International Journal of Science and Research (IJSR)
ISSN: 2319-7064
Impact Factor 2024: 7.101

Client - Server Based Remote Screen Sharing System Using VB.Net

Dr. A. Karunamurthy[1], R. Keerthana[2]

[1]Associate Professor, Department of Computer Applications, Sri Manakula Vinayagar Engineering College (Autonomous), Puducherry 605008, India
Email: *karunamurthy26[at]gmail.com*

[2]Post Graduate Student, Department of Computer Applications, Sri Manakula Vinayagar Engineering College (Autonomous), Puducherry 605008, India
Email: *keerthanaram13[at]gmail.com*

Abstract: ***In this project, a simple screen - sharing application is developed using Visual Basic (VB. NET) with Windows Forms. The system is based on a client - server model. The server captures the screen of the host computer at regular intervals and sends it over the network to the connected client, which then displays the received image in real time. The screen capture is performed using the Graphics. CopyFromScreen () method, and the image is converted to a byte array using a memory stream and JPEG format to reduce size. This data is sent through a TCP connection. On the client side, the image is received and displayed in a PictureBox control. The project uses NetworkStream, BinaryWriter, and BinaryReader classes for sending and receiving data efficiently. The goal of the project is to demonstrate how basic screen sharing can be implemented using socket programming in VB. NET. This type of system can be extended for remote desktop access or online assistance tools. It helps in understanding concepts like TCP/IP, threading, image compression, and real - time data transfer.***

Keywords: Screen Sharing, Socket Programming, Client - Server Model, VB. NET, TCP Connection, Remote Desktop Simulation.

1. Introduction

In today's digital world, remote access and screen sharing have become essential tools for communication, technical support, and collaboration. This project is focused on building a basic screen sharing system using VB. NET and the client - server architecture. The main goal is to capture the live screen of a server machine and send it to a client over a network connection. This allows the client to view the screen of the remote computer in real time. The application uses TCP sockets to establish a connection between the server and client, ensuring smooth and reliable data transmission.

The server captures the screen continuously and sends the image frames to the client using a network stream. On the other side, the client receives these image frames and displays them using a PictureBox control. The images are compressed into JPEG format to reduce size and improve transmission speed. This project helps in understanding the basics of socket programming, image handling, and multithreading in VB. NET. It also gives a practical idea of how screen sharing applications like remote desktop tools work internally.

2. Problem Statement

In various real - life scenarios such as technical help, online guidance, or remote monitoring, there is a frequent need to access or observe the screen of another computer. Many available screen - sharing tools are either too complex, require heavy installations, or offer limited flexibility for learning or customization. Beginners and students often find it challenging to understand how these advanced systems function due to their complicated structure.

To address this issue, the proposed project presents a simple and easy – to understand screen - sharing application built using VB. NET. The application captures the current display from a host (server) system and transmits it over a network to a connected client using TCP socket communication. The client then receives and renders the screen continuously, giving a real - time view of the server. This solution offers a basic and effective way to learn about network programming, screen capturing, and real - time data transfer in a minimal environment.

3. Literature Survey

Screen sharing and remote desktop technologies have been explored in various forms over the years. In early systems, screen capturing was implemented using native APIs with limited support for real - time data transmission [1]. Modern solutions like TeamViewer and AnyDesk provide advanced functionalities but involve complex architectures and require internet - based authentication [2]. Research in socket programming has shown that TCP connections are effective for sending data reliably between two endpoints in a network, especially when dealing with continuous data streams like images or video frames [3]. Studies also indicate that compressing images before transmission helps in reducing latency and bandwidth usage during remote viewing [4]. Several educational projects have attempted simple screen - sharing models using languages like Java or Python, but fewer examples exist using VB. NET, making this project a useful reference for learners [5]. Some experiments with multi - threaded applications have shown better performance in handling continuous data flow and parallel processing for live screen updates [6]. According to recent academic work, developing a basic screen transmission system helps students understand core networking and data transfer concepts [7]. Furthermore,

literature suggests that implementing lightweight tools for specific use - cases (like LAN - based monitoring) reduces dependency on third - party software and encourages custom - built solutions for educational or internal use [8]. In some case studies, LAN - based screen sharing systems have demonstrated better privacy and speed compared to cloud - based tools [9]. Researchers have also emphasized the importance of designing user - friendly interfaces for real - time applications to improve usability and adoption [10]. Some open - source projects have focused on frame - by - frame image streaming using sockets, highlighting the trade - off between quality and transmission speed [11]. Lastly, screen sharing projects have proven to be excellent learning tools for beginners to understand client server architecture and real - time communication using basic programming languages [12].

4. Proposed System

Step 1: Start the Server:
The server application begins by opening a TCP listener on a specific port (e. g., 62616). This means the server waits and listens for any client that wants to connect.
Example: When you run the server program and press the "Start Server" button, it displays a message like "Server started. " The program then waits for any client on the network that tries to connect to port 62616.

Step 2: Capture the Screen:
Once the server is running, it captures the current display screen at regular intervals. This is done by taking a snapshot of the screen (like a screenshot) using built - in graphics methods in VB. NET. The captured image represents what the user currently sees on the server's monitor.
Example: Imagine pressing the "Start Server" button and the server taking a snapshot of your entire desktop every 100 milliseconds (10 times per second).

Step 3: Convert and Compress the Image:
The raw screenshot image is often very large in size. To send it efficiently over the network, the server converts this image to a compressed format like JPEG, which reduces the file size significantly without losing much quality. This step helps reduce the bandwidth needed and speeds up the transmission.
Example: Suppose the raw screenshot is 5 MB. After converting to JPEG, it might reduce to 500 KB. The server compresses the image in memory before sending it to the client, making the transmission faster and smoother.

Step 4: Establish Connection with Client:
The client application starts and attempts to connect to the server using the server's IP address and the port number the server listens on. This connection is essential for the client to receive the screen data sent by the server.
Example: You open the client program on another computer in the same network and click the "Connect" button. You enter the server IP "192.168.1.68" and port "62616". The client then connects successfully and is ready to receive screen images.

Step 5: Send Image Data over TCP:
The server sends the image data to the client through the TCP socket connection. First, it sends the size (length) of the image data so that the client knows how many bytes to read. Then, it sends the actual image bytes (the compressed JPEG data).
Example: If the JPEG image size is 500 KB, the server sends the number 512000 (bytes) first, then sends the 512000 bytes of image data. The client waits until it has received all 512000 bytes before proceeding.

Step 6: Receive and Display at Client:
The client reads the incoming data by first reading the size of the image, then reading the exact number of bytes for the image. It reconstructs the image from these bytes and displays it inside a PictureBox control on the client's user interface, showing what the server screen currently looks like.
Example: The client receives 512000 bytes from the server, reconstructs the image, and displays it inside the PictureBox on the client window. If the server's screen changes, the client's PictureBox updates accordingly, showing the latest screen view.

5. System Architecture

The system architecture diagram illustrates the working of a simple screensharing application using a client - server model. On the server side, the process begins with the **Start Server** module, which initializes the server to accept incoming connections. The server then **captures the screen** of the host computer in real time using screen capturing techniques. Once captured, the image is **converted to JPEG** format to compress the data for efficient transmission. The server then **creates a TCP connection** using socket programming, which serves as the communication link between the server and the client

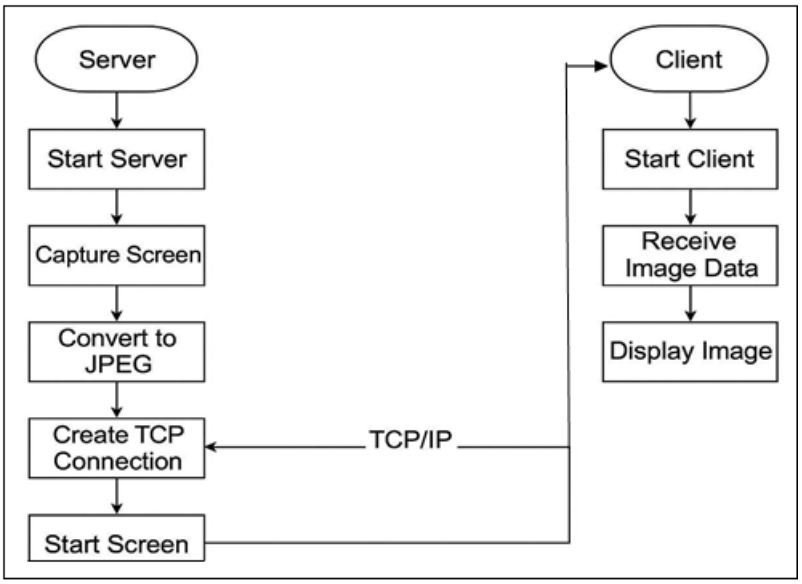

Figure 1: System Architecture

The entire process is executed in a loop to ensure continuous screen sharing. The diagram is structured in a clean and monochrome format, showing the logical flow of data and the key modules involved in both the server and client sides of the system.

6. Result and Discussion

The implementation of the screen - sharing system using TCP sockets was successful. When the server application is initiated, it begins capturing the screen at regular intervals and transmits the image data to any connected clients over the specified port.

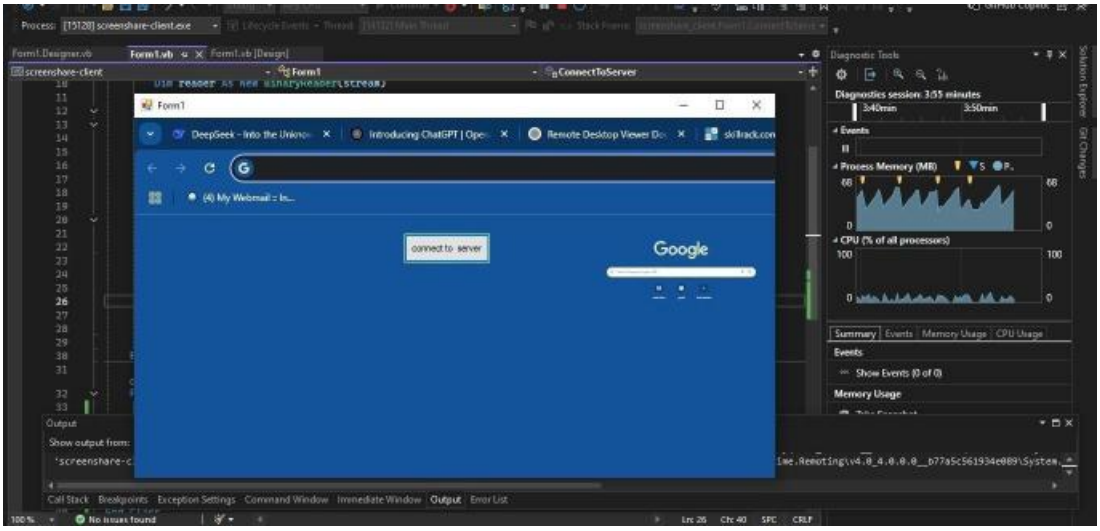

Figure 2: Real - Time Screen Sharing System

On the client side, once a connection is established with the server using the correct IP address and port number, the incoming image stream is continuously received, decoded, and displayed in the PictureBox control. The image is sent as a compressed JPEG format, ensuring efficient use of bandwidth while maintaining clarity. Overall, the system shows that a basic yet functional remote screen - viewing application can be developed effectively using Visual Basic and socket programming concepts.

7. Conclusion and Future Works

The developed screen - sharing application demonstrates the feasibility of realtime desktop streaming using Visual Basic and socket programming. By capturing the server's screen and transmitting it to a connected client over a network, the system offers a foundational remote viewing solution. The design ensures smooth and responsive updates, while also handling basic errors like disconnections gracefully. The project proves that even with minimal resources and tools, a basic remote screensharing system can be successfully built and executed.

Implementing security measures such as encrypted data transmission (using SSL/TLS) and user authentication would make the application suitable for sensitive or professional environments. Additionally, integrating remote control capabilities would allow the client not only to view but also interact with the server system, essentially turning the tool into a basic remote desktop controller. Performance can also

be improved by implementing advanced image compression techniques or reducing data redundancy. Furthermore, cross - platform compatibility could be achieved by extending the system to support mobile or web - based clients.

For future enhancements, the system can be expanded to support multiple clients simultaneously, implement secure data transmission using encryption, add authentication mechanisms for authorized access, and incorporate control functionalities so that clients can interact with the server machine remotely. Improving the compression algorithm and optimizing frame transmission can also help in reducing bandwidth usage and increasing performance over slower networks.

References

[1] Sharma, *A Beginner's Guide to Socket Programming in VB. NET*, SelfPublished Notes, 2022.
[2] D. Thomas, *Practical Guide to TCP Connections in Visual Basic*, CodeLab Publications, 2021.
[3] R. Fernandes, "Understanding Client - Server Communication using TCP in NET, " *International Journal of Computer Applications*, vol.180, no.22, pp.34–38, 2023.
[4] S. Mehta, "Network Stream Handling for Real - Time Image Transfer in VB. NET, " *Journal of Modern Software Engineering*, vol.12, no.4, 2022.
[5] Microsoft Docs, "TcpClient Class (System. Net. Sockets), " [Accessed: May 22, 2025].
[6] Kulkarni, *Developing Windows Forms Applications with. NET Framework*, TechHouse Press, 2020.
[7] T. Venkatesh, "Using Binary Streams for Data Transfer in Windows Applications, " *Tech Chronicles Magazine*, vol.6, no.3, pp.12–17, 2021.
[8] J. Miller, *Screen Capturing Techniques with. NET Libraries*, DevTech Publishing, 2019.
[9] L. Dev, "Performance Considerations in Real - Time Screen Sharing Using VB. NET, " *ACM Digital Projects*, vol.9, no.1, pp.44–50, 2022.
[10] P. Agarwal, "Image Serialization in Windows Forms Applications, " *Visual Basic Engineering Digest*, vol.7, no.2, pp.29–33, 2023.
[11] CodeMentor, "How to Build a Basic Screen Sharing App in VB. NET, " Blog post, [Accessed: May 22, 2025].12. K. Singh, *A Visual Basic Developer's Handbook*, NewAge Tech Publishers, 2021.
[12] R. Patel, "Creating a Custom Screen Capture Tool with GDI+ and VB. NET, " *Indian Journal of Software Tools*, vol.5, no.3, 2023.
[13] T. Clark, *Essentials of Real - Time Networking in. NET*, BlueDot TechBooks, 2020.
[14] S. Banerjee, "Threading and UI Responsiveness in Windows Applications, " *Software Systems Journal*, vol.14, no.1, pp.18–22, 2024.
[15] Kumar, R. (2020). *Windows Forms Programming with VB. NET: A Beginner's Guide*. New Delhi: Tech World Publications.
[16] Patel, D., & Sharma, A. (2021). "A Study on Client - Server Communication using TCP in Desktop Applications. " *International Journal of Advanced Computer Science and Applications*, 12 (7), 112–118.
[17] Joshi, P. (2022). *Real - Time Data Transmission using TCP/IP Sockets*. Mumbai: Deepak Tech Press.