**🧠 Project Report: Real-Time Communication and Packet Analysis using Raw Sockets in C**

**📌 Table of Contents**

1. Introduction
2. Objectives
3. System Architecture
4. Technologies Used
5. Theoretical Background
6. Implementation
   * Raw Socket Creation
   * Packet Construction
   * Client-Server Communication
   * Packet Sniffing
   * Jitter Calculation
   * Packet Loss Detection
7. Wireshark Analysis
8. Testing & Evaluation
9. Conclusion
10. Future Enhancements
11. References

**🔰 1. Introduction**

This project focuses on creating a real-time communication system using **Raw Sockets** in the **C programming language**, supporting client-server chat and live network packet analysis. It also implements metrics like **jitter** and **packet loss**. **Wireshark**, a popular packet analysis tool, is used to verify and analyze the traffic.

**🎯 2. Objectives**

* Enable communication between two systems via raw sockets.
* Understand and manipulate packets at the IP and transport layer.
* Capture packets using a sniffer and analyze their content.
* Calculate real-time **jitter** and **packet loss**.
* Use **Wireshark** to visualize packet structures and validate correctness.

**🏗️ 3. System Architecture**

plaintext

CopyEdit

+----------------+ +----------------+

| Client | <---> | Server |

+----------------+ +----------------+

| |

| |

v v

+----------------------------------------+

| Raw Socket Communication |

+----------------------------------------+

| |

v v

+----------------+ +----------------+

| Packet Sniffer | | Wireshark Tool |

+----------------+ +----------------+

**🛠️ 4. Technologies Used**

| **Component** | **Description** |
| --- | --- |
| Language | C |
| Socket API | Raw Sockets |
| Analysis Tool | Wireshark |
| Operating System | Linux (root access required) |
| Protocols | IP, ICMP, TCP, UDP |

**📚 5. Theoretical Background**

**🔹 Raw Socket**

A **raw socket** allows direct sending and receiving of IP packets without any protocol-specific transport layer formatting (like TCP/UDP). It bypasses the kernel's standard protocol stack for full control over packet headers.

c

CopyEdit

int sockfd = socket(AF\_INET, SOCK\_RAW, IPPROTO\_RAW);

* AF\_INET: IPv4 address family
* SOCK\_RAW: Raw socket type
* IPPROTO\_RAW: Allow sending custom protocol data

🧠 Requires root access.

**🔹 IP Header**

Part of the OSI Layer 3; contains source IP, destination IP, version, TTL, etc.

**🔹 Transport Layer**

You can use either:

* **UDP** (connectionless, low overhead)
* **TCP** (reliable, connection-based)

For chat, UDP is generally preferred in raw sockets to avoid overhead.

**🔹 Jitter**

**Jitter** refers to the variation in time between packets arriving, caused by network congestion, route changes, or timing drift.

**Jitter = |arrival\_time(n) - arrival\_time(n-1) - expected\_interval|**

**🔹 Packet Loss**

When one or more packets fail to reach the destination.

**Packet Loss Rate = (Lost Packets / Sent Packets) × 100%**

**🔧 6. Implementation**

**6.1 Client-Server Communication via Raw Sockets**

**🔸 Server Code Steps:**

1. Create a raw socket.
2. Bind to the IP address and port.
3. Continuously receive data.
4. Capture timestamps for jitter.

**🔸 Client Code Steps:**

1. Construct a raw packet.
2. Manually fill IP and UDP headers.
3. Send using sendto().
4. Maintain packet sequence for loss calculation.

**6.2 Packet Construction**

**IP Header (20 bytes):**

c

CopyEdit

struct iphdr {

unsigned int ihl:4;

unsigned int version:4;

unsigned char tos;

unsigned short tot\_len;

unsigned short id;

unsigned short frag\_off;

unsigned char ttl;

unsigned char protocol;

unsigned short check;

unsigned int saddr;

unsigned int daddr;

};

**UDP Header (8 bytes):**

c

CopyEdit

struct udphdr {

unsigned short source;

unsigned short dest;

unsigned short len;

unsigned short check;

};

**6.3 Packet Sniffing**

Use raw sockets with IPPROTO\_TCP or IPPROTO\_UDP to receive packets.

c

CopyEdit

int sockfd = socket(AF\_INET, SOCK\_RAW, IPPROTO\_UDP);

recvfrom(sockfd, buffer, BUFFSIZE, ...);

Sniffer parses IP headers and logs packet time, size, and content.

**6.4 Jitter Calculation**

* Record timestamps (use gettimeofday()).
* Calculate delay differences between subsequent packets.
* Store and plot jitter values for analysis.

**6.5 Packet Loss Detection**

* Add sequence numbers to payload.
* Track which packets were received and which were not.
* Infer missing packets by detecting gaps.

**🔍 7. Wireshark Analysis**

Wireshark is used to capture and visually analyze:

* Source & Destination IPs
* TTL, Protocol, Checksum
* UDP header (Port numbers, Length, Checksum)
* Payload content
* Arrival time differences (for jitter)
* Sequence tracking (for loss)

Use display filters like udp.port == 9090 or ip.src == 192.168.1.2

**🧪 8. Testing & Evaluation**

| **Test Case** | **Expected Outcome** | **Result** |
| --- | --- | --- |
| Send Message | Message appears on the server | ✅ |
| Packet Loss | Missing sequence detected | ✅ |
| Jitter Test | Variation in arrival times shown | ✅ |
| Wireshark Capture | Packet headers and payloads visible | ✅ |

**✅ 9. Conclusion**

The project successfully demonstrates:

* Low-level networking with raw sockets
* Full control over IP/UDP headers
* Custom implementation of communication protocols
* Real-time packet analysis using sniffing
* Accurate calculation of jitter and packet loss
* Effective verification with Wireshark

**🔮 10. Future Enhancements**

* Add **encryption** to payload for secure communication.
* Extend to support **TCP with three-way handshake**.
* Integrate GUI-based **real-time charts** for jitter/loss.
* Use **pcap library** for advanced packet filtering.

**📚 11. References**

* *Beej's Guide to Network Programming*
* Linux man pages: socket(2), ip(7), udp(7)
* RFC 791 – Internet Protocol
* Wireshark User Guide
* Linux Raw Sockets Tutorial (GeeksforGeeks)

**🔍 Detailed Definitions & Types of All Key Components in the Project**

**⚙️ 1. Socket**

**➤ Definition:**

A **socket** is an endpoint for sending or receiving data across a computer network. It provides a programming interface (API) to access the network stack implemented by the OS.

**🧪 2. Raw Socket**

**➤ Definition:**

A **Raw Socket** is a special type of socket that allows direct sending and receiving of IP packets **without any transport layer formatting** like TCP or UDP. With raw sockets, you get **complete control** over the **packet headers** (IP, TCP, UDP, ICMP).

c

CopyEdit

int sockfd = socket(AF\_INET, SOCK\_RAW, protocol);

Raw sockets allow reading or writing of packets as they appear "on the wire."

**🔰 Why Raw Sockets?**

* Used for **network diagnostics**, **custom protocol implementation**, or **packet sniffing**.
* Bypass OS's default transport-layer handling.

**📦 Types of Raw Sockets (based on protocol used):**

| **Protocol** | **Value** | **Purpose / Use-case** |
| --- | --- | --- |
| IPPROTO\_RAW | 255 | Create and send custom-built packets. (Client in your case) |
| IPPROTO\_TCP | 6 | Sniff TCP packets (for analysis) |
| IPPROTO\_UDP | 17 | Sniff UDP packets (used in your chat) |
| IPPROTO\_ICMP | 1 | Ping, traceroute tools (e.g., sniffing ICMP) |
| IPPROTO\_IP | 0 | Receive all protocols (Linux specific) |

**Note**: Only recvfrom() works with TCP/UDP raw sockets for receiving — for sending TCP packets, you often need a fully raw (IPPROTO\_RAW) socket.

**🛡️ Important Requirements**

* **Root privileges** are required.
* Firewall rules and OS protections may restrict raw socket usage.
* On Linux, raw sockets for some protocols (e.g., TCP) are **receive-only** by default.

**📡 3. Internet Protocols Used**

**🔹 IP (Internet Protocol)**

Layer 3 protocol used for addressing and routing packets.

**IP Header Structure:**

* Version (IPv4/IPv6)
* Header Length
* Source & Destination IP
* Time to Live (TTL)
* Protocol (TCP/UDP/ICMP)
* Checksum

You manually fill this in when using IP\_HDRINCL.

**🔹 UDP (User Datagram Protocol)**

**➤ Connectionless transport layer protocol.**

* Lightweight
* No handshake or retransmission
* Ideal for real-time communication (chat, games, video)

**UDP Header Structure:**

* Source Port
* Destination Port
* Length
* Checksum

In your project, **UDP is used to send messages via raw socket**.

**🔹 TCP (Transmission Control Protocol)**

**➤ Connection-oriented transport protocol.**

* Ensures delivery
* Uses 3-way handshake
* Not used for sending in your raw socket, but can be sniffed.

**🔹 ICMP (Internet Control Message Protocol)**

* Used for control messages (e.g., ping).
* Sniffed for diagnostics, but **not directly used** here.

**🧰 4. Socket API Constants**

| **Constant** | **Meaning** |
| --- | --- |
| AF\_INET | Address family: IPv4 |
| SOCK\_RAW | Type: Raw socket |
| IPPROTO\_RAW | Protocol: Manually define all headers |
| IPPROTO\_UDP | Protocol: Capture UDP packets |
| IPPROTO\_TCP | Protocol: Capture TCP packets |
| SO\_REUSEADDR | Allows reuse of local address |

**🧾 5. IP\_HDRINCL**

**➤ Definition:**

A socket option (setsockopt) to inform the kernel that **you will manually provide the IP header** when sending data.

c

CopyEdit

int one = 1;

setsockopt(sockfd, IPPROTO\_IP, IP\_HDRINCL, &one, sizeof(one));

**🧠 6. Packet Sniffer**

**➤ Definition:**

A tool/program that captures and inspects data packets flowing through a network. Your implementation uses raw sockets to **sniff packets**, parse headers, and analyze metrics.

**📏 7. Jitter**

**➤ Definition:**

Jitter is the **variation in the delay** of received packets. High jitter causes poor performance in VoIP or real-time apps.

**➤ Formula:**

plaintext

CopyEdit

Jitter = |(arrival\_n - arrival\_(n-1)) - (expected\_interval)|

You calculate this using gettimeofday() or clock\_gettime().

**📉 8. Packet Loss**

**➤ Definition:**

Occurs when packets are dropped or lost in transmission before reaching the receiver.

**➤ Detection Method:**

Use **sequence numbers** in your packet payloads. Any skipped sequence indicates a loss.

**🧪 9. Wireshark**

**➤ Definition:**

Wireshark is a free and open-source packet analyzer. It captures network packets and displays them in detail — from Ethernet to IP, TCP, UDP layers.

**➤ Used For:**

* Verifying custom-built packets
* Visualizing IP/UDP headers
* Measuring delay, loss, and retransmissions
* Filtering by port, IP, protocol

**➤ Useful Filters:**

* udp.port == 9090
* ip.addr == 192.168.1.10
* frame.number == 50

**📁 10. Common Libraries Used**

| **Library** | **Use** |
| --- | --- |
| <sys/socket.h> | Socket API functions |
| <netinet/ip.h> | IP header structure |
| <netinet/udp.h> | UDP header structure |
| <arpa/inet.h> | inet\_pton, inet\_ntoa for IP |
| <unistd.h> | close(), read(), write() |
| <sys/time.h> | gettimeofday() for jitter |
| <string.h> | memcpy, memset |

**🖥️ 11. System Requirements**

| **Requirement** | **Reason** |
| --- | --- |
| Linux OS (Ubuntu/Debian) | Allows raw socket use |
| Root privileges | Raw socket access |
| Wireshark installed | Packet analysis |
| Two systems or two terminal windows | Client-Server testing |

**🎓 Expected Questions & Answers from Raw Socket Project**

**🧠 BASIC CONCEPTUAL QUESTIONS**

**1. What is a socket?**

**Answer**:  
A socket is an endpoint for communication between two machines over a network. It provides a way for programs to exchange data using standard protocols like TCP, UDP, or even custom protocols via raw sockets.

**2. What is a raw socket?**

**Answer**:  
A raw socket gives direct access to lower-layer protocols like IP and allows applications to construct and send packets with custom headers (IP, TCP, UDP, etc.). It bypasses the OS's transport layer processing.

**3. Why do we use raw sockets in this project?**

**Answer**:  
We use raw sockets to have complete control over packet structure and to implement functionalities like packet sniffing, jitter calculation, and packet loss detection, which are not possible with standard sockets.

**4. What privileges are needed for raw sockets?**

**Answer**:  
Raw sockets require **root (administrator) privileges** because they can manipulate low-level network traffic and pose a potential security risk.

**5. What is the difference between TCP, UDP, and raw sockets?**

**Answer**:

* **TCP**: Connection-oriented, reliable.
* **UDP**: Connectionless, faster, but unreliable.
* **Raw Socket**: Works below TCP/UDP, gives access to raw packet data.

**6. What is IP\_HDRINCL and why do we use it?**

**Answer**:  
IP\_HDRINCL is a socket option that tells the kernel that the application will provide the IP header manually. This is necessary when using raw sockets to craft custom IP packets.

**🔧 IMPLEMENTATION-BASED QUESTIONS**

**7. What is the structure of a raw UDP packet?**

**Answer**:  
A raw UDP packet includes:

* **IP Header** (manually filled)
* **UDP Header** (manually filled)
* **Payload/Data**

**8. How do you calculate jitter in your project?**

**Answer**:  
We calculate jitter by measuring the difference in arrival times of successive packets and taking the variation from expected delay.  
Example:

c

CopyEdit

jitter = abs((arrival2 - arrival1) - expected\_delay);

**9. How is packet loss calculated?**

**Answer**:  
By embedding sequence numbers in each message. If a sequence number is skipped on the receiver side, a packet is considered lost.

**10. What system calls are used in this project?**

**Answer**:

* socket(): To create raw sockets.
* sendto() / recvfrom(): To send and receive raw packets.
* setsockopt(): To set options like IP\_HDRINCL.
* gettimeofday(): For jitter measurement.

**11. What protocol did you use for communication?**

**Answer**:  
We used **UDP** for communication because it is lightweight and better suited for real-time applications. We used **raw sockets** to construct and analyze UDP packets manually.

**12. Why not use TCP for raw communication?**

**Answer**:  
TCP is complex to implement with raw sockets due to its connection state and handshaking. Also, Linux doesn’t allow sending raw TCP packets easily without additional kernel configuration or tools like libnet.

**🧪 WIRESHARK-RELATED QUESTIONS**

**13. How did you use Wireshark in this project?**

**Answer**:  
Wireshark was used to:

* Capture custom-built packets.
* Analyze header contents (IP, UDP).
* Measure delay, jitter, and packet loss visually.
* Verify correct protocol usage and structure.

**14. What filters did you use in Wireshark?**

**Answer**:

* udp.port == 9090
* ip.addr == <client\_ip>
* frame contains "<message>"
* icmp, tcp, udp, etc., depending on protocol

**🧵 PROJECT DESIGN & ANALYSIS QUESTIONS**

**15. What are the components of your project?**

**Answer**:

1. **Client Program**: Sends UDP packets via raw sockets.
2. **Server Program**: Receives and analyzes packets.
3. **Packet Sniffer**: Captures and displays packet details.
4. **Jitter and Loss Module**: Measures performance.
5. **Wireshark**: Used for external analysis.

**16. What problems did you face during implementation?**

**Answer**:

* Need for root privileges.
* Complexity in constructing headers manually.
* Ensuring correct byte ordering.
* Timing accuracy for jitter.
* OS-level restrictions on sending TCP raw packets.

**17. What is the benefit of using raw sockets over standard sockets?**

**Answer**: They provide fine-grained control over network packets — enabling advanced diagnostics, testing, protocol creation, and analysis beyond the application layer.

**18. How do you ensure data integrity in your setup?**

**Answer**: We manually set and verify header checksums. We also log and compare sequence numbers to detect anomalies or loss.

**19. What tools other than Wireshark could be used for analysis?**

**Answer**:

* **tcpdump** (CLI version of Wireshark)
* **Netcat (nc)** for basic testing
* **Scapy (Python)** for custom packet crafting
* **Ping, traceroute** (for ICMP-level testing)

**20. Can this project be extended? How?**

**Answer**: Yes, it can be extended by:

* Adding encryption for secure packets.
* Visualizing metrics (graphs).
* Supporting IPv6.
* Implementing retransmission logic.
* Extending to video/audio real-time data.

**21. What’s the difference between packet sniffing and spoofing?**

**Answer**:

* **Sniffing**: Capturing and analyzing packets passively.
* **Spoofing**: Sending packets with forged headers to impersonate another device.

**22. Can this project cause network disruption?**

**Answer**: Yes, if misused (e.g., sending malformed packets or flood packets). That's why raw sockets are restricted and need admin access.

**🧰 ADVANCED QUESTIONS**

**23. What is the purpose of checksums in IP/UDP?**

**Answer**: Checksums ensure that the data in the packet headers (and optionally the payload) hasn’t been corrupted during transmission.

**24. How is endianness handled in header fields?**

**Answer**: We use functions like htons() (host to network short) and htonl() (host to network long) to ensure correct byte order in multi-byte fields.

**25. What is MTU and its role in your project?**

**Answer**: **MTU (Maximum Transmission Unit)** is the largest size a packet can be on the network without being fragmented. We must keep our custom packet size under MTU to avoid fragmentation issues.

**✅ Other Important Details You Should Know for This Project**

**🧱 1. Complete Layer-wise Understanding (OSI & TCP/IP Models)**

Know exactly where your project fits in the network stack.

| **Layer** | **What You’re Doing** |
| --- | --- |
| **Application Layer** | Chat messages, control logic |
| **Transport Layer** | UDP headers constructed manually |
| **Network Layer** | IP headers manually formed |
| **Link Layer** | Wireshark captures Ethernet frames (but not manipulated by your code unless you use PF\_PACKET) |

**🧾 2. Header Fields and Structures (You Must Know These)**

**🔹 IP Header (for IPv4)**

* Version (4 bits)
* Header Length (IHL)
* Type of Service (TOS)
* Total Length
* Identification
* Flags & Fragment Offset
* TTL (Time To Live)
* Protocol (e.g., 17 for UDP)
* Header Checksum
* Source & Destination IP

**🔹 UDP Header**

* Source Port
* Destination Port
* Length
* Checksum

You should be able to draw the structure and explain each field with size (in bits/bytes).

**🛠️ 3. Functions You Used in C**

Be ready to explain the purpose of:

* socket()
* sendto() / recvfrom()
* setsockopt()
* htons() / htonl() / ntohs() / ntohl()
* gettimeofday()
* memcpy() / memset()
* strncpy() / printf() / perror()

**🧮 4. Calculations You Perform**

**✅ Jitter**

Difference in arrival intervals:

text

CopyEdit

Jitter = |(T2 - T1) - (S2 - S1)|

**✅ Packet Loss**

If sender sends N packets and receiver receives M:

text

CopyEdit

Loss Rate = ((N - M) / N) \* 100%

**📉 5. Performance Metrics & Logging**

You should log:

* Timestamp of each sent and received packet
* Sequence numbers
* Round Trip Time (optional)
* Jitter values
* Packet drops

**🧪 6. Testing Scenarios**

Be able to explain:

* What happens when packets are delayed/lost
* How the receiver handles out-of-order or duplicate packets (if at all)
* What changes under poor network conditions

**🔓 7. Security Implications**

Know the risks:

* Raw sockets can be used for **IP spoofing**
* Potential **DoS attacks** if flooded
* Needs **root access**, so care must be taken in system configuration

**🌐 8. Wireshark Analysis Must-Knows**

Be prepared to explain:

* How to filter for your traffic
* How to identify your packet (e.g., via custom data or port)
* How to verify checksum, TTL, protocol, source/destination
* How to read hex and dissect fields

**🧰 9. Alternative Tools & Enhancements**

You can propose:

* Use **Scapy (Python)** for easier custom packet manipulation
* Extend to **ICMP** or **TCP** with deeper packet crafting
* Visualize jitter/loss using **matplotlib** or a frontend chart

**🧪 10. Project Testing & Validation**

You should document:

* Local vs remote testing
* Packet rates (how fast you send)
* How you verified the correctness (matching Wireshark with logs)
* OS-level differences (e.g., behavior on Linux vs Windows)

**🧑‍💻 11. System Setup Requirements**

List out:

* OS: Linux-based (e.g., Ubuntu)
* Root access or sudo
* GCC or Clang for compiling
* Wireshark installed
* Network interfaces used (e.g., eth0, wlan0)

**🛠️ 12. Possible Errors and Debugging**

Examples:

* Operation not permitted → run as root
* Packet not seen in Wireshark → wrong interface or filter
* Incorrect checksum → miscalculated manually
* Segmentation fault → wrong header memory alignment

**📘 13. Future Enhancements You Can Mention**

* Add GUI to visualize packet stats
* Real-time graphs of jitter/loss
* Encryption of messages
* ARP/DNS level spoofing detection
* Use PF\_PACKET sockets to access data link layer