

UNCLASSIFIED

File Delivery with UDP and an Unreliable Network

Katelyn Atkinson

Team Monarch

Timothy Reidy

Ashley Yu

Alicia Unterreiner

Spencer Scamman

Ian Paap-Gray

July 25, 2022

UNCLASSIFIED

Executive Summary

The Virtual Institutes for Cyber and Electromagnetic Spectrum Research and Employ (VICEROY) assigns Team Monarch, a team of six interns, a challenge problem related to communication across a network. The team must deliver a file with the user datagram protocol (UDP) across an unreliable network. From the client, we must send a file given to us by Mr. Allen, and deliver the file to the server. In our solution, we must utilize the IP addresses associated with 10.65.97.0/24 on the Ubuntu VICEROY virtual machine (VM). We must also use ports 2001 and 2002. The file on the server must match the file on the client to demonstrate delivery uncompromised by packet loss in the network.

We assume access to the team VM and virtual private network (VPN) as set up by Mr. Allen and the graduate assistants. We also assume delivery of our file occurs without a constraint on time. We assume the size of the files transferred across the network fall within the memory capacity of the system's hardware. Ports 2001 and 2002 we assume to exist as open and available for use.

We base our solution on a TCP model for communication and finalize our solution with UDP. We access the team VM in the *ubuntu@viceroy-01.vcry* environment and start a connection on the VICEROY VPN. We then secure the "example.data" file from Mr. Allen's lecture folder. To format the packets we send over the network, we use an index in front of the packet's data. Inside the VM, we use Visual Studio Code to create python scripts for a client and server, each hosted across two computers. Both utilize Python methods that allow their sockets to send and receive. We use acknowledgements from client to server to overcome the issue of packet loss. We finish with the creation of a file which contains packets in the order and quality of the original.

1. Problem Statement

The Virtual Institutes for Cyber and Electromagnetic Spectrum Research and Employ (VICEROY) assign six interns to Team Monarch. As guest-lecturer, Mr. Samuel Allen provides a challenge problem to our team. The challenge problem requires the team to deliver a file via a transport protocol known as the user datagram protocol (UDP) over the Internet. We must overcome the limitations of UDP, which include loss of packets and failure of a file's delivery, to produce a file transferred without errors.

The network used to deliver the file hosts a foundation prone to errors, such as a packet dropped and not delivered to the server or client. Our team must provide two deliverables to complete the challenge problem. They consist of this document which specifies the steps to our solution and a demonstration to show delivery of the file. The file follows the format or may consist of the the "example.data" file Mr. Allen provides in his lecture. Delivery of the file must occur in packets as designed by UDP. Communication to the server must end with the packets sorted in order and with the quality they contained before their transfer across the network.

The team must build and implement a script in Python for one client and one server on the network. The client must exist on a device separate from the device which hosts the server. The challenge problem's instructions mandate us to utilize 10.65.97.xxx as the server's IP. The secure shell (SSH) Mr. Allen deploys as *ubuntu@viceroy-01.vcry* limits our addresses to those within the 10.65.97.0/24 interface. The challenge problem also specifies communication must exist on the ports 2001 and 2002. Resources we use must include the VICEROY Ubuntu virtual machine, as well as the VPN used by VICEROY. The file transferred we must show on the VM.

2. Background

The previous chapter outlines the problem statement and the need for a solution which uses UDP to deliver a file without errors. In this chapter, we detail the background and information relevant to comprehension and application of the solution.

2.1 Networks

Networks consist of two or more computers connected together to share resources [1]. In this section we cover terminology used to describe networks and the protocols used to transport information.

2.1.1 Terminology

To establish a network, we need a client, server, IP addresses, transport protocols, packets, ports, and sockets. A client interfaces with the server to receive and send transmissions [1]. Managed by administrators, servers accept and respond to requests from clients [1]. Both server and client match to Internet protocol (IP) addresses [1]. IP addresses exist as a string of numbers meant to identify a computer that communicates across a network [1]. The IP format `x.xx.xx.x/#` contains a “#” to represent the bits that identify the network and the rest of the bits left to represent the number of hosts on the network [1]. Messages sent across the network come in the form of packets, which exist as pieces segmented from the original [1]. Transport protocols describe the transmission of packets between clients and servers, such as UDP [2]. Part of the operating system, a port exists at the start or end of a network’s connection and helps sort traffic over a network [2]. A socket exists in Python programs as one endpoint in the communication between programs run on a network [3].

2.1.2 TCP vs UDP

The transmission control protocol (TCP) ensures packets arrive at their destinations and maintain a connection without the loss of packets [2]. The user datagram protocol (UDP) checks distortion of packets, but allows packets to drop without retransmission [2]. While TCP retransmits packets, the protocol maintains slower transmission than UDP [2]. TCP gains popularity for the ability to secure networks, while UDP loses popularity for the inability to maintain reliability of delivery [2].

2.2 VPN

A virtual private network (VPN) extends a private network over a public network [2]. Encryption of data and authorization of users allows the network to send and receive data across the Internet with security and reliability [2].

2.3 .data File Extension and Byte Strings

The .data file extension contains machine-readable data [4]. These files store memory dumps, text-based lists, binary data containers or data for payloads in a program [4]. Similar to strings, byte strings exist as a sequence of bytes instead of characters [5]. Often, .data files contain byte strings or contain data able to transform into a byte string format [5].

2.4 Applications

Networks allow communication across the world, as well as the ability to share resources [6]. The market for management and security of network services around the nation displays a growth of 6% in the next 8 years [6].

The number of cyberattacks on corporate networks increased 50% in 2021 [7]. This demonstrates the need to design networks and protocols which allow for confidentiality, integrity, and availability of data [7]. In 2020, Russian hackers attacked email networks in the Department of Justice and Homeland of Security after their breach of the SolarWinds company [8]. Our challenge problem utilizes UDP and attempts to build communication over an unreliable network with the delivery of a file uncorrupted by the transmission. Our solution describes how organizations and individuals may harden their unreliable networks to ensure the delivery of messages communicated across their networks.

3. Assumptions

The previous chapter details the background to our challenge problem. In this chapter, we list the assumptions we make to establish our solution.

For the completion of our solution, we assume access to the team VM and VPN, as Mr. Allen and the graduate assistants (GAs) determine which users may access these tools. Due to the randomness of chance associated with packet delivery, we assume delivery occurs without a time constraint. We also assume the size of the files transferred across the network fall within the memory capabilities of the client's and server's systems. We also make an assumption about the availability and openness of ports 2001 and 2002 used in our solution.

4. Tools and Techniques

The previous chapter describes the assumptions made in our challenge problem. In this chapter we detail the tools and techniques behind our solution.

4.1 VICEROY VPN

Graduate assistants (GAs) set up the VICEROY VPNs with the .conf files “GI-viceroy-xx.conf” and “viceroy-xx.conf”. The “x’s” map to each intern. The VPN connects to the VICEROY server.

4.2 Visual Studio Code

Released in 2016, Visual Studio (VS) Code exists as a code editor used to build and debug programs written in a variety of languages [9]. VS Code thrives as a platform for the development of code in macOS, Linux, and Windows [9].

4.3 Python 3.0

The 3rd version of the Python language, Python 3.0 works in tandem with Python 2.0 and maintains the language’s ability to write scripts at high-level and with object-orientation [10].

The command *python3* in a terminal runs a script for this version of Python [10].

4.4 Python Functionalities

Here we discuss the libraries, methods, and data formats we utilize in Python and describe their functions.

4.4.1 *to_bytes()* and *from_bytes()*

The *to_bytes()* function takes an integer and returns an array of bytes [11]. The *from_bytes()* function takes an array of bytes and returns an integer [11].

4.4.2 *set* and *list*

The data type of *set* in Python exists to store collections of data [12]. The data inside sets occur unordered, unchangeable, and unindexed [12]. The *list* data structure contains the functions *add()*, which adds elements to a list, and *sort()*, which sorts the elements based on parameters specified by the programmer [13]. A *set* passed into a *list* with the *add()* function may become sorted with the *sort()* function [13].

4.4.3 *socket*

The *socket* library contains the methods *bind()*, *close()*, *sendto()*, *recvfrom()*, *settimeout()*, and contains exceptions built in to provide error messages [14]. The library creates socket objects and constants, such as *socket.AF_INET* and *socket.SOCK_DGRAM* for UDP implementation [14]. The *bind()* function binds a socket object to an address and port, while the *close()* function closes a socket [14]. The *sendto()* function sends data to a socket at an address specified, while *recvfrom()* receives data from a socket on a port and address specified [14]. The *settimeout()* function disables the socket's ability to receive packets if the time passed since transmission exceeds the limit inside the parentheses [14].

4.4.4 *open()*, *read()*, and *write()*

These methods built into Python handle files imported to a script [15]. The *open()* method opens the file, the *read()* method reads a file line-by-line, and the *write()* method creates a file [15].

4.5 Ubuntu VICEROY Virtual Machine

Virtual Box, a platform used to support a variety of operating systems (OS) in a virtual environment, hosts the Ubuntu VICEROY virtual machine (VM) [16]. A VM exists as software which mimics the abilities of a physical computer [16]. The VICEROY VM provides the Ubuntu OS in order to run Linux.

4.6 Vim

Developed by Sven Guckes as an editor for text files, Vim allows for the creation and ability to change files from the terminal [17]. The command *vi* in the terminal opens a file and permits a user to read, edit, or delete the file [17].

5. Problem Solution

The previous chapter catalogs the tools implemented. In this chapter, we explain how we utilize the Ubuntu VICEROY VM to create a client and server for transportation of packets over an unreliable network.

5.1 Packet Set Up

We format our packets to send over the network with variables we call “index” and “temp”.

Figure 1 illustrates index as the integer which identifies the packet and tells the order of the packet in the file. The variable “temp” represents the data of the packet. We format our packets to contain 1024 bytes.

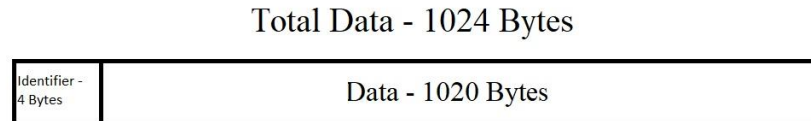


Figure 1. Packet Format

5.2 VPN and VM Set up

Inside the Ubuntu VICEROY VM, we use the terminal to access the “wireguard” folder in our VICEROY directory and edit the “GI-viceroy-27.conf” and “viceroy-27.conf” files to configure our VPN. We enter the IP address 10.65.97.0/24 under the “AllowedIPs” section. We enter the command `wg-quick up <VPN name>` to access the VPN. To connect to the team VM, we enter the command `ssh ubuntu@viceroy-01.vcry`, along with the password “viceroy”.

5.3 Download “example.data”

From Mr. Allen’s Gitlab repository, we download the “example.data” file, as shown in Figure 2.

This file contains bytes with characters not recognized as valid so we open with “write in binary” mode. We download the 50kb file to our local machines.

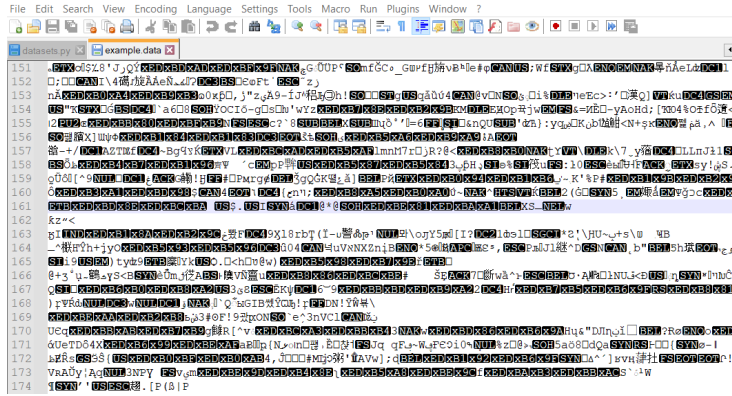


Figure 2. “example.data” file

5.4 Client Build

To create our “client.py” script, we read in the “example.data”. With *open()* and *read()*, we create the variable “teststuff”. This variable holds the contents of “example.data” in one byte string.

We then create our packets and sockets to send to the server, along with a timeout feature.

5.4.1 Formation of Packets

A while loop splits “teststuff” into byte strings of size 1020 bytes. Another loop for iteration tacks a 4 byte index onto the front with the *to_bytes()* function and appends the 1020 bytes to the end to create a 1024 byte packet.

5.4.2 Socket Creation and Packet Transmission

We use the server’s IP address to create and bind two sockets, one meant to receive on port 2001 and another meant to send packets on port 2002. In a while loop, we send the quantity of packets to the server until we get an acknowledgement of “ACK”. Here, we send 50 packets, one with fewer bytes than the others. The second while loop transmits the packets until the server sends back an “ACK”. This “ACK” closes the sockets and ports for termination of the program.

5.4.3 Timeouts and Exceptions

We code the client to pause with `settimeout()` when the line holds silence for 0.1 seconds. This prevents a stall in the program. In the case of packets lost, we display “timed out” on the terminal. We employ exceptions whenever an error occurs in the configuration of the network or transmission of packets. An exception gives us the line of code the error occurred on and ensures the sockets close to enable use of the program in the future. Figure 3 illustrates timeout and acknowledgement in green and the exception code in blue.

```

46     response = b""
47     packetsNum = len(data).to_bytes(length=4,byteorder='big')
48     while response.decode().rstrip() != "ACK":
49         socksend.sendto(packetsNum, ("10.65.97.110",2002))
50         try:
51             response, addr = sockrecv.recvfrom(3)
52             print(response)
53         except Exception as e:
54             print(e)
55             continue
56     response = b""
57     while response.decode().rstrip() != "ACK":
58         for x in range(len(data)):
59             socksend.sendto(data[x], ("10.65.97.110",2002))
60         try:
61             response, addr = sockrecv.recvfrom(1024)
62             print(response)
63         except:
64             continue
65     sockrecv.close()
66     socksend.close()
67 except Exception as e:
68     exc_type, exc_obj, exc_tb = sys.exc_info()
69     print(exc_tb.tb_lineno)
70     print(e)
71     sockrecv.close()
72     socksend.close()
73

```

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER
ace-intern@ceintern-VirtualBox:~/JDPServerCP$ python3 ChallengeTCPClient.py
Connected!
timed out
timed out
timed out
timed out
timed out
b'ACK'
b'ACK'
ace-intern@ceintern-VirtualBox:~/JDPServerCP$ python3 ChallengeTCPClient.py []

```

Figure 3. Timeout and Acknowledgement in Terminal

5.5 Server Build

In VS Code inside the VM, we create a python file named “server.py”. We create two sockets, one to receive packets on port 2002 and one to send packets on port 2001. We use `bind()` to bind the socket which receives packets to the IP address of the machine which hosts the server. The socket we use to send data we allow to bind to the client without `bind()`. In order to edit our code from the terminal, we use Vim with the command `vim <file name>`.

5.5.1 *sortsets()*

We create a function called *sortsets()* with the *sort* library, as shown in Figure 4. This function takes our packet's index and data as an element and sorts them into a list. The list maintains the packets in the order found in "example.data".

```
def sortsets(input):
    data = list(input) #converts set to list
    data.sortsets(reverse = False) #sorts set based on first element of tuple in list
    return data
```

Figure 4. *sortsets()* function

5.5.2 *handle_client()*

We write a function called *handle_client()* with two while loops, which we use for iteration. The function also includes the ability to set a timeout and write data to a file.

5.5.2.1 Number of Packets

The first while loop receives an integer from the client that specifies the number of packets. We use *from_bytes()* to convert this integer into 4 bytes. To compensate for the loss of packets, we continue to receive transmissions with *recvfrom()* until we receive the integer. Once received, we use *sendto()* to send "ACK" back to the client.

5.5.2.2 Packets Added to a List

The second while loop receives packets and places the first 4 bytes made from *from_bytes()* into the "index" variable. The loop sets the packet's data to the "temp" variable and places both into a set with the *add()* function. The loop continues until the number of packets received matches the integer for length we received in the first while loop. We send an "ACK" back to the client.

5.5.2.3 Set a Timeout

We employ the `settimeout()` function with a limit of 0.01 seconds to make the server pause the receive function until the client sends another packet. As long as the server takes less than 20 timeouts, we receive and send acknowledgements of “ACK” back to the client.

5.5.2.4 “finaldata.data”

With the packets received and ordered in our list, we take the packets without their index and store them as “finaldata.data” with the `open()` and `write()` commands. To compensate for the characters not valid in the “example.data” file, we use “wb” for “write in binary”, as shown at the bottom of Figure 5 in red.

```

while True:
    index_encoded = temp[:4]
    temp = temp[4:]
    index = int.from_bytes(index_encoded, byteorder="big")
    data = (index, temp)
    dataset.add(data)
    if len(dataset) != num_packets:
        temp, addr = con_rec.recvfrom(1024)
    else:
        break
print("we made it")

con_rec.sendto("ACK".encode(), (addr[0], 2001))
quiet_times = 0
con_rec.settimeout(.01)
while quiet_times < 20:
    try:
        con_rec.recvfrom(1024)
        con_send.sendto("ACK".encode(), (addr[0], 2001))
    except:
        quiet_times += 1
        continue
#print(dataset)
finaldata = sortsets[dataset]
print(finaldata[1])

f = open("finaldata.data", "wb")
for x in finaldata:
    f.write(x[1])

handle_client(sockrec, socksend)

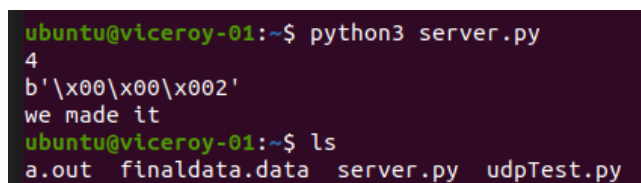
```

Figure 5. “server.py” and “finaldata.data” creation

5.6 Tests and Finalization

To ensure our code compiled and allowed us to deliver “example.data” between client and server, we tested the python scripts over a reliable network. The basis for this test used TCP as a model to mimic reliable delivery of packets. We used the IP addresses associated with the *GI-viceroy-27* and *viceroy-27* VPN configurations.

We then moved our programs to the virtual network on *ubuntu@viceroy-01.vcry* and sent packets through the 10.65.97.0/24 interface, which Mr. Allen designed as unreliable [18]. Figure 6 shows execution of “server.py” in the team VM’s terminal and the directory’s contents. The *ls* command displays “finaldata.data” as our deliverable.



```
ubuntu@viceroy-01:~$ python3 server.py
4
b'\x00\x00\x00\x002'
we made it
ubuntu@viceroy-01:~$ ls
a.out  finaldata.data  server.py  udpTest.py
```

Figure 6. Delivery of “finaldata.data”

6. Risk Assessment

The previous chapter lays out the steps taken to deliver a file over a network which contains the risk for packet loss. In this chapter, we review the assumptions and limitations of the problem’s solution.

For our solution, we assume access to the team VM and VPN. Access to these tools limits our solution to a network defined by the standards built into the Ubuntu VICEROY VM, as well as the configuration settings predetermined by Mr. Allen.

We assume delivery of our file occurs without time constraints. We must revisit how to handle packets and their retransmission if the network requires speed.

We also assume the system allows transfer of our file size. Changes to the size of the file transported must exist in systems whose memory capabilities interfere with delivery.

UNCLASSIFIED

We make an assumption about the availability and openness of ports 2001 and 2002. Limitations arise if these ports become unavailable, thus we must sift through ports to find alternatives.

An oversight includes the ability for an attacker to transmit packets across our connection if they obtain knowledge of our ports, transmission times, and IP addresses. Replication of our solution manifests a chance to also include security standards and the application of user authentication, firewalls, and privileges.

References

- [1] “Computer Network Terminology: 7 Essential Terms,” *Ohio University*, Jun. 05, 2018. <https://onlinemasters.ohio.edu/blog/computer-network-terminology/> [Accessed: 23-Jun-2022].
- [2] “Transport Protocols,” *www.cl.cam.ac.uk*. <https://www.cl.cam.ac.uk/~jac22/books/www/book/node21.html> [Accessed: 23-Jun-2022].
- [3] “Socket in Computer Network,” *GeeksforGeeks*, May 09, 2020. <https://www.geeksforgeeks.org/socket-in-computer-network/> [Accessed: 23-Jun-2022].
- [4] “DATA File Extension - What is it and how to open DATA format - Review,” *www.filetypeadvisor.com*. <https://www.filetypeadvisor.com/extension/data> [Accessed: 23-Jun-2022].
- [5] “3.5 Bytes and Byte Strings,” *cs.brown.edu*. <https://cs.brown.edu/courses/cs173/2008/Manual/guide/bytestrings.html> [Accessed: 23-Jun-2022].
- [6] T. B. Insights, “Managed Network Services Market Size Worth \$101.54 Billion by 2030: The Brainy Insights,” *www.prnewswire.com*. <https://www.prnewswire.com/news-releases/managed-network-services-market-size-worth-101-54-billion-by-2030-the-brainy-insights-301487277.html> [Accessed: 23-Jun-2022].
- [7] “Cyber attacks on corporate networks increased 50% in 2021,” *IT PRO*. <https://www.itpro.co.uk/security/cyber-attacks/361944/cyber-attacks-on-corporate-networks-increased-50-in-2021> [Accessed: 23-Jun-2022].
- [8] D. Temple-Raston, “A ‘Worst Nightmare’ Cyberattack: The Untold Story Of The SolarWinds Hack,” *NPR*, Apr. 16, 2021. <https://www.npr.org/2021/04/16/985439655/a-worst-nightmare-cyberattack-the-untold-story-of-the-solarwinds-hack> [Accessed: 23-Jun-2022].
- [9] Microsoft, “Visual Studio Code,” *Visualstudio.com*, Apr. 14, 2016. <https://code.visualstudio.com/docs/editor/whyvscode> [Accessed: 23-Jun-2022].
- [10] Python Software Foundation, “What is Python? Executive Summary,” *Python.org*, 2019. <https://www.python.org/doc/essays/blurb/> [Accessed: 23-Jun-2022].
- [11] “Built-in Types — Python 3.8.1rc1 documentation,” *Python.org*, 2019. <https://docs.python.org/3/library/stdtypes.html> [Accessed: 23-Jun-2022].
- [12] “Python Sets,” *W3schools.com*, 2020. https://www.w3schools.com/python/python_sets.asp [Accessed: 23-Jun-2022].

UNCLASSIFIED

- [13] “5. Data Structures — Python 3.8.3 documentation,” *docs.python.org*.
<https://docs.python.org/3/tutorial/datastructures.html> [Accessed: 23-Jun-2022].
- [14] “socket — Low-level networking interface — Python 3.8.1 documentation,” *Python.org*, 2020. <https://docs.python.org/3/library/socket.html> [Accessed: 23-Jun-2022].
- [15] “Python File Methods,” *www.w3schools.com*.
https://www.w3schools.com/python/python_ref_file.asp [Accessed: 23-Jun-2022].
- [16] “Downloads – Oracle VM VirtualBox,” *Virtualbox.org*, 2019.
<https://www.virtualbox.org/wiki/Downloads> [Accessed: 23-Jun-2022].
- [17] “welcome home : vim online,” *www.vim.org*. <https://www.vim.org/> [Accessed: 23-Jun-2022].
- [18] S. Allen, “Introduction to Networking,” in VICEROY Lecture Series, 18-Jul-2022.