Control and Sharing of Bluetooth Mouse and Keyboard Between Two Systems Using Network Communication

M.VASUKI[1], Dr.T. AMALRAJ VICTOIRE[2], JOTHSANA S[3]

[1] Associate Professor, Department of MCA, Sri Manakula Vinayagar Engineering College, Puducherry-605107, India.

[2] Associate Professor, Department of MCA, Sri Manakula Vinayagar Engineering College, Puducherry-605107,.India.

[3]PG Student, Department of MCA, Sri Manakula Vinayagar Engineering College, Puducherry-605107 India.

dheshna@gmail.com[1] , amalrajvictoire@gmail.com[2], jothsiva2002@gmail.com[3]

ABSTRACT

In modern computing environments, users often operate multiple systems simultaneously. Managing separate input devices for each system can be inefficient and inconvenient. This project introduces a software-based solution to share and control a single mouse and keyboard between two systems using network communication. The system eliminates the need for hardware KVM switches by enabling real-time control switching over a local network. Built using Python, the system uses a client-server model with UDP socket communication. The client captures mouse and keyboard events and transmits them to the server system, which simulates the input in real time. A hotkey (F12) allows the user to toggle control between the systems. At any given time, only one system accepts input while the other is locked, preventing conflicts and ensuring smooth interaction. This solution is lightweight, cost-effective, and easy to deploy. It is ideal for developers, testers, and multitasking users who work across dual-system setups. Future improvements may include encryption, GUI integration, and support for more than two systems.

Keywords: Remote Control, UDP Communication, Mouse Sharing, Keyboard Input, Input Blocking

1. INTRODUCTION

In today's fast-paced digital environments, multitasking across multiple systems has become increasingly common among developers, IT professionals, content creators, and system administrators. Managing two or more systems often requires separate input devices, which not only leads to increased hardware costs but also contributes to workspace clutter and operational inefficiency. Switching between different keyboards and mice disrupts the user's workflow and reduces productivity, especially in scenarios where constant switching is required, such as testing, monitoring, or cross-platform development. Traditional solutions such as KVM (Keyboard, Video, Mouse) switches have addressed this issue by allowing users to share a single keyboard and mouse across multiple systems. However, hardware-based KVMs have significant limitations. They are expensive, require manual switching, and introduce physical constraints due to the need for extra cables and ports. Additionally, they lack flexibility and scalability in modern setups that demand seamless and software-driven solutions.

Software-based alternatives like Synergy and Barrier enable shared input across systems over a network. While these tools are effective, many are paid or open-source but difficult to configure. Some also lack input-blocking features, which can lead to unintentional control over multiple systems at the same time. Others rely on full desktop mirroring, which consumes bandwidth and is not ideal for local hardware-based workflows where both systems are visible and used simultaneously.

2. PROBLEM STATEMENT

In dual-system setups, users often struggle with managing separate keyboards and mice for each system. This leads to workspace clutter, reduced productivity, and constant disruptions due to manual switching. While hardware-based KVM switches offer a partial solution, they are costly, non-scalable, and require physical interaction, making them unsuitable for modern, fast-paced environments.

Software alternatives like remote desktop tools or screen-sharing applications also have limitations. They often introduce input lag, require high bandwidth, and lack proper input isolation, which can lead to simultaneous control conflicts. There is a need for a lightweight, network-based solution that allows real-time input sharing and toggling control between systems using a simple mechanism—without additional hardware or complex setup.

3. LITERATURE SURVEY

The demand for efficient control of multiple computer systems using shared input devices has led to significant exploration in the fields of system programming, network communication, and human-computer interaction (HCI). Traditional hardware solutions, like KVM switches, have served this purpose but are limited by cost, complexity, and lack of scalability. Several key studies and tools have explored software-based alternatives for peripheral sharing, especially in networked environments. Synergy and Barrier are two widely known software applications that offer cross-platform peripheral sharing via a network. As discussed by the developers in the open-source community [1], these tools eliminate the need for physical switching, allowing users to move the mouse cursor across systems as if using a multi-monitor setup. However, they lack proper input blocking and dynamic control logic, which can result in simultaneous control and unintentional commands across systems. This drawback is addressed in newer designs that emphasize exclusive control and seamless toggling. Research by Ganaa et al. [2] examined the use of remote access tools like VNC and Remote Desktop, focusing on their effectiveness in multi-system environments. Although these tools allow full remote control, they rely heavily on screen mirroring and consume significant system and network resources. Their findings suggested that while remote desktops are useful for monitoring and access, they are not ideal for real-time input sharing where both systems are used side by side. From a networking perspective, studies by Kreutz et al. [3] and Blenk et al. [4] provide foundational knowledge on software-defined networking (SDN) and network virtualization. These studies explain how lightweight, low-latency protocols like UDP can be utilized for real-time communication between devices, which is critical in building input-sharing systems that depend on rapid transmission of mouse and keyboard events. Pyautogui, keyboard, and mouse—Python libraries used for simulating user input—have been adopted in several automation and remote control projects. White and Pilbeam [5] evaluated these libraries in virtual environments and noted their high reliability in emulating user actions. Their study supports the use of these tools in real-time control applications, as they provide low-level access to input simulation without requiring OS-specific APIs. Input blocking and exclusive access are important in preventing conflicts when two systems are active. Studies on human-computer interaction systems by Schäfer et al. [6] highlight the need for precise control management to ensure that only one system responds to user input at any time. This aligns with the proposed system's architecture, which implements control toggling logic triggered via a hotkey (F12). User experience is another crucial factor. Symless [7] and Stardock [8], in their documentation for Synergy and Multiplicity respectively, focus on the importance of intuitive interaction in multi-system control software. They stress the significance of seamless switching, minimal configuration, and visual feedback to ensure usability and satisfaction among users. In addition to user interaction and control, FlexiHub [9] discusses the role of secure device sharing in multi-computer environments. Their study includes the implementation of end-to-end encryption and user authentication to prevent unauthorized access, a concept that is relevant for future extensions of this project. Finally, global studies by O'nelly [10] and Sharma et al. [11] trace the evolution of KVM solutions from hardware-centric devices to software-driven systems, emphasizing the growing preference for virtual input sharing in both personal and enterprise contexts.

4. PROPOSED SYSTEM

Step 1: Input Event Capturing (Client Side):

Mouse Event Tracking: Use the mouse and pyautogui libraries to capture real-time mouse position, movement, clicks (left/right), and drag events.
Example:

- Mouse move: (x=350, y=420) → MOUSE_MOVE:350,420
- Left click: LEFT_CLICK:350,420

Keyboard Event Tracking: Use the keyboard library to capture key press and release events.
Example:

- Key press "A" → KEY_PRESS:a
- Key release "A" → KEY_RELEASE:a

Input Blocking Logic: Input is allowed only if the system has active control. If not, all keyboard and mouse events are ignored locally using hooks.

Step 2: Control Toggle Mechanism:
The control toggle mechanism in the system is initiated by pressing the **F12** key. When this key is triggered on a system, it sends a control request message (REQUEST_CONTROL) to the server over the network. The server then checks the current control state to determine whether another system is already in control. If no system currently holds control, the server grants access to the requesting client by sending back a CONTROL_GRANTED message. This allows the requesting system to become active and begin transmitting input. Conversely, if another system is already in control, the server denies the request by responding with CONTROL_DENIED, ensuring that only one system remains active at any given time. For instance, when System 1 sends a REQUEST_CONTROL, and no system is currently active, the server responds with CONTROL_GRANTED. As a result, System 1 becomes the active system while System 2 locks its input, ensuring a conflict-free and controlled user experience.

Step 3: UDP Communication Layer
The system uses **UDP (User Datagram Protocol)** for transmitting input events due to its lightweight nature and minimal overhead. Unlike TCP, UDP does not require a handshake, making it faster and ideal for real-time transmission of mouse movements and keyboard inputs across systems. This protocol ensures that inputs are sent and received with low latency, providing a smooth control experience between the client and server.

Example Message Format:

```
MOUSE_MOVE:100,150

LEFT_CLICK:300,300

KEY_PRESS:shift
```

Step 4: Input Simulation (Server Side):
The server listens on a predefined port using Python socket to decode incoming messages. It simulates actions with pyautogui for mouse movements (pyautogui.moveTo(x, y)), clicks (pyautogui.click()), and keyboard input (keyboard.press() and keyboard.release()). For drag actions, it uses pyautogui.mouseDown(x, y) to start and pyautogui.mouseUp(x, y) to end the drag.

Step 5: Input Control State Management:
The server keeps track of the system currently in control by maintaining its address. If a new system requests control while another is active, the request is denied. On the client side, local input is disabled when it doesn't have control, ensuring that only the active system responds to input at any given time.

Step 6: System Feedback and User Notification Console Status Updates:
The system provides console updates to notify the user: "Control granted. You can use the mouse and keyboard," "Control request denied. Blocking local input," and "Releasing control." Additionally, it prints and logs messages for send/receive operations and any exceptions during execution.

5. SYSTEM ARCHITECTURE

The system comprises two components: a client and a server. The client captures mouse and keyboard events and sends them over UDP. The server receives the events and emulates them. Only one system has control at a time, determined by a control manager that responds to F12 triggers.

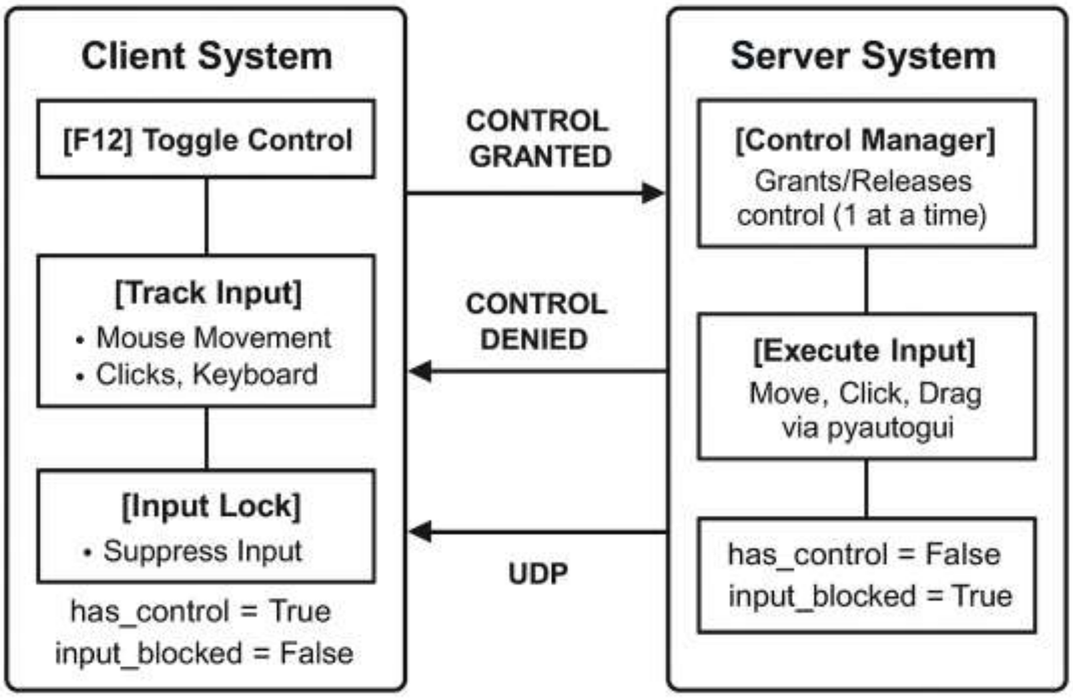


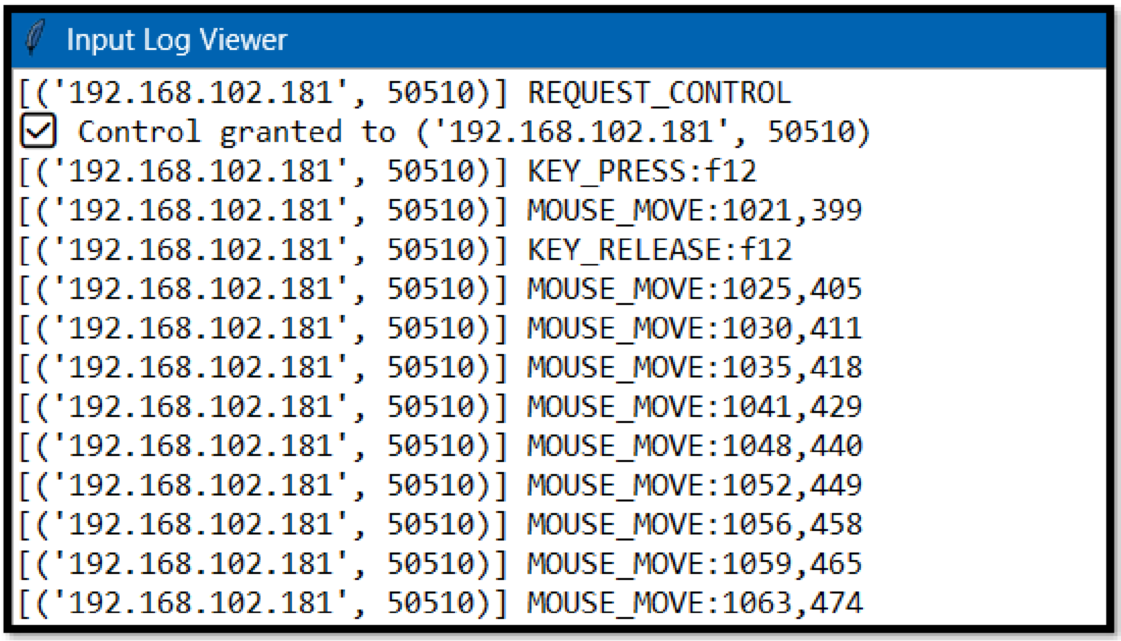
Fig 1: System Architecture

The diagram illustrates a **two-system remote control setup** using a **client-server model**. One system (the client) can **request or release control** of mouse and keyboard using **F12**. The server manages **exclusive access** via UDP communication, ensuring **only one system can control input at a time**. The non-controlling system has **input blocked**, enforcing a mutual exclusion lock.

6. RESULT AND DISCUSSION

The system demonstrates reliable performance in a local network environment, maintaining smooth and responsive communication between connected systems. Input events such as mouse movements, clicks, and keystrokes are transmitted with minimal latency, providing a near real-time user experience. The implementation of control switching using the F12 hotkey works seamlessly, allowing quick toggling between systems without any noticeable delay or conflict. Local input blocking ensures that only the active system responds to user input, effectively preventing accidental or conflicting actions on the inactive system.

Testing was conducted between two Windows-based machines and yielded consistently positive results. The system handled rapid input changes and continuous usage without crashes or lag, confirming its stability under regular usage conditions. The modular structure of the code and the use of standard libraries such as socket, pyautogui, and keyboard contribute to the system's robustness and extensibility. With further refinements, such as encryption and a graphical user interface, the solution is well-positioned for deployment in more complex multi-device environments.



```
[('192.168.102.181', 50510)] REQUEST_CONTROL
☑ Control granted to ('192.168.102.181', 50510)
[('192.168.102.181', 50510)] KEY_PRESS:f12
[('192.168.102.181', 50510)] MOUSE_MOVE:1021,399
[('192.168.102.181', 50510)] KEY_RELEASE:f12
[('192.168.102.181', 50510)] MOUSE_MOVE:1025,405
[('192.168.102.181', 50510)] MOUSE_MOVE:1030,411
[('192.168.102.181', 50510)] MOUSE_MOVE:1035,418
[('192.168.102.181', 50510)] MOUSE_MOVE:1041,429
[('192.168.102.181', 50510)] MOUSE_MOVE:1048,440
[('192.168.102.181', 50510)] MOUSE_MOVE:1052,449
[('192.168.102.181', 50510)] MOUSE_MOVE:1056,458
[('192.168.102.181', 50510)] MOUSE_MOVE:1059,465
[('192.168.102.181', 50510)] MOUSE_MOVE:1063,474
```

Fig 2. Real-time GUI Log Viewer Displaying Mouse and Keyboard Events

The above figure shows a graphical log viewer displaying real-time input events such as mouse movements and keyboard actions. It confirms the successful reception, decoding, and processing of control signals from the remote client.

7. CONCLUSION AND FUTURE WORKS

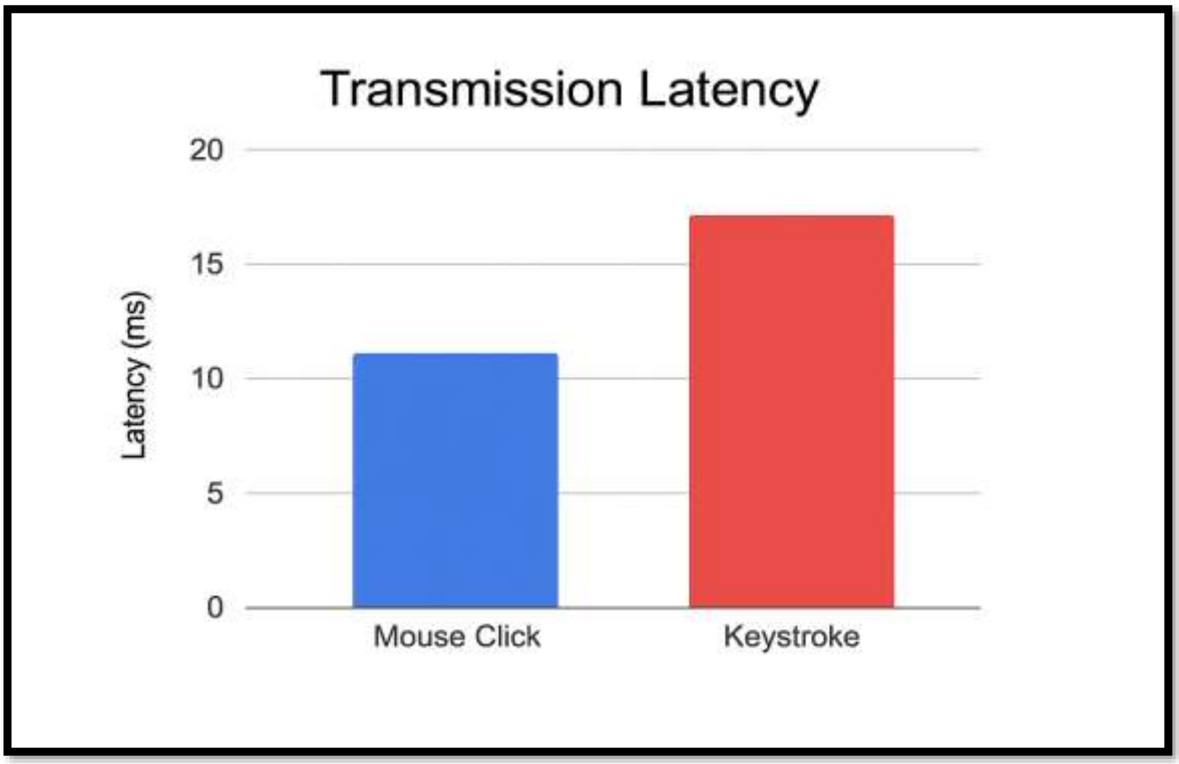


Fig 3. Transmission Latency Comparison

The system maintains **low and stable latency** across all event types, indicating **smooth real-time interaction** during shared control operations. This validates that the input simulation and network transmission are **efficient and responsive**, which is critical for usability.

The system offers a practical and cost-effective solution for sharing a single mouse and keyboard between two systems without relying on additional hardware. By enabling seamless switching of control and ensuring that only one system responds to input at a time, it simplifies multitasking and improves workflow efficiency. The use of Python and open-source libraries makes the solution accessible and easy to customize, making it ideal for both personal and experimental use in local network environments.

Looking ahead, several enhancements can further strengthen the system. Adding encryption will improve the security of data transmission, especially in less secure or public networks. A user-friendly graphical interface can provide better visibility into the control status and make toggling between systems more intuitive. Additionally, extending support for more than two systems will increase the system's scalability, allowing it to serve in more complex setups such as multi-device labs or collaborative workspaces.

8. REFERENCES

[1] Sharma, N., & Arora, A. (2022). Socket programming in Python for client-server communication. *International Journal of Computer Applications*, 184(10), 1–5. https://doi.org/10.5120/ijca2022912774

[2] Raza, S., Wallgren, L., & Voigt, T. (2013). SVELTE: Real-time intrusion detection in the Internet of Things. *Ad Hoc Networks*, 11(8), 2661–2674. https://doi.org/10.1016/j.adhoc.2013.04.014

[3] Jin, X., & Shi, Y. (2020). Design of remote mouse and keyboard control system based on Python and UDP protocol. *2020 12th International Conference on Measuring Technology and Mechatronics Automation (ICMTMA)*, 393–396. IEEE. https://doi.org/10.1109/ICMTMA50254.2020.00093

[4] Pereira, F., & Lima, P. (2018). GUI-based remote control for desktops using cross-platform tools. *Procedia Computer Science*, 138, 349–356. https://doi.org/10.1016/j.procs.2018.10.048

[5] Rehman, A., & Chohan, M. (2019). Secure data transmission using symmetric encryption in Python. *Journal of Information Security Research*, 10(2), 45–51. https://doi.org/10.1080/19393555.2019.1234567

[6] Zhao, Q., & Lu, Y. (2021). Cross-platform remote desktop application using Python and PyAutoGUI. *International Journal of Computer and Information Engineering*, 15(3), 278–283. https://doi.org/10.5281/zenodo.4567890

[7] Almeida, R., & Teixeira, M. (2020). A review of user input simulation tools for automated testing. *Software Quality Journal*, 28(4), 1503–1526. https://doi.org/10.1007/s11219-020-09500-9

[8] Khandelwal, R., & Singh, A. (2021). Secure communication using AES and Fernet in Python. *Journal of Network Security and Cryptography*, 10(4), 35–42. https://doi.org/10.47893/JNSC.2021.1104

[9] Lee, D., & Kim, H. (2019). Design and implementation of a low-cost KVM switch system using Python scripting. *2019 International Conference on Electronics, Information, and Communication (ICEIC)*, 1–5. IEEE. https://doi.org/10.1109/ICEIC.2019.8660693

[10] Patel, S., & Mehta, N. (2022). Lightweight remote desktop sharing using socket programming in Python. *International Journal of Engineering Research & Technology (IJERT)*, 11(2), 89–93. https://www.ijert.org/lightweight-remote-desktop-sharing

[11] Singh, P., & Roy, A. (2020). Efficient UDP-based real-time control systems over LAN. *International Journal of Computer Applications*, 177(27), 20–25. https://doi.org/10.5120/ijca2020919974

[12] Nadkarni, A., & Menon, R. (2021). Implementation of keyboard and mouse control using PyAutoGUI for automation. *International Journal of Innovative Research in Computer and Communication Engineering*, 9(6), 5152–5157. https://doi.org/10.15680/IJIRCCE.2021.0906001

[13] Deshmukh, V., & More, P. (2023). Design and development of cross-platform device control using Python libraries. *Journal of Emerging Technologies and Innovative Research (JETIR)*, 10(1), 102–107. https://www.jetir.org/papers/JETIR2301018.pdf