# Network Video & Image Displayer Using Docker Containers

Erik S. LaNeave, Gilbert I. Guzman, Fernando H. Remes, & Jacob A. Gonzalez

Department of Computer Science, University of Texas at El Paso

CS 4316/CS 5313: Computer Networks [CRN: 21955]

Dr. Deepak Tosh

April 28, 2024

## Abstract

In the rapidly evolving landscape of computing, the secure and efficient transmission of digital data remains a critical challenge. This research project introduces a comprehensive solution; Network Video & Image Displayer Using Docker Containers (NVID-DC). Drawing inspiration from the topics covered in the team's Computer Networks class, the NVID-DC system has been built to be rapidly deployable, user-friendly, secure, and adaptable. Firstly, NVID-DC provides a rapidly deployable environment, by harnessing Docker Containers to create a networked video and image display platform. By encapsulating rendering processes within these small "clients", we achieve flexibility and scalability due to NVID-DC's small file size and low system requirements. Secondly, NVID-DC is User-Friendly, by using Python's Tkinter Web Framework, a simple front-end is created. Clients can seamlessly interact with any multimedia content sent to them via a GUI, so they can choose to display or send messages. Thirdly, NVID-DC provides Security, by implementing AES & RSA. This allows NVID-DC to ensure that its messages are encrypted and authenticated so that clients, their peers, and the server are safe. Lastly, NVID-DC is adaptable, by sending multimedia in real time to their assigned destinations. This allows NVID-DC to dynamically display what is sent through it, which means it can be used in a multitude of industries or use cases, such as corporate environments for presentations, public spaces for digital signage, or private homes for sharing messages or personal photos.

# Network Video & Image Displayer Using Docker Containers

## *Introduction*

In our interconnected world, information flows seamlessly between billions of people. Whether it's a million-dollar financial transaction between large banks or a delightful cat video sent to your significant other, digital data travels across routers, switches, and devices at every waking moment of the day. Yet, what if one wished to share a crucial file with another device wirelessly, but without relying on large central services like Microsoft Teams or Discord? Is there no robust solution to send something across your network, without exposing your entire machine to bad actors who may be lurking on your router?

Enter Network Video & Image Displayer Using Docker Containers (NVID-DC). NVID-DC began development to minimize dependence on large platforms for file transfer and media sharing for close family, friends, or colleagues. It takes inspiration from Peer-to-Peer Networking seen in platforms such as BitTorrent and other file sharing protocols, but uses a lightweight, central server to coordinate between other machines that may want to talk to one another. This server, hosted locally, serves as a client list and authenticator to ensure that clients get their information to who they need to, without exposing their entire machine to the entire network. It also acts as an "address book" allowing for users on the network to see each other and quickly figure out who they want to send data to.

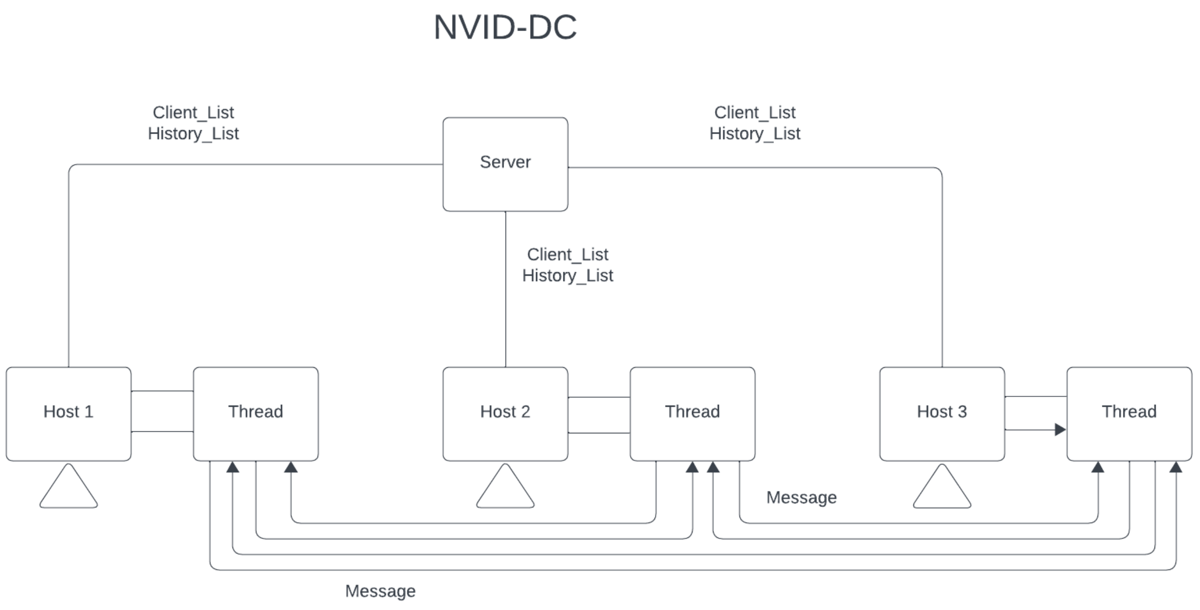
The team knew that if we wanted NVID-DC to fully align with our vision of privacy, speed, and use case robustness, we needed to use Docker containers. This is because the dockers we created would be incredibly lightweight, easy to deploy and have low system costs. In the spirit of Richard Stallman's quote, "Sharing is good, and with digital technology, sharing is easy," NVID-DC strives to make the sharing of digital data simple, and easy to use (Stallman, 2012).

## *Methodology*

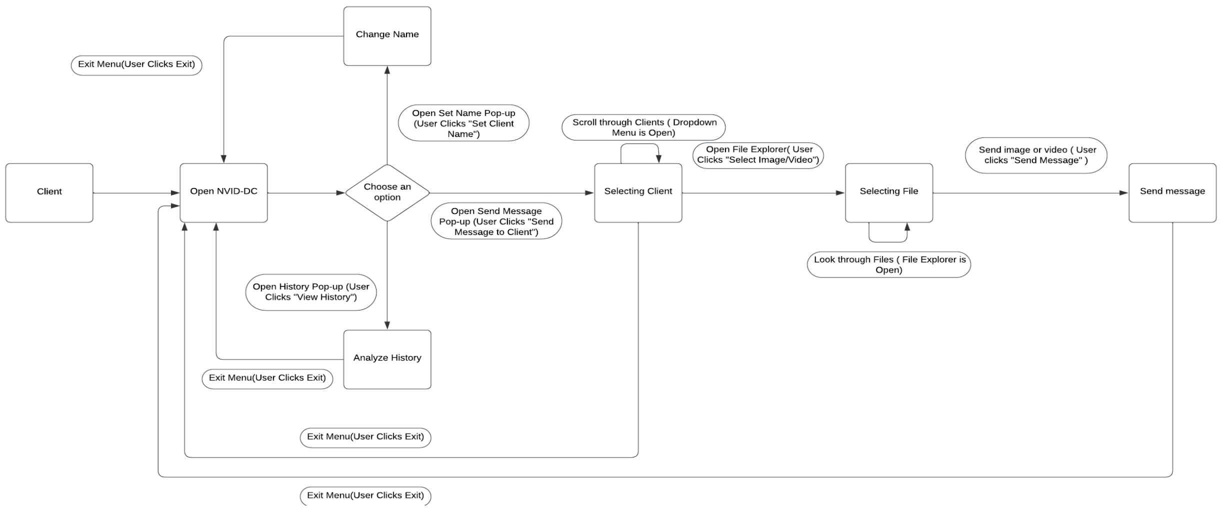
The discussion surrounding the use of Virtual Machines (VMs) versus Docker was the focal point at the infancy of NVID-DC. While virtual machines emulate the functionality of a physical server, able to emulate multiple different operating systems while ensuring security. However, Docker containers run as separate instances of our media sharing application, have fast boot times, and are less resource intensive than VMs, which make them a desirable means for architectures used in innovative application deployment. Unlike virtual machines, docker containers are cost effective and lightweight on the host machine. For instance, docker containers can start up and shut down much quicker due to minimal strain on the system resources. Additionally, with the inclusion of X server, dockers can run GUI applications, as X servers provide the foundation for GUI application rendering.

When it came to designing the Python interface for our product, Dash was the clear favorite since it could create web applications that were enhanced with interactive data visualizations. nevertheless, a deeper look uncovered that Dash's primary asset is its ability to plot results graphically via its interaction with Plotly.js. So, in terms of our requirements, Dash seemed not to be as reliable. Given that Tkinter is a pre-built Python library whose services are effective at creating simple, standard GUI applications that are ideal for Docker environments, we decided to research it as an alternative.

After, confirming the software tools that will be used to create NVID-DC. The next step was to create a diagram for how the server-to-client communicates and how a client-to-client communicates with each other.

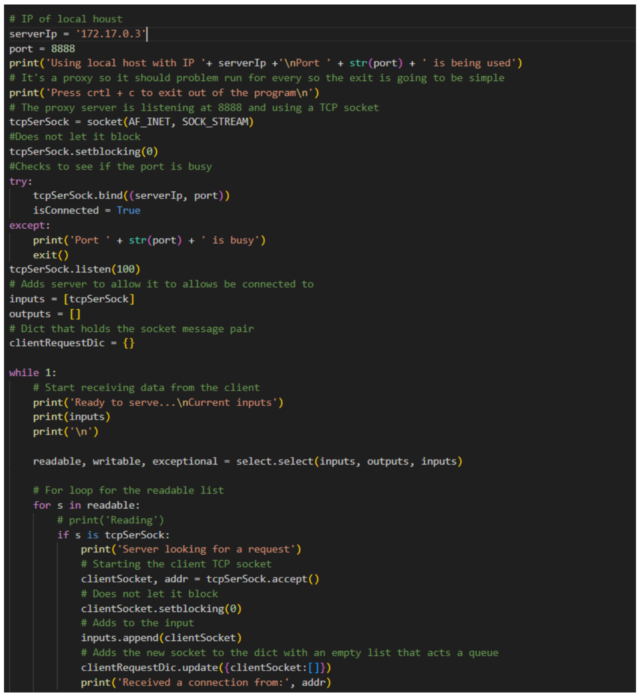


Afterward, a flow diagram was created of how the process of the client interacting with NVID-DC and most importantly how they go through to send an image client.


Client
Open NVID-DC
Choose an option
Change Name
Analyze History
Selecting Client
Selecting File
Send message
Exit Menu(User Clicks Exit)
Open Set Name Pop-up (User Clicks "Set Client Name")
Open Send Message Pop-up (User Clicks "Send Message to Client")
Open History Pop-up (User Clicks "View History")
Exit Menu(User Clicks Exit)
Scroll through Clients ( Dropdown Menu is Open)
Open File Explorer( User Clicks "Select Image/Video")
Look through Files ( File Explorer is Open)
Send image or video ( User clicks "Send Message" )
Exit Menu(User Clicks Exit)
Exit Menu(User Clicks Exit)

After the design process, next required the development process. The first step was creating the server to client communication. As a test case, we created a basic echo communication from one docker container to another.

Server:



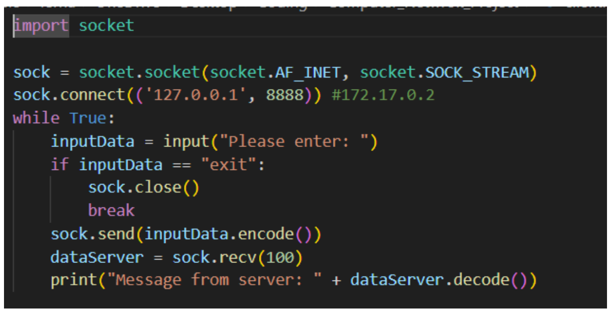
```
# IP of local houst
serverIp = '172.17.0.3'
port = 8888
print('Using local host with IP '+ serverIp +'\nPort ' + str(port) + ' is being used')
# It's a proxy so it should problem run for every so the exit is going to be simple
print('Press crtl + c to exit out of the program\n')
# The proxy server is listening at 8888 and using a TCP socket
tcpSerSock = socket(AF_INET, SOCK_STREAM)
#Does not let it block
tcpSerSock.setblocking(0)
#Checks to see if the port is busy
try:
    tcpSerSock.bind((serverIp, port))
    isConnected = True
except:
    print('Port ' + str(port) + ' is busy')
    exit()
tcpSerSock.listen(100)
# Adds server to allow it to allows be connected to
inputs = [tcpSerSock]
outputs = []
# Dict that holds the socket message pair
clientRequestDic = {}

while 1:
    # Start receiving data from the client
    print('Ready to serve...\nCurrent inputs')
    print(inputs)
    print('\n')

    readable, writable, exceptional = select.select(inputs, outputs, inputs)

    # For loop for the readable list
    for s in readable:
        # print('Reading')
        if s is tcpSerSock:
            print('Server looking for a request')
            # Starting the client TCP socket
            clientSocket, addr = tcpSerSock.accept()
            # Does not let it block
            clientSocket.setblocking(0)
            # Adds to the input
            inputs.append(clientSocket)
            # Adds the new socket to the dict with an empty list that acts a queue
            clientRequestDic.update({clientSocket:[]})
            print('Received a connection from:', addr)
```

Client:

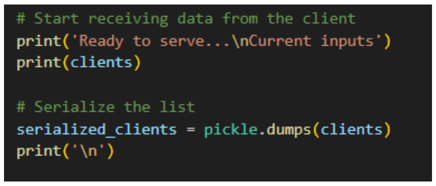


```
import socket

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(('127.0.0.1', 8888)) #172.17.0.2
while True:
    inputData = input("Please enter: ")
    if inputData == "exit":
        sock.close()
        break
    sock.send(inputData.encode())
    dataServer = sock.recv(100)
    print("Message from server: " + dataServer.decode())
```

Next was to develop the server to send the list of current clients within it. Thus, the objective was to have the client send a message to the server, and the server responded with the client list. Not only was this successful, but it also helped us understand that a server needs to serialize large packets of messages to send to the client, where the packet would need to deserialize the message. The decision was to use the Python built-in feature to pickle large packets. The pickle module implements binary protocols for serializing and deserializing a Python object structure. This helped to send large pictures or videos through the network.

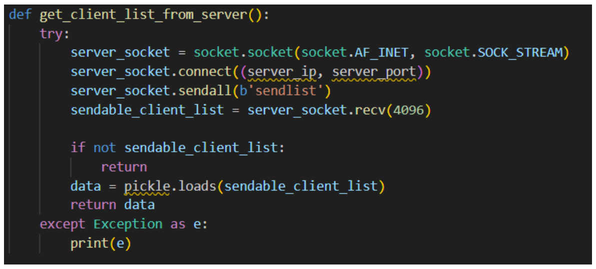
Server:



```
# Start receiving data from the client
print('Ready to serve...\nCurrent inputs')
print(clients)

# Serialize the list
serialized_clients = pickle.dumps(clients)
print('\n')
```

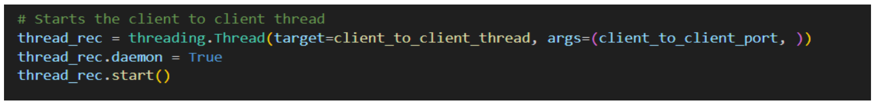
Client:



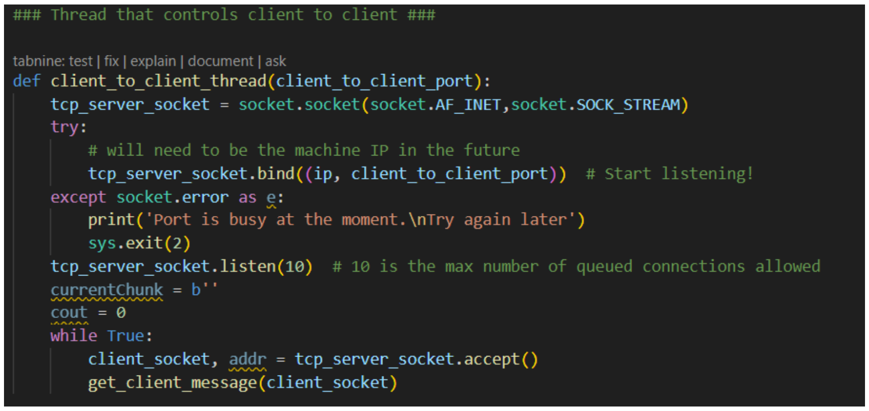
```
def get_client_list_from_server():
    try:
        server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        server_socket.connect((server_ip, server_port))
        server_socket.sendall(b'sendlist')
        sendable_client_list = server_socket.recv(4096)

        if not sendable_client_list:
            return
        data = pickle.loads(sendable_client_list)
        return data
    except Exception as e:
        print(e)
```

The following step was to set up client-to-client communication. The peer-to-peer network, which consists of two or more computers sharing files and accessing devices, served as the foundation for client-client communication. Furthermore, threading has been shown to be useful for managing multiple clients. subsequently also met the requirement of a lightweight processor with minimal memory cost and assisted in assigning ports to clients.



```
# Starts the client to client thread
thread_rec = threading.Thread(target=client_to_client_thread, args=(client_to_client_port, ))
thread_rec.daemon = True
thread_rec.start()
```

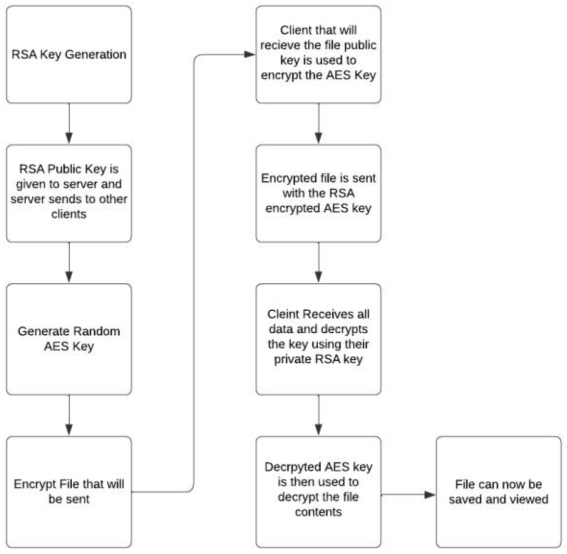


```
### Thread that controls client to client ###

tabnine: test | fix | explain | document | ask
def client_to_client_thread(client_to_client_port):
    tcp_server_socket = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
    try:
        # will need to be the machine IP in the future
        tcp_server_socket.bind((ip, client_to_client_port))  # Start listening!
    except socket.error as e:
        print('Port is busy at the moment.\nTry again later')
        sys.exit(2)
    tcp_server_socket.listen(10)  # 10 is the max number of queued connections allowed
    currentChunk = b''
    cout = 0
    while True:
        client_socket, addr = tcp_server_socket.accept()
        get_client_message(client_socket)
```

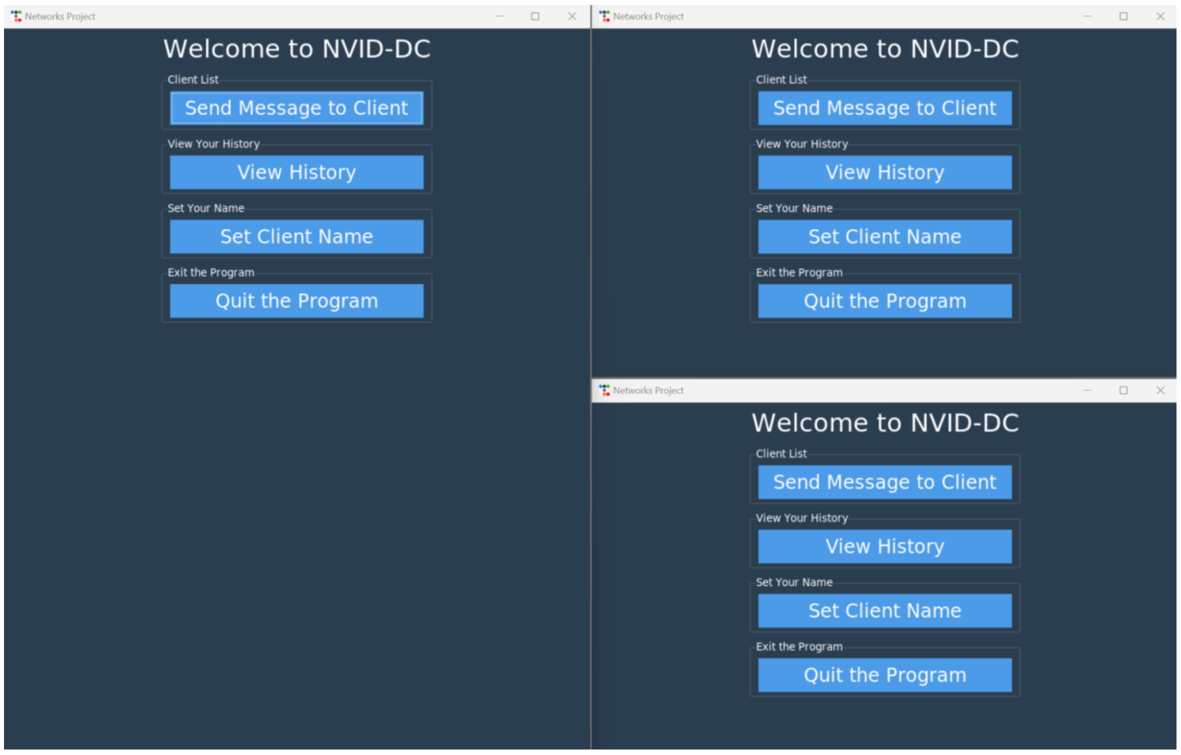
Upon verifying that both the server-to-client and client-to-client communication met the requirements, the process of bringing the users' ability to exchange photographs and videos into practice was put into place. As a result, it became possible to improve image encoding and share videos amongst customers.

As we integrated the front-end and back-end, the next objective was to improve the system's user interface and take care of any bugs that became apparent. One of the bug fixes encountered during the integration of both the front-end and back-end was the lack of security in client-to-client communication, as it became apparent that a separate client could potentially interfere with network communication. Therefore, the solution was to encrypt the image and video files using AES. After implementing AES, a secure method for sending the keys between sender and receiver would be needed. RSA was used to generate a public and private key for each host that connects and wants to use the application. The public key was sent to the server to be distributed among the different clients, when another client wants to send a message the public key will be used to encrypt the AES key used. The encrypted AES key can only be decrypted by that client with their private key. This implementation now ensures that some form of security is protecting the client’s image or video transmissions between each other.

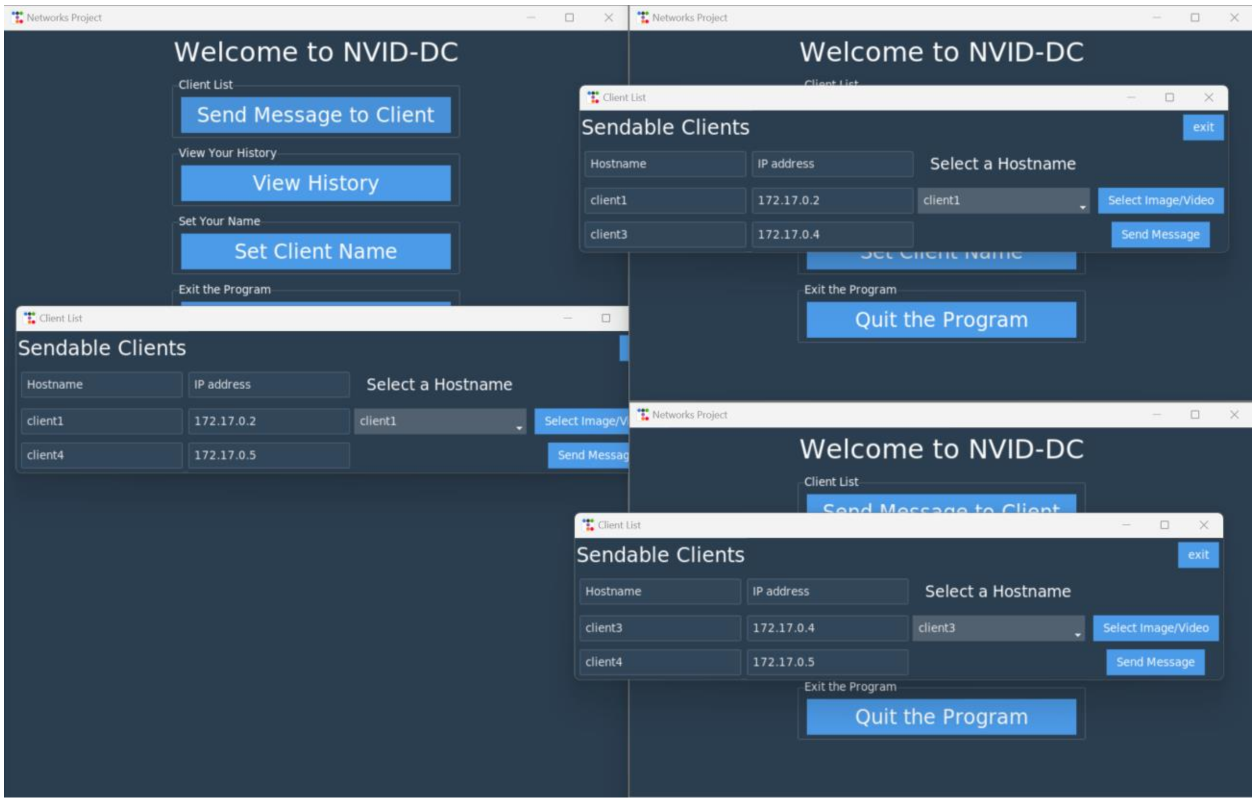

RSA Key Generation
RSA Public Key is given to server and server sends to other clients
Generate Random AES Key
Encrypt File that will be sent
Client that will recieve the file public key is used to encrypt the AES Key
Encrypted file is sent with the RSA encrypted AES key
Cleint Receives all data and decrypts the key using their private RSA key
Decrpyted AES key is then used to decrypt the file contents
File can now be saved and viewed

***Results With Test Cases & Execution Samples***

The following results showcase the robustness of the application to handle errors, along with the effectiveness of the protocol to facilitate the successful and secure transmission of the clients desired media file to other clients.



The figure above illustrates the scalable nature of the environment that was used to test and run NVID-DC. Four docker containers are operating in the figure above, one is running the server that is providing connectivity between clients, and the other three are running sperate instances of NVID-DC. This allowed for easy and straightforward testing of the system while all being the same host. The number of containers running NVID-DC could be greatly increased with one of the only limitations being the hardware of the host machine.



| Hostname | IP address |
|---|---|
| client1 | 172.17.0.2 |
| client3 | 172.17.0.4 |

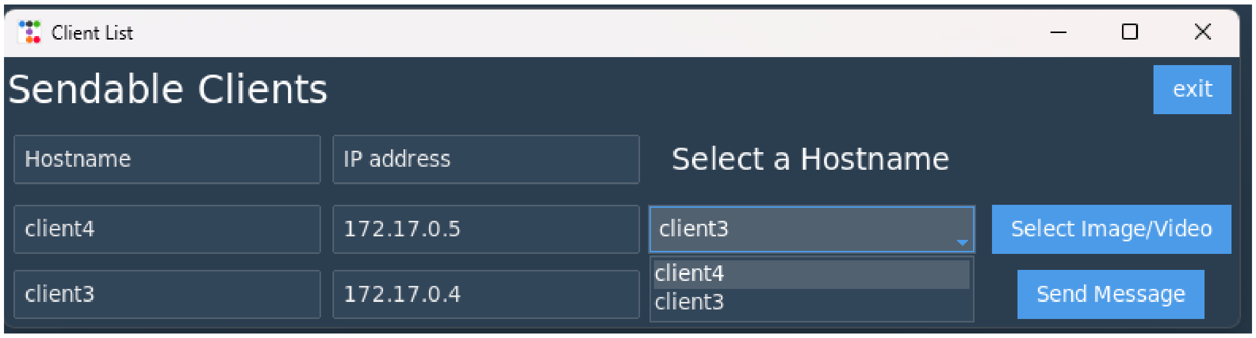
Select a Hostname: client1

| Hostname | IP address |
|---|---|
| client1 | 172.17.0.2 |
| client4 | 172.17.0.5 |

Select a Hostname: client1

| Hostname | IP address |
|---|---|
| client3 | 172.17.0.4 |
| client4 | 172.17.0.5 |

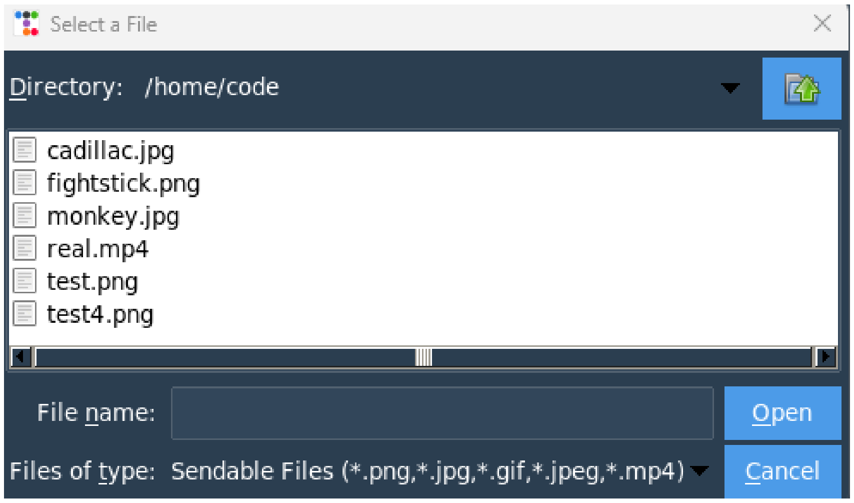
Select a Hostname: client3



| Hostname | IP address |
|---|---|
| client4 | 172.17.0.5 |
| client3 | 172.17.0.4 |

Select a Hostname: client3 (client4, client3)

The purpose of these figures is to further demonstrate that each container is sperate from each other, but all are connected to the same central server. This is seen each of the containers sendable client’s popup that shows who they can send a message to, this list is provided and maintained by the server.



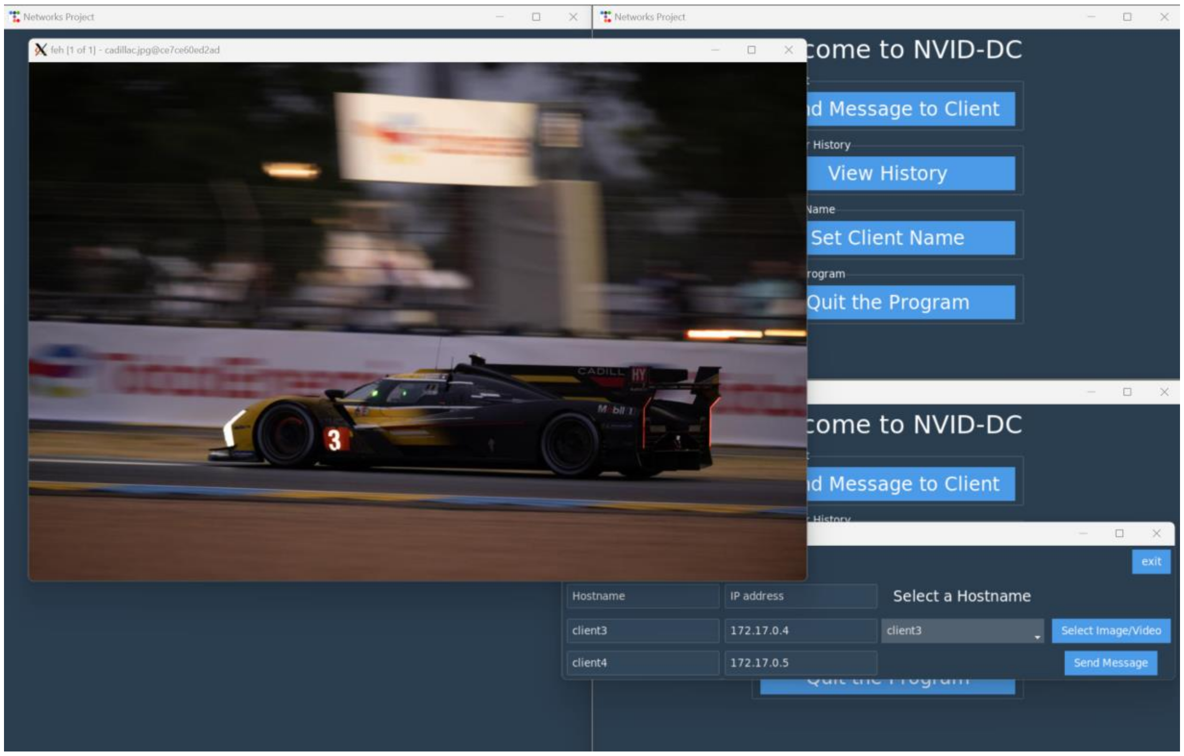
Directory: /home/code

cadillac.jpg
fightstick.png
monkey.jpg
real.mp4
test.png
test4.png

File name:

Files of type: Sendable Files (*.png,*.jpg,*.gif,*.jpeg,*.mp4)

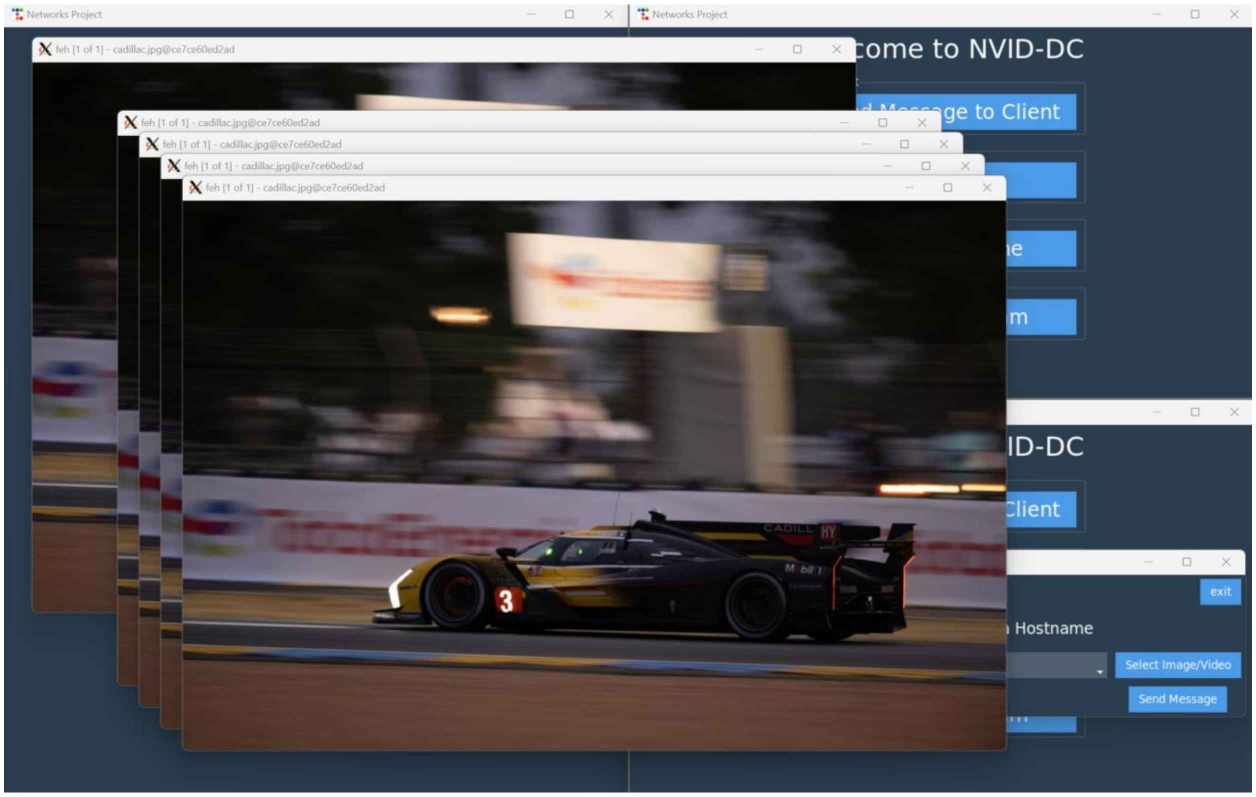
In the previous figure the client can and was selected by the user. Once a client is determined the user will then select the file they wish to send. The user is limited to pick files that are either images or videos. This ensures that NVID-DC will not receive a file format it can’t handle, which acts as a simple form of error prevention.



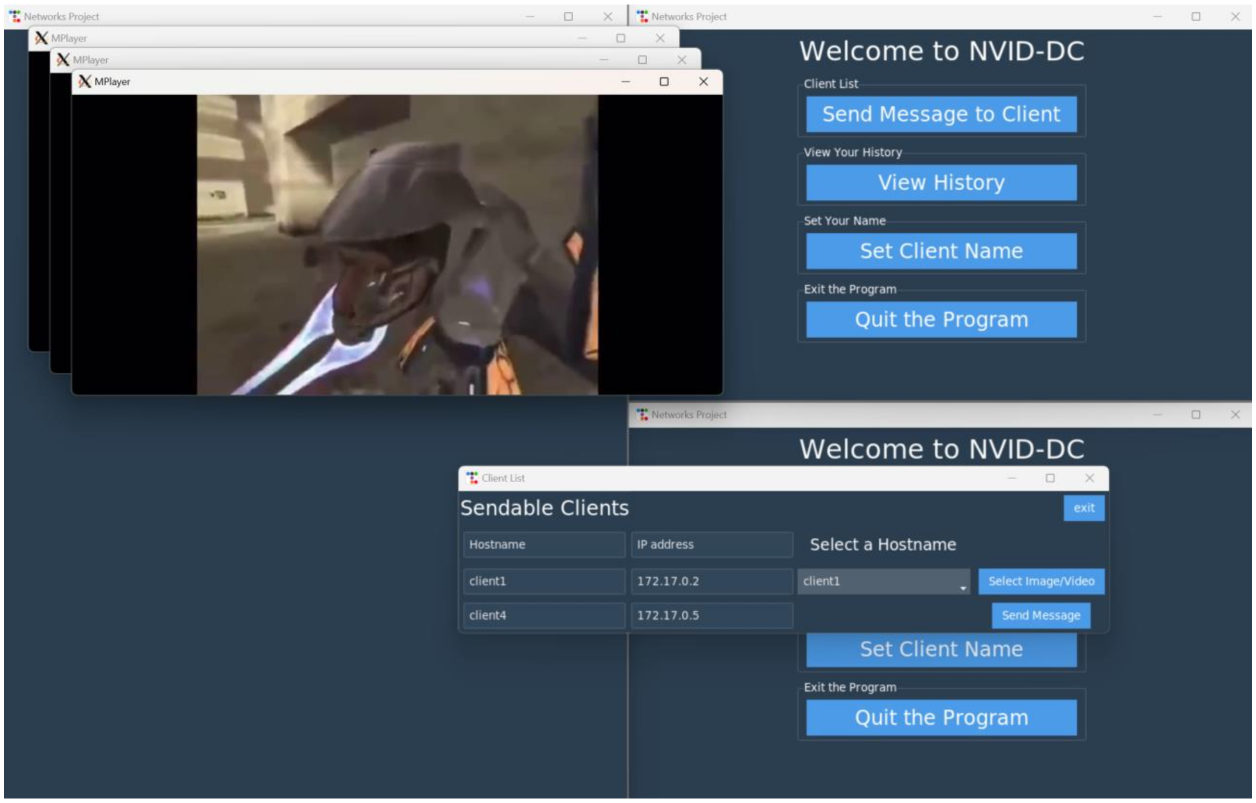
| Hostname | IP address |
|---|---|
| client3 | 172.17.0.4 |
| client4 | 172.17.0.5 |

client3

Once the file has been selected, encrypted, and sent to the chosen client. When receiving client will immediately display the sent image to the user.



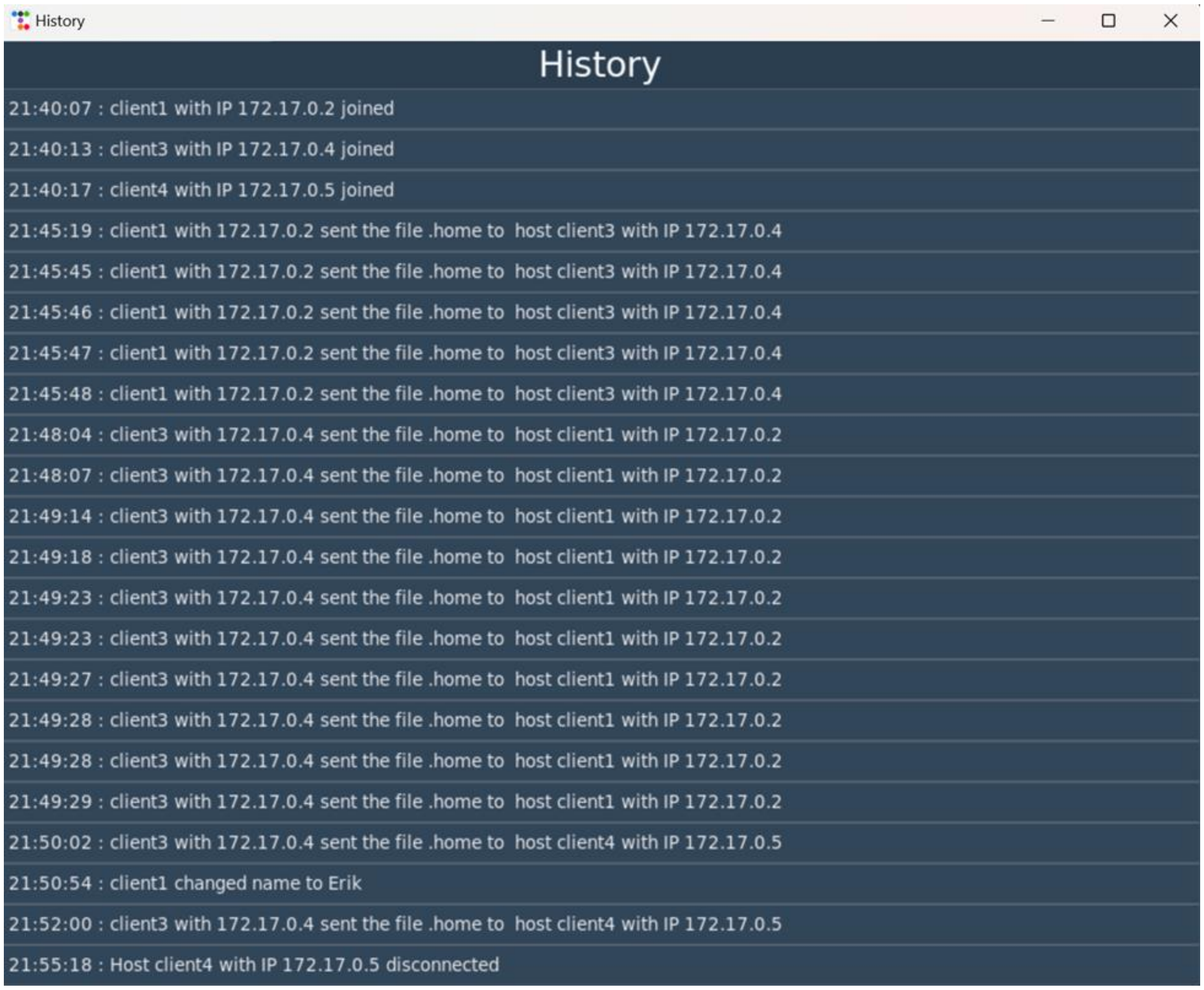
The sending client can either resend the same image or a different image as many times as they would like, and the receiving client will continue to display the images. The receiving client can take in images from any other client at the same time and display those images to the users. This functionality is by design and demonstrations that a client can handle many messages and message types at a time.



| Hostname | IP address |
|---|---|
| client1 | 172.17.0.2 |
| client4 | 172.17.0.5 |

client1

This figure shows the same functionality of NVID-DC as discussed in the figure above but instead with videos. Each time the video is sent the receiving client will open it and immediately play the video till it is complete. The container can handle many video messages, but they do take more resources compared to the images. On top of showing that NVID-DC can handle multiple messages it also shows that it can display and play videos.

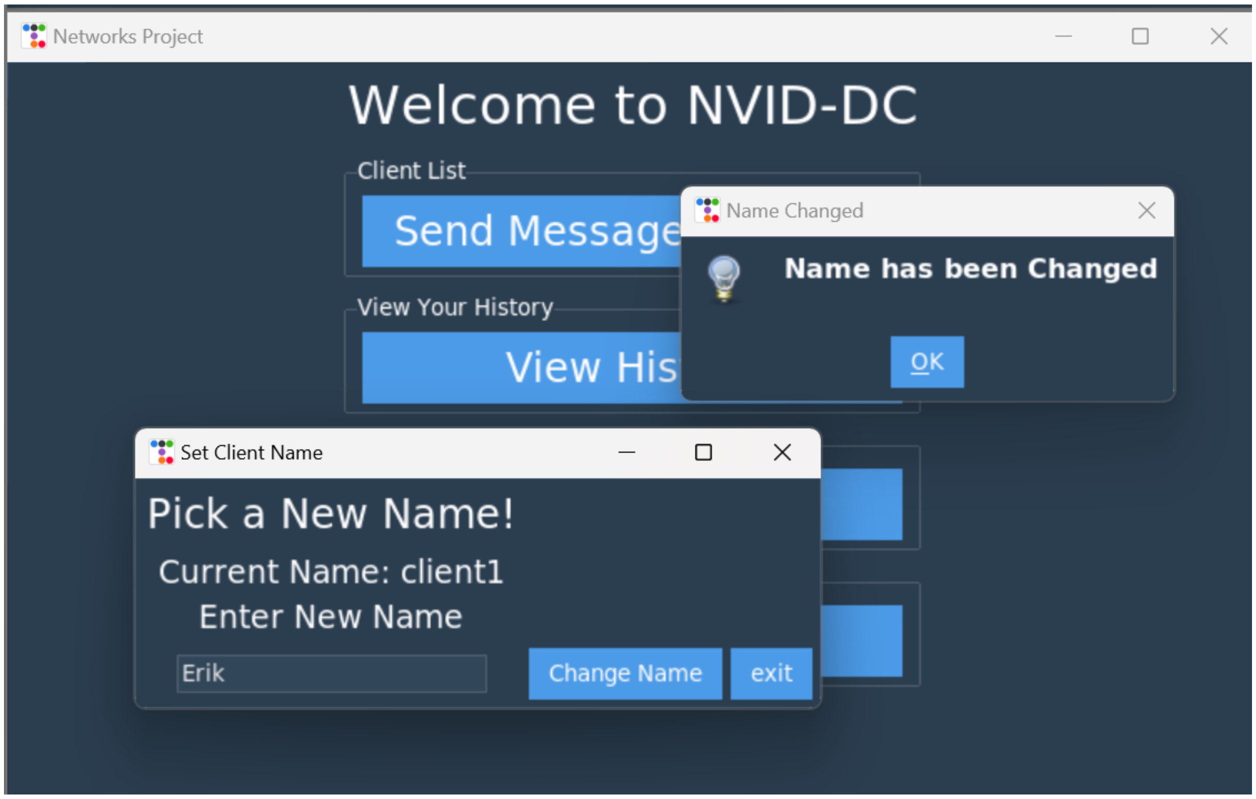


History

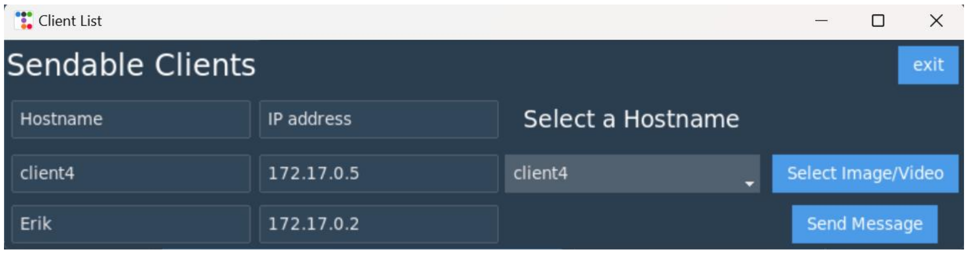
History

```
21:40:07 : client1 with IP 172.17.0.2 joined
21:40:13 : client3 with IP 172.17.0.4 joined
21:40:17 : client4 with IP 172.17.0.5 joined
21:45:19 : client1 with 172.17.0.2 sent the file .home to  host client3 with IP 172.17.0.4
21:45:45 : client1 with 172.17.0.2 sent the file .home to  host client3 with IP 172.17.0.4
21:45:46 : client1 with 172.17.0.2 sent the file .home to  host client3 with IP 172.17.0.4
21:45:47 : client1 with 172.17.0.2 sent the file .home to  host client3 with IP 172.17.0.4
21:45:48 : client1 with 172.17.0.2 sent the file .home to  host client3 with IP 172.17.0.4
21:48:04 : client3 with 172.17.0.4 sent the file .home to  host client1 with IP 172.17.0.2
21:48:07 : client3 with 172.17.0.4 sent the file .home to  host client1 with IP 172.17.0.2
21:49:14 : client3 with 172.17.0.4 sent the file .home to  host client1 with IP 172.17.0.2
21:49:18 : client3 with 172.17.0.4 sent the file .home to  host client1 with IP 172.17.0.2
21:49:23 : client3 with 172.17.0.4 sent the file .home to  host client1 with IP 172.17.0.2
21:49:23 : client3 with 172.17.0.4 sent the file .home to  host client1 with IP 172.17.0.2
21:49:27 : client3 with 172.17.0.4 sent the file .home to  host client1 with IP 172.17.0.2
21:49:28 : client3 with 172.17.0.4 sent the file .home to  host client1 with IP 172.17.0.2
21:49:28 : client3 with 172.17.0.4 sent the file .home to  host client1 with IP 172.17.0.2
21:49:29 : client3 with 172.17.0.4 sent the file .home to  host client1 with IP 172.17.0.2
21:50:02 : client3 with 172.17.0.4 sent the file .home to  host client4 with IP 172.17.0.5
21:50:54 : client1 changed name to Erik
21:52:00 : client3 with 172.17.0.4 sent the file .home to  host client4 with IP 172.17.0.5
21:55:18 : Host client4 with IP 172.17.0.5 disconnected
```

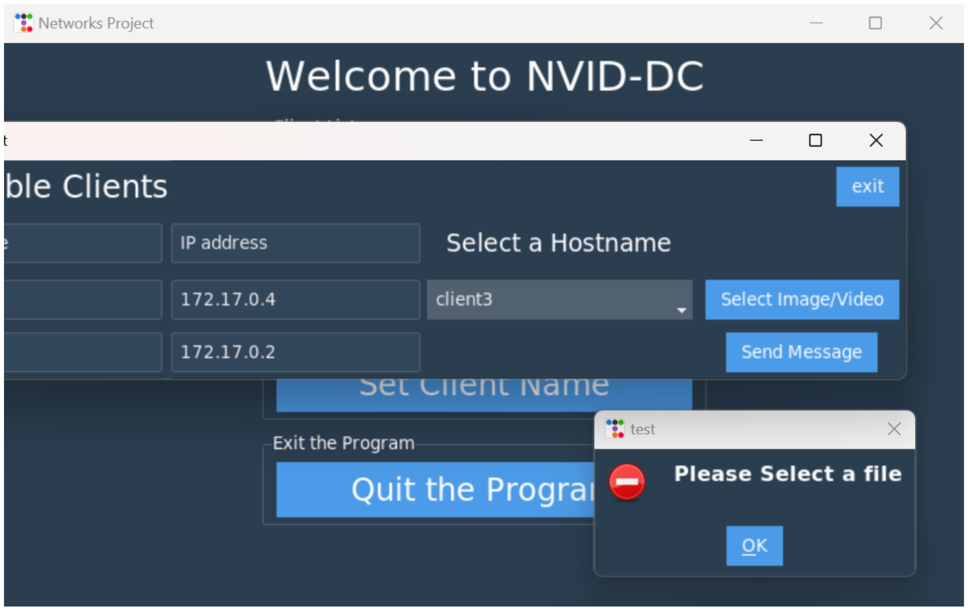
During the execution of NVID-DC, the client will send records of what they are sending to other clients, while the server records updating actions like clients joining, leaving, or changing their name. The history record that a client can view is a visual of what is taking place in NVID-DC, and it also displays that the protocol is working correctly. If the internal protocol was not working, then the updates would not take place.



Clients will have a default name when they initially join the service, an option to change that name is provided. This did require special messages in the protocol and more rules in the server. The reason for this is because the hostname with the IP is used to maintain the client list on the server side. Additional logic was needed to ensure that other clients don't forget about the client because of the name change. The history figure and the figure below of other containers sendable list illustrate that the protocol is successfully in allowing a name change.



| Hostname | IP address |
|---|---|
| client4 | 172.17.0.5 |
| Erik | 172.17.0.2 |



The last figure again shows the robustness of the system in handling errors that could appear when selecting files through the GUI. The file explorer used in the UI caused errors in NVID-DC when porting the system from the command line to the GUI. This was quickly fixed and made the UI more usable in the process.

## *Individual Contributions*

### *Erik S. LaNeave*

I handled the set up for the different testing environments used throughout the development of NVID-DC. This includes building and maintaining the many different docker containers that were needed to ensure that NVID-DC operated in the correct and predictable manner. The most challenging part of getting the environment set up was getting the many different docker containers to all use the same Xserver to display their UIs and images/videos. Once the environment was stable real work could be done on the back end and the design of the protocol. As the back-end lead, I made major contributions to the structure, design, and implementation of both the client and server side of NVID-DC. I decided that threading would have to be used on the client side to ensure that communications aren't missed while running the UI. I came up with the initial protocol that described how the server would talk to clients and how clients would talk to clients. I helped in making the switch from dash by setting up a skeleton GUI in Tkinter and assisted with improving the GUI. I also set up the client-to-client security with RSA and AES to have encryption in NVID-DC.

When we began to work hard on the code, I became the manager of the GitHub for the project and tried to always have up-to-date code available to the team. Most of my contributions were focused on getting the system running, mostly by coding and getting the environment set up. As the back-end lead, I spent my time coding the server, client, client to client thread, and getting the protocol for NVID-DC working. I designed a diagram that shows the encryption process when sending messages between clients and provided/wrote about the images in the results section of the report. This project would not have been possible without the other members of the team.

### *Gilbert Guzman*

I began this project as the main communications coordinator, creating the starter documents for this project and recruiting my teammates to the group. We attended our first meeting, where we decided to try to pitch this idea to our professor, which had me creating the initial document pitch that was sent via email for verification of the idea. When it was accepted and progress was made, I created the Microsoft Teams for this project and consolidated all of members into it to use as a secondary communication to Discord, which we all had already added each other on. This Microsoft Teams was used for the upload and collaboration of our members across multiple submittables, all of which were formatted by myself. This was done to ensure I fulfilled my role as Report Lead.

In addition to the initial pitch document, this included our two update PowerPoints, (where we showcased progress to the professor up until we had a substantial test build), and the initial research on Tkinter implementation. I also worked as the coordinator for the finalized research report and PowerPoint we will be showcasing in class, which I have spent a decent amount of time creating and refining to ensure compliance with all formatting expectations is met. In the case of our Project PowerPoint, my contributions were the slide order, design, half the content, formatting, and all APA citations. The report saw me contributing to the format, the abstract, the introduction, and all APA citations. I also helped develop vertical slices by providing bug fixes and brainstorming ideas for what NVID-DC should be capable of, but most of the substantial progress in programming was done by my teammates. For this project, I was able to fulfill the role I was originally assigned, and I know I was a great asset to my team.

#### ***Fernando H. Remes***

Initially, I assumed the role of Test Lead, where I would evaluate the program's functionality and oversee ensuring it adhered to specified requirements. However, as the project progressed, my responsibilities expanded to be leader of front-end development and design. Crafting intuitive user interfaces using Tkinter Bootstrap and creating a comprehensive flow chart diagram and ensured user friendly interaction with NVID-DC.

Nevertheless, I remained focused on the insurance of the server-to-client communication was running smoothly. For instance, I concentrated on optimizing server-to-client communication, tackling challenges such as recurrent reconnect attempts causing the server to increment client list even though there would be only one client. Also, during the documentation process I assisted in the methodology, which includes describing our research and design process done before developing the program. Thus, as Test and Front-End Lead, I not only contributed to the technical robustness and functionality of the project, such as refining or refactoring communication protocols, but also played a role in enhancing its overall user-friendly experience and interface design.

***Jacob Gonzalez***

One of my main contributions to the project was my designs for the initial graphical interface that we were initially trying although we eventually switched tools for the gui upon switching to Dockers. I also assisted wherever possible by assisting and researching different tools to try to find the best ones and assisted with the creation of the presentation and report. My role in the team fluctuated based on what was needed originally, I was meant to be a back-end support for the servers and clients helping make sure everything ran smoothly but as we went along I transitioned to researching different ways to create a gui and made a mock up design of possible ways we could connect the back end to the front end. As we came closer to the end of the project, I transitioned from gui to helping with the Documentation side of the project drafting flowcharts and filling out the presentation slides as well as working on the Conclusion and Future direction parts of the report with input from my team member.

**Conclusion**

Throughout this project, we were constantly learning new things. From learning how to use Dockers to sending messages and connecting clients to each other in a peer-to-peer manner through a server. We even learned how to better work in a group by managing our time, scheduling meetings, setting deadlines, and dividing up the work. One of the main things we learned about during this project was Docker. Getting to know all about containers and their advantages over virtual machines for certain applications was both exciting and frustrating. Dockers, even though being a much better tool to use for our project came with its own setbacks and difficulty having to learn a whole new tool is always tough but the advantages of it far outweighed any difficulties we had. In addition to Docker, we also learned two different ways to code the graphical interface. Originally, the team used Dash but had several issues getting it to connect to Docker, so we ended up switching to Tkinter, which is what we ended up using in the end.

This project was a great way for us to supplement classroom learning with outside knowledge due to the amount of research we had to do and what we learned from our homework. It also allowed us to learn better research techniques and strategies to problem solve. Many of the issues we had to overcome in this project correlated closely with what we were learning and being able to put into practice the things we were learning was a great way to cement the knowledge in our minds. Overall, this project was a great learning experience, while being challenging and was a fun way to implement what we were learning in class.

## Future Directions

A lot of research that went into this project is also applicable in many other applications and projects. One of the biggest things that we learned how to use and got experience with was Docker. There are many situations where Docker would be a much better option than using virtual machines. For example, in software development 2 the customer wants the final project to be implemented with Docker. This project is also a large steppingstone into the world of peer-to-peer data transfer, not only how it is done but as well as possible security methods we can use to protect any data transferred. Not only did we learn many new things about computer networks, but we also learned how to work better in a team and manage our time with little direction from a professor or supervisor. We learned how to manage a project and how to coordinate with others to ensure that everything gets done both correctly and timely.

Our project could also be a good basis for future projects that might require peer to peer data transfer of images or videos. There are even a couple ways that given more time, knowledge, and research we could improve what we have. One way we could build upon our project was if we were to encrypt the messages between the server and clients instead of leaving them plain text. Another way we could improve on the project would be by trying different security methods to find which one works best for both efficiency and security. Additionally, we could also change the project to allow for more file types than what is currently accepted by the code. We could also expand on the GUI making it more robust and fix any issues that there might still be in the code. While our code adequately does what we wanted there are still multiple ways that given more time, or knowledge we could improve but, but that could be said for most projects that have been created there is always room for improvement.

## References

Amos, D. (2019). Python GUI Programming with Tkinter. Retrieved from https://realpython.com/python-gui-tkinter/

Docker Forums. (2013). *How to EXPOSE Port on running container*. Retrieved from https://forums.docker.com/t/how-to-expose-port-on-running-container/3252/1

Du, W. (2022). Internet Security: A Hands-on Approach (3rd ed.). Independently Published. ISBN: 1733003967.

Kurose, J. F., & Ross, K. W. (2024). Computer Networks: A Top-Down Approach (8th ed.). Pearson Education. ISBN: 9356061319.

Peterson, L. L., & Davie, B. S. (2019). Computer Networks: A Systems Approach (6th ed.). Pearson Education. ISBN: 0128182008.

Plotly Community Forum. (2018). Running Dash App in Docker Container. Retrieved from https://community.plotly.com/t/running-dash-app-in-docker-container/16067

Potatowagon. (2020). How to Use GUI Apps in Linux Docker Container from Windows Host. Retrieved from https://medium.com/@potatowagon/how-to-use-gui-apps-in-linux-docker-container-from-windows-host-485d3e1c64a3

Python Software Foundation. (n.d.). tkinter — Python interface to Tcl/Tk — Python 3.12.3 documentation. Retrieved from https://docs.python.org/3/library/tkinter.html

Stack Overflow. (2012). Does Tkinter have a table widget? Retrieved from https://stackoverflow.com/questions/9348264/does-tkinter-have-a-table-widget

Stallman, R. (2012). Sharing is good, and with digital technology, sharing is easy. QuoteFancy. Retrieved from https://quotefancy.com/quote/1425553/Richard-Stallman-Sharing-is-good-and-with-digital-technology-sharing-is-easy