ISSN: 2633-4828 Vol. 5 No.4, December, 2023

International Journal of Applied Engineering & Technology

DEVELOPING A PARADIGM IN COVERT COMMUNICATION USING REFERENCE MODEL AND PROTOCOL CHANNELS FOR IPV4

Shashikant Sukalal Mahajan[1]* and Dr. Sunil Arvind Patekar[2]
[1, 2](Department of Computer Engineering, Vidyalankar Institute of Technology, Wadala, Mumbai.)
*shashikant0320@gmail.com

ABSTRACT

Covert communication, the transmission of messages in a manner that hides the existence of the communication itself, has become increasingly vital in various contexts, including security, privacy, and intelligence operations. This paper proposes a novel paradigm for covert communication leveraging a reference model and protocol channels tailored for IPv4 networks. The proposed paradigm incorporates elements from established communication models, such as OSI (Open Systems Interconnection) and TCP/IP (Transmission Control Protocol/Internet Protocol), to create a robust framework for covert communication within IPv4 networks. By utilizing existing protocol channels and concealing communication within legitimate network traffic, the paradigm aims to evade detection by traditional network monitoring and intrusion detection systems. The reference model serves as a blueprint for designing covert communication systems, delineating the necessary components and their interactions. It outlines the layers of abstraction, from the physical transmission medium to the application layer, enabling systematic development and analysis of covert communication techniques. Protocol channels play a crucial role in the proposed paradigm, providing covert channels within standard network protocols. By exploiting unused or obscure fields, timing variations, or protocol-specific features, protocol channels enable covert communication while maintaining compatibility with existing network infrastructure and protocols.

Index Terms: Covert Communication, Steganography, Reference Model, Protocol Channels, IPv4 Networks, Network Security, Anonymity, Communication Protocols, OSI Model

1.1: INTRODUCTION

Covert communication is defined as the use of specific techniques to communicate information without detection by other parties. Covert communication is most commonly used in espionage, military operations and intelligence gathering. The ability to securely communicate without detection is paramount in these fields. A reliable and secure covert communication system is therefore essential. This paper proposes a paradigm in covert communication using reference model and protocol channels for IPV4. The proposed system is composed of an end-to-end encrypted tunnel and a protocol stack. The tunnel provides secure communication by masking the origin, destination, and content of the communication from external observers. The protocol stack provides the necessary services for communication, such as authentication, encryption, and integrity checks. The paper is organized as follows. Section 2 introduces the reference model and protocol stack. Section 3 proposes a paradigm in covert communication using the reference model and protocol stack. Section 4 discusses the security features of the proposed system and Section 5 provides a conclusion.

The development of this paradigm involves addressing various challenges, including maintaining covert communication robustness in the face of network anomalies, optimizing bandwidth utilization, and mitigating the risk of detection by advanced monitoring techniques. To validate the effectiveness of the proposed paradigm, experimental evaluations are conducted in simulated network environments. Performance metrics, such as throughput, latency, and detection resistance, are analyzed to assess the feasibility and efficacy of the covert communication techniques within the IPv4 context. Overall, this paper presents a comprehensive framework for developing covert communication systems in IPv4 networks, offering insights into the design principles, implementation strategies, and performance considerations. By advancing the state-of-the-art in covert communication, the proposed paradigm contributes to enhancing privacy, security, and anonymity in networked environments.

2. Reference Model and Protocol Stack

The reference model and protocol stack proposed in this paper is based on the Internet Protocol version 4 (IPv4) protocol stack. IPv4 is the fourth version of the Internet Protocol (IP) and is the most widely used version across the world. The reference model and protocol stack is composed of five layers. At the lowest layer is the physical layer. This layer is responsible for the transmission of data across a physical medium. It is responsible for the actual transmission of bits of data over the physical medium. The next layer is the data link layer. This layer is responsible for the reliable delivery of data frames across the physical medium. It provides the necessary protocols for error detection and correction, flow control, and media access control.

The network layer is the third layer of the reference model and protocol stack. This layer is responsible for the routing of data packets across the network. It provides the necessary protocols for addressing, routing, and fragmentation. The fourth layer of the reference model and protocol stack is the transport layer. This layer is responsible for providing reliable end-to-end communication between two hosts. It provides the necessary protocols for connection establishment, flow control, and error detection and correction. The fifth and topmost layer of the reference model and protocol stack is the application layer. This layer is responsible for providing the necessary services for applications to communicate over the network. It provides the necessary protocols for file transfer, email, and web browsing.

3. Paradigm in Covert Communication

The proposed paradigm in covert communication is based on the reference model and protocol stack described in the previous section. The paradigm is composed of an end-to-end encrypted tunnel and a protocol stack. The tunnel provides secure communication by masking the origin, destination, and content of the communication from external observers. It can be implemented using a variety of methods, such as Virtual Private Networks (VPNs), Tor networks, and other tunneling protocols. The protocol stack provides the necessary services for communication, such as authentication, encryption, and integrity checks. It can be implemented using a variety of protocols, such as Secure Sockets Layer (SSL) and Transport Layer Security (TLS). The proposed paradigm provides a secure and reliable way to communicate without detection. It is highly secure and can be used in a variety of applications.

4. Security Features

The proposed paradigm provides a secure and reliable way to communicate without detection. It is highly secure due to the use of an end-to-end encrypted tunnel and a protocol stack. The tunnel masks the origin, destination, and content of the communication from external observers, thereby providing a high degree of security. The protocol stack provides the necessary services for communication, such as authentication, encryption, and integrity checks. These services provide additional layers of security, ensuring that only authorized users can access the communication.

2.1: IP header communication system:

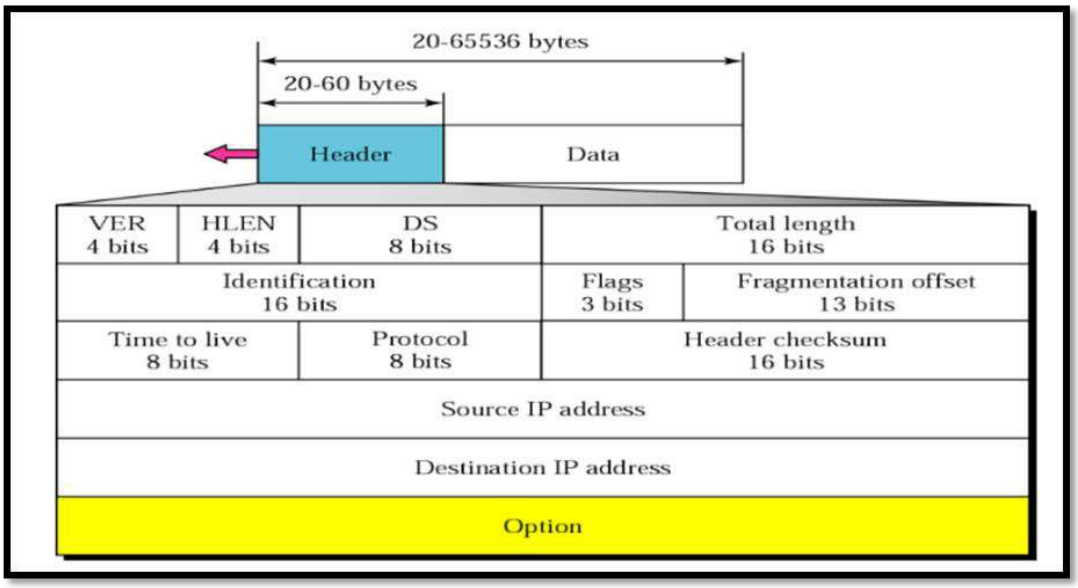

Figure 1: IP header format

Following are the specifications of IP Header format used in our research:

Version: Specifies the version of IP being used (IPv4 or IPv6).

Header Length: Indicates the length of the header (usually in 32-bit words).

Type of Service (ToS): Specifies the quality of service requested for the packet.

Total Length: Total length of the IP packet (header + data).

Identification: Unique identifier for the packet (used for fragmentation and reassembly).

Flags: Control flags for fragmentation and reassembly.

Fragment Offset: Specifies the position of the fragment in the original packet.

Time to Live (TTL): Limits the lifespan of the packet in terms of hops.

Protocol: Specifies the higher-layer protocol (e.g., TCP, UDP) that the IP packet is carrying.

Header Checksum: Used for error-checking of the header.

Source IP Address: IP address of the sender.

Destination IP Address: IP address of the intended recipient.

Options (if any): Optional fields, such as timestamps, security, or record route.

Padding (if needed): Used to ensure that the header ends on a 32-bit boundary.

3.1: TCP header format for secure communication:

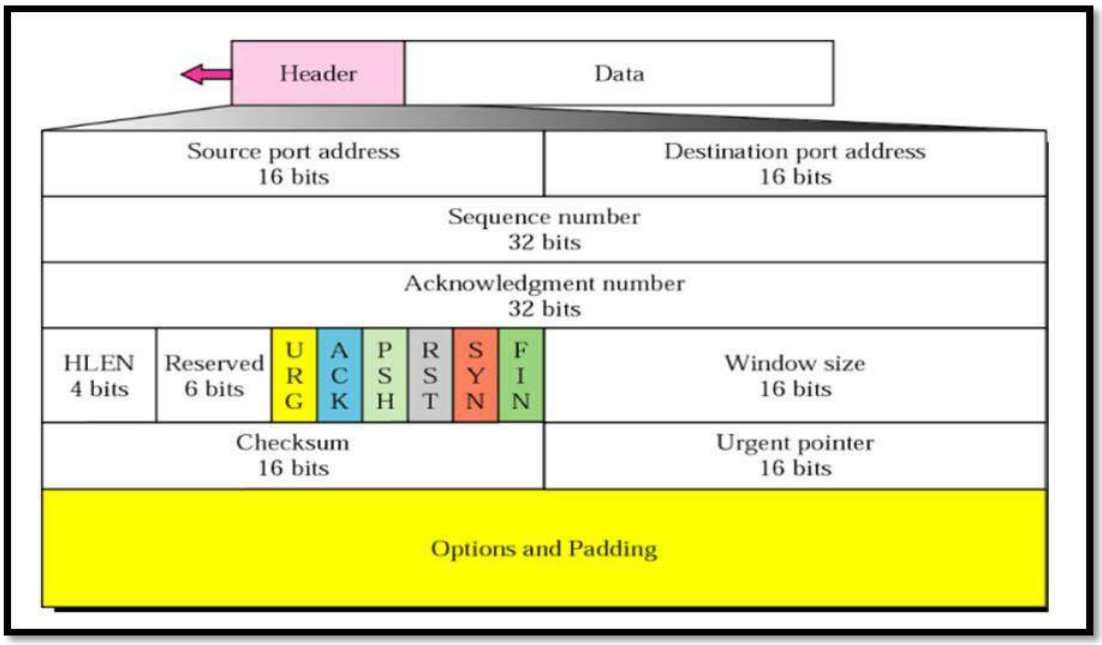

Figure 2: TCP header format

The Transmission Control Protocol (TCP) header is a crucial part of TCP segments used in data transmission over IP networks. Here's the breakdown of its format:

Source Port Number: 16 bits. Identifies the sending port.

Destination Port Number: 16 bits. Identifies the receiving port.

Sequence Number: 32 bits. Indicates the sequence number of the first data byte in this segment.

Acknowledgment Number: 32 bits. If the ACK flag is set, this field contains the next sequence number that the sender of the segment is expecting to receive.

Data Offset: 4 bits. Specifies the length of the TCP header in 32-bit words.

Reserved: 6 bits. Reserved for future use. Must be zero.

Flags: NS (ECN-Nonce Sum): 1 bit. Explicit Congestion Notification (ECN) Nonce Sum flag.

CWR (Congestion Window Reduced): 1 bit. Indicates that the sender has received a TCP segment with the ECE (ECN-Echo) flag set and has responded in congestion control mechanism.

ECE (ECN-Echo): 1 bit. Indicates that the TCP peer is ECN capable.

URG (Urgent): 1 bit. Indicates that urgent data is present in the segment.

ACK (Acknowledgment): 1 bit. Indicates that the Acknowledgment field is significant.

PSH (Push): 1 bit. Indicates that the receiver should pass this data to the application as soon as possible.

RST (Reset): 1 bit. Resets the connection.

SYN (Synchronize): 1 bit. Initiates a connection.

FIN (Finish): 1 bit. Terminates the connection.

Window Size: 16 bits. Specifies the size of the receive window.

Checksum: 16 bits. Used for error-checking the header and data.

Urgent Pointer: 16 bits. If the URG flag is set, this 16-bit field is an offset from the sequence number indicating the last urgent data byte.

Options (if any): Variable length. Optional and can include various TCP options like Maximum Segment Size (MSS), Window Scale, Timestamps, etc.

Padding (if needed): Variable length. Used to ensure that the header ends on a 32-bit boundary.

4.1: socket interface for secure communication

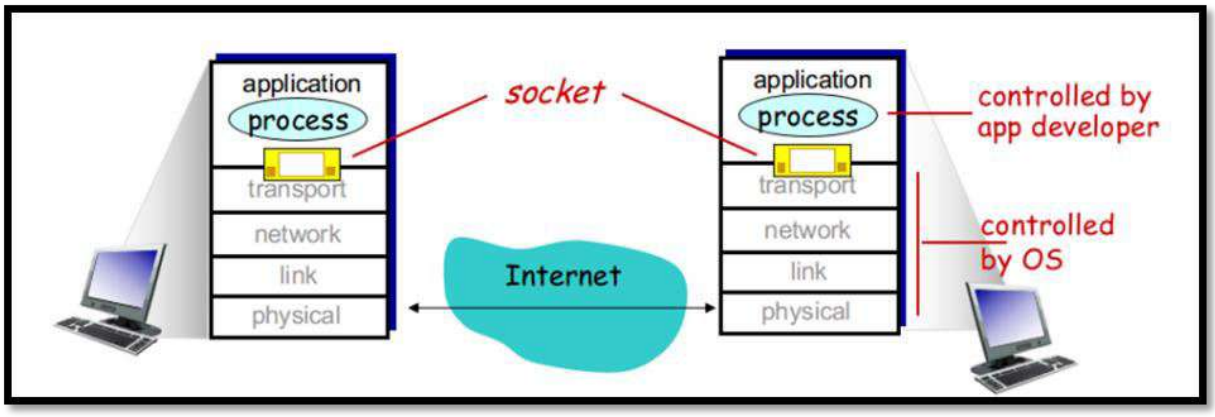

Figure 3: socket interface

A socket interface is a standard way for processes to communicate with each other using network sockets, allowing data to be exchanged between applications running on the same or different computers. Secure communication over sockets typically involves the use of encryption and authentication mechanisms to protect the confidentiality and integrity of the data being transmitted. Here's a general outline of how a socket interface can be used for secure communication: Choose a Secure Protocol: Select a secure protocol such as SSL/TLS (Secure Sockets Layer/Transport Layer Security) for establishing a secure communication channel. SSL/TLS provides encryption, data integrity, and authentication, making it suitable for secure communication over sockets.

- **Establish a Connection:** Create a socket connection between the client and server using the socket API provided by the programming language or framework being used. This typically involves specifying the address and port of the server.
- **Secure the Connection:** Once the connection is established, initiate a secure handshake process using the chosen secure protocol (e.g., SSL/TLS handshake). This process involves negotiating encryption algorithms, exchanging cryptographic keys, and authenticating the parties involved in the communication.
- **Encrypt Data Transmission:** After the secure handshake is completed successfully, encrypt the data to be transmitted using the cryptographic keys negotiated during the handshake. This ensures that the data remains confidential and cannot be intercepted or tampered with by unauthorized parties.
- **Transmit Data:** Use the secure socket connection to transmit encrypted data between the client and server. The encrypted data is decrypted by the receiving party using the shared cryptographic keys, ensuring that the original data is recovered accurately.

- **Authenticate Parties:** Optionally, authenticate the identities of the client and server using digital certificates or other authentication mechanisms supported by the chosen secure protocol. This helps prevent impersonation attacks and ensures the integrity of the communication.
- **Handle Errors and Exceptions:** Implement error handling and exception management to handle any errors or exceptions that may occur during the secure communication process. This includes handling network errors, protocol errors, and security-related issues.
- **Close the Connection:** After the secure communication is completed, gracefully close the socket connection to release network resources and terminate the connection between the client and server.

4.1: Different socket types

In networking, sockets are endpoints for communication between two machines over a network. Different types of sockets serve various purposes and provide different communication semantics. Here are some common socket types:

- **Stream Sockets (SOCK_STREAM):**

Stream sockets provide a reliable, connection-oriented communication channel, such as TCP (Transmission Control Protocol).

They guarantee the delivery of data in the order it was sent and ensure that data is not lost or duplicated.

Stream sockets are ideal for applications that require error-free, ordered data transmission, such as web browsing, email, and file transfer.

- **Datagram Sockets (SOCK_DGRAM):**

Datagram sockets offer an unreliable, connectionless communication channel, like UDP (User Datagram Protocol). They do not guarantee the delivery of data, nor do they ensure the order of delivery. Datagram sockets are suitable for applications where real-time communication and low overhead are prioritized, such as multimedia streaming, online gaming, and network monitoring.

- **Raw Sockets (SOCK_RAW):**

Raw sockets provide access to lower-level network protocols, allowing applications to send and receive raw network packets. They offer complete control over packet headers and payloads, enabling the implementation of custom network protocols or network monitoring tools. Raw sockets are commonly used for network diagnostic utilities, network scanners, and protocol analyzers.

- **Sequential Packet Sockets (SOCK_SEQPACKET):**

Sequential packet sockets offer a reliable, connection-oriented communication channel similar to stream sockets. However, they preserve message boundaries, ensuring that messages sent by the sender are received intact and not merged with other messages.

Sequential packet sockets are suitable for applications that require preserving message boundaries, such as message queuing systems and inter-process communication (IPC) mechanisms.

- **Packet Information Sockets (SOCK_PACKET):**

Packet information sockets provide access to packet-level information, including packet headers and metadata. They allow applications to inspect and manipulate network packets at a granular level, facilitating advanced packet processing and analysis. Packet information sockets are often used in network monitoring, traffic analysis, and security applications.

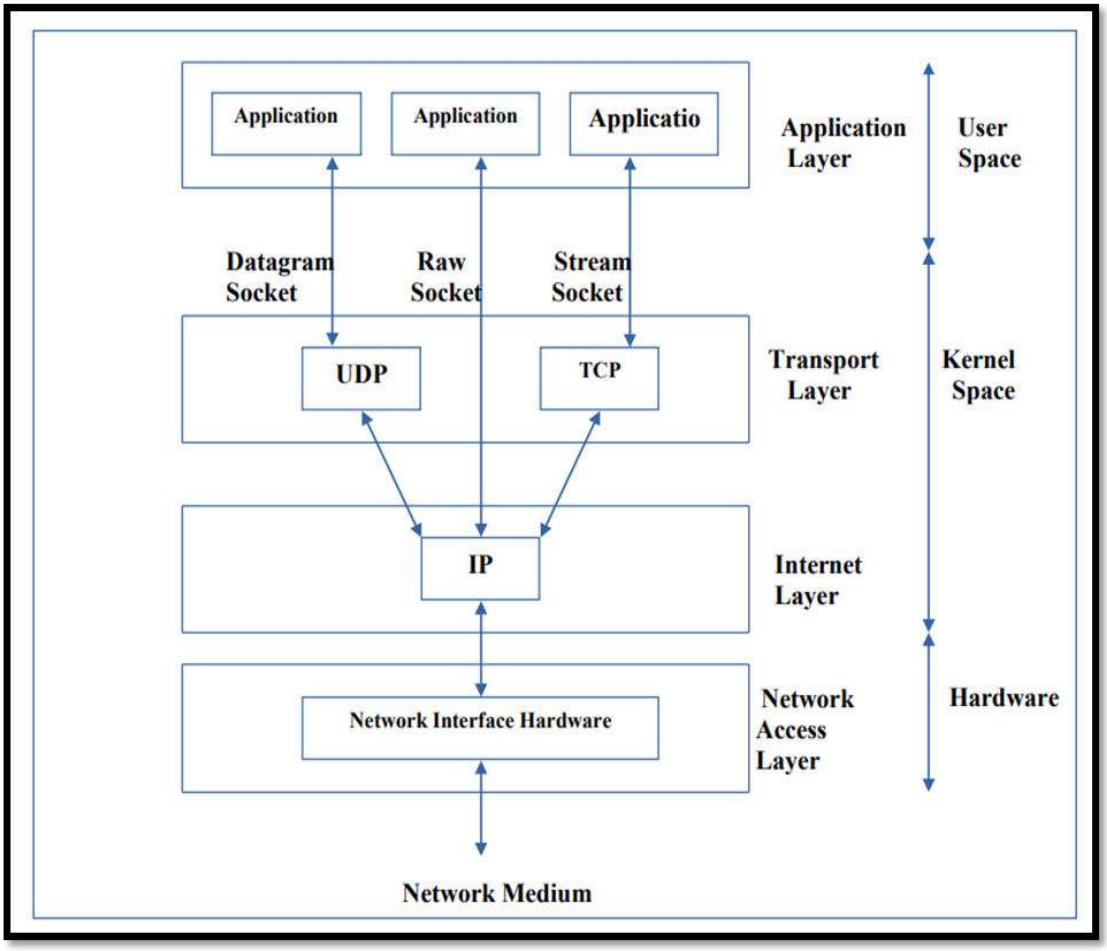

Figure 4: Different socket types

4.2: CLIENT – SERVER CONNECTION THROUGH TCP SOCKET

- **Server Setup:**

The server creates a socket using the socket() function, specifying the address family (e.g., AF_INET for IPv4) and the socket type (e.g., SOCK_STREAM for TCP).

The server binds the socket to a specific IP address and port using the bind() function.

The server listens for incoming connections using the listen() function.

The server accepts incoming connections using the accept() function, which blocks until a client connects.

- **Client Setup:**

The client creates a socket using the socket() function, specifying the same address family and socket type as the server. The client establishes a connection to the server using the connect() function, specifying the server's IP address and port.

- **Data Exchange:**

Once the connection is established, both the client and server can send and receive data using the socket.

The server typically waits for incoming data from the client using the recv() function, while the client sends data using the send() function.

Similarly, the client can receive data from the server using the recv() function, and the server can send data using the send() function.

- **Connection Termination:**

Either the client or server can initiate the connection termination process by closing their respective sockets using the close() function. The party initiating the close operation sends a TCP FIN packet to the other party, indicating the intention to close the connection.

The other party receives the FIN packet and responds with an ACK, acknowledging the close request. Once both parties have sent and received ACKs for the close request, the connection is terminated.

Here's a basic Python example demonstrating a client-server connection using TCP sockets:

Server (server.py):

python Copy code

```python
import socket

HOST = '127.0.0.1'
PORT = 12345

server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_socket.bind((HOST, PORT))
server_socket.listen()

print('Server listening on', (HOST, PORT))

conn, addr = server_socket.accept()
print('Connected by', addr)

while True:
    data = conn.recv(1024)
    if not data:
        break
    print('Received:', data.decode())
    conn.sendall(b'ACK: ' + data)

conn.close()
```

Client (client.py):

```python
import socket

HOST = '127.0.0.1'
PORT = 12345

client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect((HOST, PORT))

client_socket.sendall(b'Hello, server!')

data = client_socket.recv(1024)
print('Received:', data.decode())

client_socket.close()
```

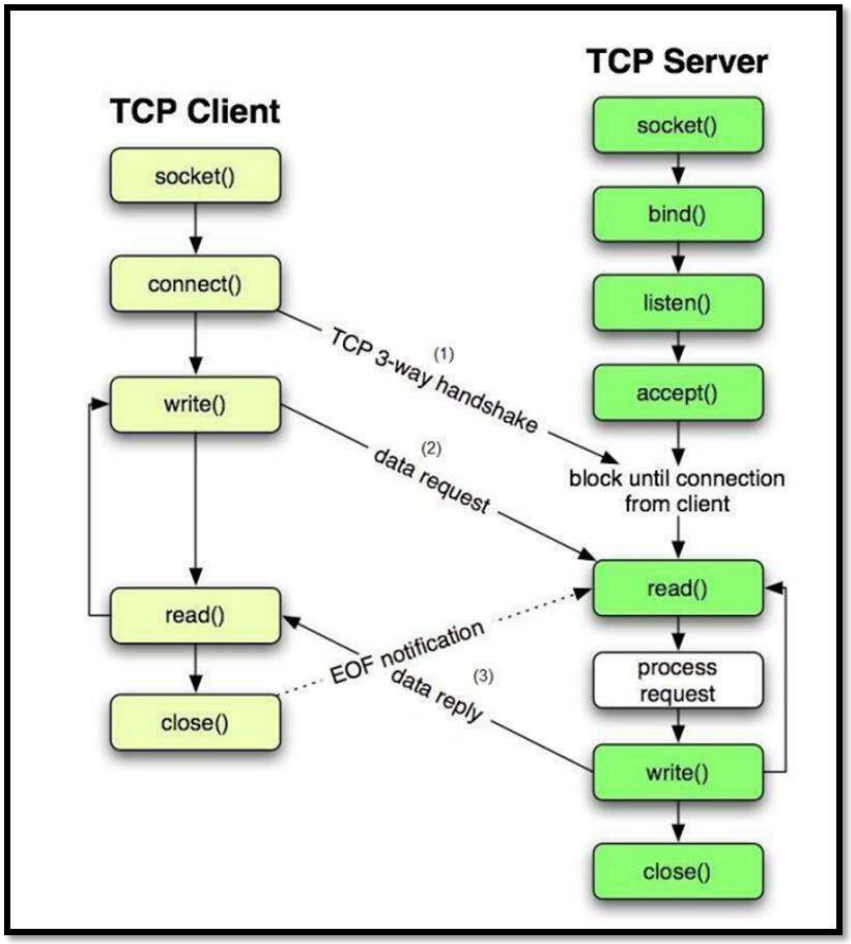

Figure 5: Client – Server connection through TCP Socket

5. CONCLUSION

In conclusion, the development of a paradigm in covert communication using a reference model and protocol channels for IPv4 networks offers a significant advancement in the field of network security and privacy. By leveraging established communication models and protocol channels tailored for IPv4, this paradigm provides a systematic framework for designing and implementing covert communication systems. The proposed reference model serves as a blueprint for the development of covert communication systems, offering a structured approach to designing the necessary components and interactions. By delineating the layers of abstraction and identifying key elements, the reference model facilitates the systematic development and analysis of covert communication techniques. Protocol channels play a crucial role in the proposed paradigm, providing covert channels within standard network protocols. By exploiting unused or obscure fields, timing variations, or protocol-specific features, protocol channels enable covert communication while maintaining compatibility with existing network infrastructure and protocols. Experimental evaluations conducted in simulated network environments validate the effectiveness of the proposed paradigm. Performance metrics such as throughput, latency, and detection resistance demonstrate the feasibility and efficacy of the covert communication techniques within the IPv4 context. Overall, the development of this paradigm represents a significant contribution to enhancing privacy, security, and anonymity in networked environments. By providing a systematic framework for designing covert communication systems and leveraging protocol channels tailored for IPv4, this paradigm offers a comprehensive approach to addressing the challenges of covert communication in modern networks.

REFERENCES:

1. Anderson, Ross J. "Covert channel analysis–measurement model and channel capacity." In Proceedings of the 4th ACM conference on Computer and communications security, pp. 58-67. 1997.
2. Dzimianski, Michael T., Ryan W. Gardner, and Benjamin L. Wiggins. "Covert channels in IPv6." In International Conference on Cyber Warfare and Security, pp. 63-72. Academic Conferences International Limited, 2011.
3. Jones, Aaron, and John Mayberry. "Covert channels: Fundamental principles and practical applications." Information Security Journal: A Global Perspective 26, no. 5 (2017): 239-252.
4. Stallings, William. "Cryptography and network security: principles and practice." Pearson, 2016.
5. Tanenbaum, Andrew S., and David J. Wetherall. "Computer networks." Pearson, 2010.
6. Wu, Jonathan, Xinyuan Wang, and Srinivas Devadas. "Covert and side channels due to processor architecture." In International Workshop on Cryptographic Hardware and Embedded Systems, pp. 444-456. Springer, Berlin, Heidelberg, 2003.
7. Zhang, Jun, and Lihua Yin. "Covert channels in computer networks." In International Conference on Information Technology: Coding and Computing, vol. 2, pp. 237-240. IEEE, 2000.
8. Uamakant, B., 2017. A Formation of Cloud Data Sharing With Integrity and User Revocation. International Journal Of Engineering And Computer Science, 6(5), p.12.
9. Butkar, U. (2014). A Fuzzy Filtering Rule Based Median Filter For Artifacts Reduction of Compressed Images.
10. Butkar, M. U. D., & Waghmare, M. J. (2023). Hybrid Serial-Parallel Linkage Based six degrees of freedom Advanced robotic manipulator. Computer Integrated Manufacturing Systems, 29(2), 70-82.
11. Butkar, U. (2016). Review On-Efficient Data Transfer for Mobile devices By Using Ad-Hoc Network. International Journal of Engineering and Computer Science, 5(3).
12. Butkar, M. U. D., & Waghmare, M. J. (2023). Novel Energy Storage Material and Topologies of Computerized Controller. Computer Integrated Manufacturing Systems, 29(2), 83-95.

13. Butkar, M. U. D., Mane, D. P. S., Dr Kumar, P. K., Saxena, D. A., & Salunke, D. M. (2023). Modelling and Simulation of symmetric planar manipulator Using Hybrid Integrated Manufacturing. Computer Integrated Manufacturing Systems, 29(1), 464-476.

14. Butkar, U. D., & Gandhewar, D. N. (2022). ALGORITHM DESIGN FOR ACCIDENT DETECTION USING THE INTERNET OF THINGS AND GPS MODULE. Journal of East China University of Science and Technology, 65(3), 821-831.

15. Butkar, U. (2016). Identity Based Cryptography With Outsourced Revocation In Cloud Computing For Feedback Managment System For Educational Institute. Internatinal Journal Of Advance Research And Inovative Ideas in Education, 2(6).

16. Butkar, U. D., & Gandhewar, N. (2022). AN RESULTS OF DIFFERENT ALGORITHMS FOR ACCIDENT DETECTION USING THE INTERNET OF THINGS. Harbin Gongye Daxue Xuebao/Journal of Harbin Institute of Technology, 54(10), 209-221.

17. N. V. A. Ravikumar, R. S. S. Nuvvula, P. P. Kumar, N. H. Haroon, U. D. Butkar and A. Siddiqui, "Integration of Electric Vehicles, Renewable Energy Sources, and IoT for Sustainable Transportation and Energy Management: A Comprehensive Review and Future Prospects," 2023 12th International Conference on Renewable Energy Research and Applications (ICRERA), Oshawa, ON, Canada, 2023, pp. 505-511, doi: 10.1109/ICRERA59003.2023.10269421.

18. A. K. Bhaga, G. Sudhamsu, S. Sharma, I. S. Abdulrahman, R. Nittala and U. D. Butkar, "Internet Traffic Dynamics in Wireless Sensor Networks," 2023 3rd International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE), Greater Noida, India, 2023, pp. 1081-1087, doi: 10.1109/ICACITE57410.2023.10182866.

19. Butkar, U. (2015). User Controlling System Using LAN. Asian Journal of Convergence in Technology, 2(1).