established route – moving from network to network, from router to router, while each router creates a way to the next. Some routers use static tables that show already established pathways – this called static routing, while others update, the pathway tables regularly to find the shortest and fastest route, this is dynamic routing. While dynamic routing is obviously better, it requires a lot more computational power and is mostly used by big networks. Returning to the initial packet: the “handshake” packet is going through the “reverse” process. Received by the servers’ network adapter, it is then transferred to the CPU and from the CPU through the two-layers of OS to the “Discord” Server version software that checks my login/password data from the packet and checks it with the information from the database. If my credentials are right, the server then creates the response with the acknowledgment packet (token) that confirms that the handshake was received and verified. At this point (after sending the first packet) my program sends the message to the network adapter, that it is awaiting a response from the server. The server's response packet then moves all the way back to my device (the principal is the same). When my network adapter receives this response, it informs my program and my “Discord” then responds with the last acknowledgment, that server does not reply to. This is the connection and authorization process also called “Log-in” that results in an established network tunnel. After that, my program loads a user interface that allows me to browse different “Discord servers” (“Discord server” is a small cluster of the database that contains all messages, users, and voice channels), send/receive messages, and use voice/video chat.

A big part of “Discord” functionality comes from databases – big, structured collections of data that contain all messages from different open servers and private chats between users. Two main parts of database are data model (a set of rules that describes how data is stored, organized, accessed) and the data itself. There exist a lot of different database models, but most of them use tabular system (elements are placed in rows and columns represent the attributes of this elements) and structured query language (SQL, the set of commands that used to access required data stored inside) and called relational database. But “Discord” uses Casandra, a non-SQL (it uses its own query language) database that utilize another way of storing data. It is similar in some way but uses more sophisticated way of storing data. This choice allows for fast and more flexible way to access data. I can guess that if they used SQL database instead, they had to use more hardware to provide the same query response time.

After my client version of the “Discord” is connected to the main server, my program then checks what “Discord servers” and chats I have and create a request to the database to get new messages and load chat history. This is done like the login process, but instead of transferring login information, it checks for text messages and retrieve them to the user and then show them on the screen. When we send messages, the process is similar. When I enter text into the chat box in the “Discord” window the program will send the request to the CPU, to connect to the peripheral device (keyboard on the PC and touchscreen on the smartphone) and retrieve the sequence of the inputs that I pressed. Keyboard uses mechanical switches to register what button was pressed, while the touchscreen “senses” the coordinates of the press. Touches then translated to the electrical signals and drivers then translate these signals to the letters, numbers, and symbols and send this information to the CPU through the secondary bus (SCSI, for Small Computer System Interface). When CPU translates the whole message, encodes it (for security reasons, that your information will be safe in case of traffic loss), breaks it into packages, and sends it to the server while returning the message to the software and monitor to show to the user. When the server gets this message it is then decoded, translated, and appended to the database. Next, the server sends this package to all users to whom it was addressed, and their “Discord” sends the instruction to the CPU to access the Video output device (Monitor or Mobile screen) and Audio output device (speaker or headphones) through secondary bus and create a visual and audio representation of the received message. Voice and video chats work similarly. But instead of the input from the keyboard CPU reads from the users’ microphone and camera. The CPU will work in the same way as it did with the keyboard and use the same bus. Encoded information then will be sent to the server, from there to the recipient computer. There, decoded information will be translated to the audio and video stream on the user’s computer. The difference between mobile and PC versions is only in the hardware that is used.

I also can send different files to other users, but most of the time people use this feature to send each other images and/or music. Because “Discord” is the server-based messenger it transfers all files from the sender to the server and then from the server to the recipient. Because of this, this feature put a strain on the server storage capabilities and developers implemented a size limit. This restriction force users to compressed files – that trade some of the quality of the initial media for lighter file size. For example, an image saved in JPEG format will change the color of some pixels to that of their neighbor and save space by reduction of the image's color pallet. A popular sound format MP3 and video format MP4 use the same concept and allow for smaller file sizes, compared to non-lossy compression.

**References:**

1. Glenn J. Computer Science: An Overview, 11 edn., Massachusetts: Addison-Wesley, 2012.
2. Kroenke, David M. and David J. Auer. Database Concepts. 3rd ed. New York: Prentice, 2007.
3. Wells D. (2012) Let’s Reverse Engineer Discord, Available at: https://medium.com/tenable-techblog/lets-reverse-engineer-discord-1976773f4626 (Accessed: 28.12.2020).
4. Vishnevskiy S. (2017) How Discord Stores Billions of Messages, Available at: https://blog.discord.com/how-discord-stores-billions-of-messages-7fa6ec7ee4c7 (Accessed: 28.12.2020).
5. CLOUDFLARE.COM (2020) What is routing?, Available at: https://www.cloudflare.com/learning/network-layer/what-is-routing/ (Accessed: 28.12.2020).
6. Fox P. (2020) Data compression, Available at: https://www.khanacademy.org/computing/computers-and-internet/xcae6f4a7ff015e7d:digital-information/xcae6f4a7ff015e7d:data-compression/a/file-compression-introduction (Accessed: 28.12.2020).
7. Coustan Y.D., Strickland J., Perritano J. (2020) How Smartphones Work, Available at: https://electronics.howstuffworks.com/smartphone.htm (Accessed: 28.12.2020).

**Figures:**

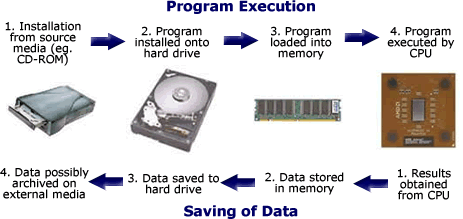


Figure 1 – Machine cycle [Image is taken from the website: <http://computerscience.chemeketa.edu/cs160Reader/ComputerArchitecture/MachineCycle.html>]

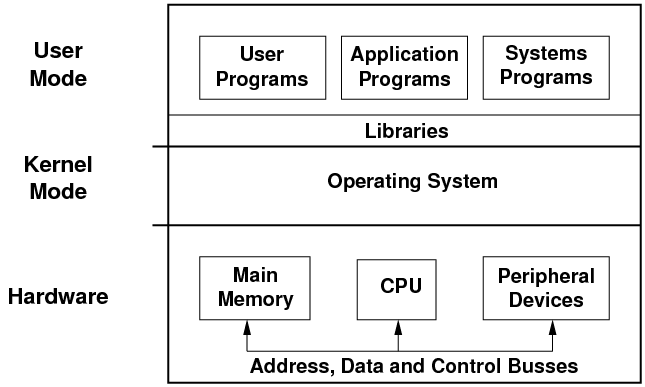


Figure 2 – Computer system levels: hardware, kernel and user [Image is taken from the website: <https://minnie.tuhs.org/CompArch/Lectures/week07.html>]

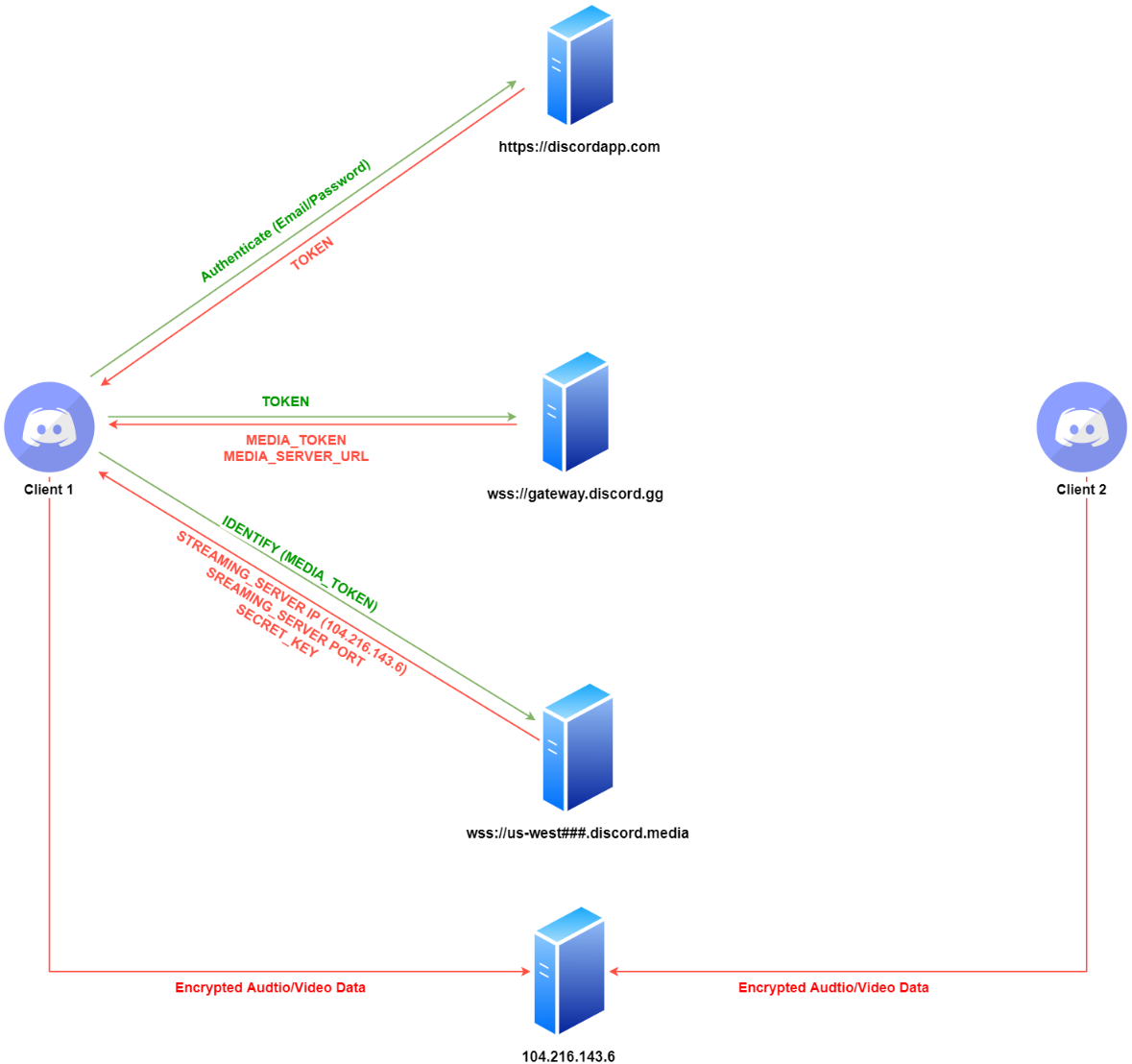


Figure 3 – “Discord” client-server communications [Image is taken from the website: (3)]