Title of the Experiment: Comparative Traffic Analysis of Real-time VoIP (RTP/UDP) vs. Bulk Data Transfer (TCP) using Wireshark

Submitted by:

Name: Prachi Sharnagat **Roll No:** 20 **Branch / Semester:** A2 CSE (III SEM) **Batch:** B2	**Name:** Mansi Gupta **Roll No:** 32 **Branch / Semester:** A2 CSE (III SEM) **Batch:** B2	**Name:** Rashi Sukhwani **Roll No:** 13 **Branch / Semester:** A2 CSE (III SEM) **Batch:** B1

GitHub Link : https://github.com/Prachi-Sharnagat/NetworkTrafficAnalysis

1. Aim / Objective

To capture, analyze, and compare the network traffic characteristics of two different application types:

1. A real-time Voice over IP (VoIP) call, to observe the behavior of UDP and RTP.
2. A large file download, to observe the behavior of the TCP. The primary goal is to use the Wireshark tool to gather empirical evidence of the protocol differences between applications that prioritize speed (VoIP) versus those that prioritize reliability (file download).

2. Theory / Conceptual Background

Computer networks use a suite of protocols to manage communication. The Transport Layer is primarily governed by two key protocols: TCP and UDP.

- **UDP (User Datagram Protocol):** This is a "connectionless" protocol. It is very fast and has low overhead because it simply sends data packets (datagrams) without establishing a formal connection or checking if they arrive. It is ideal for time-sensitive applications like online gaming or voice calls, where speed is more important than perfect data integrity.

- **RTP (Real-time Transport Protocol):** This protocol runs *on top* of UDP and is designed specifically for streaming media. It adds timestamps and sequence numbers to the UDP packets, allowing the receiving application to handle packets in the correct order and measure performance metrics like jitter and packet loss.

- **TCP (Transmission Control Protocol):** This is a "connection-oriented" protocol. It is built for reliability. Before sending any data, TCP establishes a stable connection via a **three-way handshake (SYN, SYN-ACK, ACK)**. It then meticulously tracks every packet sent, requiring acknowledgments from the receiver. If a packet is lost, TCP automatically re-transmits it, ensuring the data is 100% complete and uncorrupted. It is used for web browsing, file downloads, and email.

This experiment will capture and visualize this fundamental trade-off: UDP/RTP's speed vs. TCP's reliability.

3. Software / Simulation Tool Used

- **Tool:** Wireshark (v4.x)
- **Applications Used for Traffic Generation:** Discord (desktop client), Google Chrome (for file download)

4. Hardware / Software Requirements

- **Hardware:**
  - Processor: Intel Core i3 or above
  - RAM: 8 GB
  - Network: Active Internet Connection (Wi-Fi or Ethernet)
- **Software:**
  - Operating System: Windows 11
  - Simulation Tool: Wireshark (v4.2.0 or similar)
  - Other: Discord, Web Browser

5. Procedure / Steps Performed

The experiment was conducted in two distinct phases:

Phase A: VoIP (RTP/UDP) Traffic Capture & Analysis

1. Launched Wireshark and selected the active network interface.
2. Started a new packet capture.
3. Immediately started a voice call on the Discord desktop application and spoke for approximately 90 seconds.
4. Ended the voice call and then immediately stopped the Wireshark capture.
5. Filtered the captured packets by udp to locate the conversation.

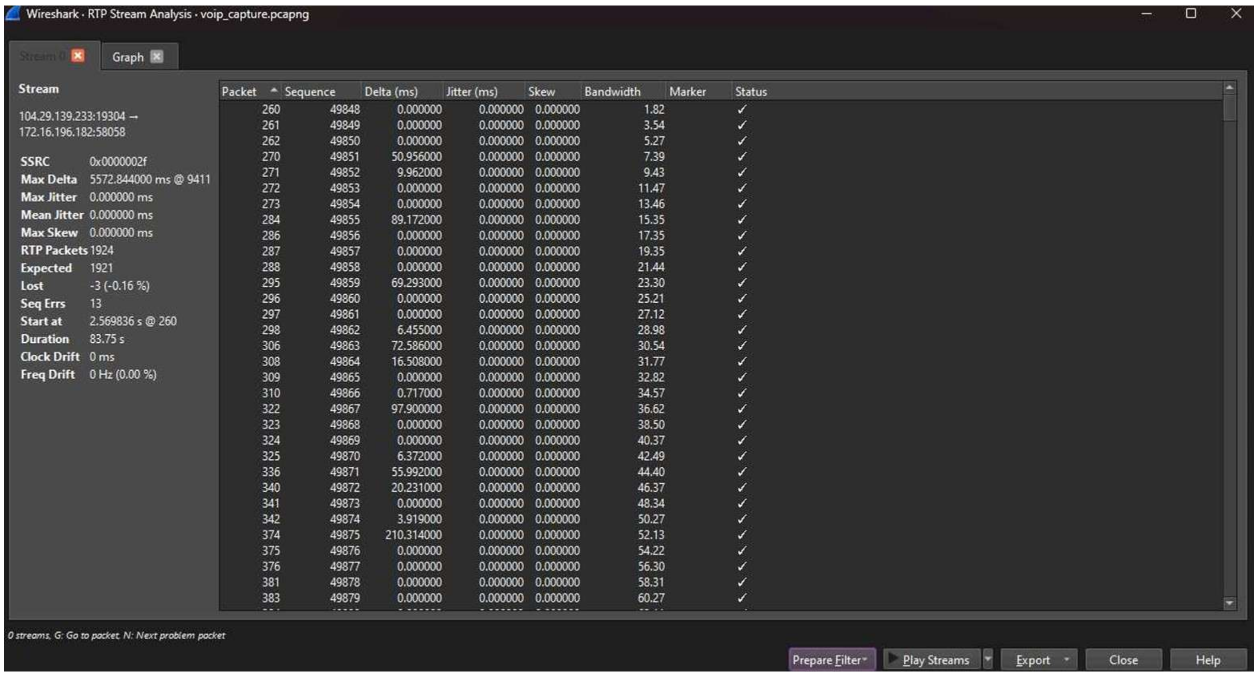
6. Right-clicked the relevant UDP stream and used the Decode As... feature to correctly interpret the packets as RTP.
7. Filtered the display for rtp to isolate the voice stream.
8. Opened the Telephony -> RTP -> RTP Streams analysis tool to gather statistics on packet loss and jitter.

Phase B: File Download (TCP) Traffic Capture & Analysis

1. Started a new, separate capture in Wireshark.
2. Immediately opened a web browser and began downloading a large file (the Ubuntu Desktop ISO).
3. Allowed the download to run for approximately 30 seconds.
4. Paused the download in the browser and then immediately stopped the Wireshark capture.
5. Located a TLSv1.3 packet from the download stream and right-clicked.
6. Used the Follow -> TCP Stream feature to isolate the entire conversation.
7. Analyzed the filtered stream, starting from the top, to observe the TCP handshake and data transfer.

6. Simulation Diagram / Screenshot

Screenshot 1: VoIP (RTP) Stream Analysis



Stream

104.29.139.233:19304 →
172.16.196.182:58058

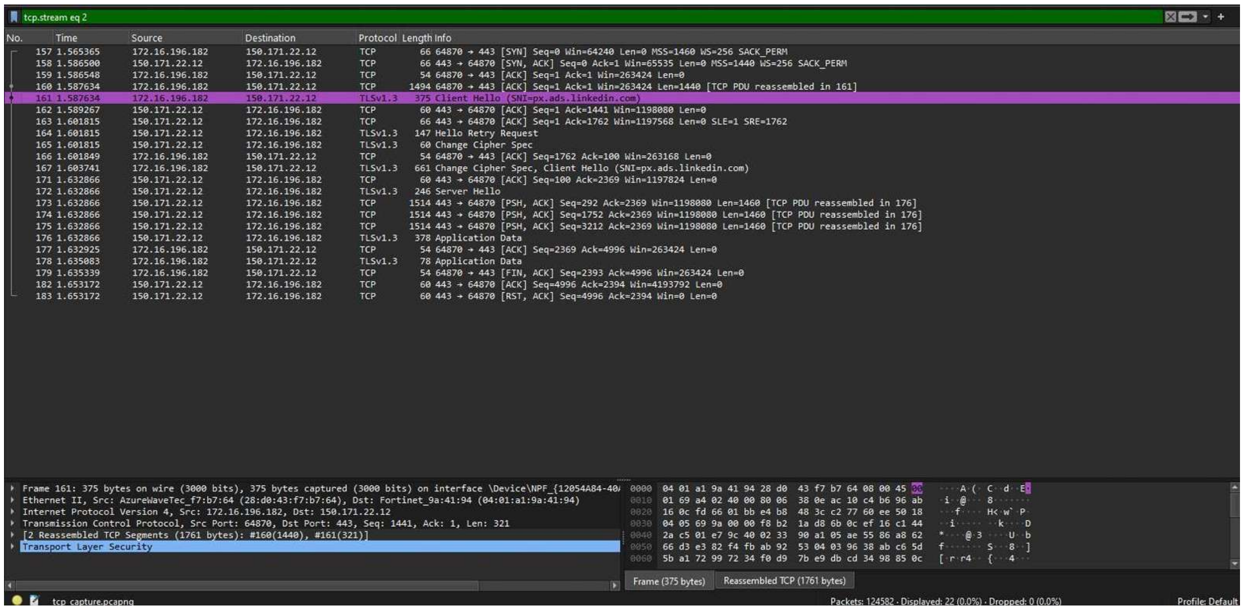
SSRC	0x0000002f
Max Delta	5572.844000 ms @ 9411
Max Jitter	0.000000 ms
Mean Jitter	0.000000 ms
Max Skew	0.000000 ms
RTP Packets	1924
Expected	1921
Lost	-3 (-0.16 %)
Seq Errs	13
Start at	2.569836 s @ 260
Duration	83.75 s
Clock Drift	0 ms
Freq Drift	0 Hz (0.00 %)

Packet	Sequence	Delta (ms)	Jitter (ms)	Skew	Bandwidth	Marker	Status
260	49848	0.000000	0.000000	0.000000	1.82		✓
261	49849	0.000000	0.000000	0.000000	3.54		✓
262	49850	0.000000	0.000000	0.000000	5.27		✓
270	49851	50.956000	0.000000	0.000000	7.39		✓
271	49852	9.962000	0.000000	0.000000	9.43		✓
272	49853	0.000000	0.000000	0.000000	11.47		✓
273	49854	0.000000	0.000000	0.000000	13.46		✓
284	49855	89.172000	0.000000	0.000000	15.35		✓
286	49856	0.000000	0.000000	0.000000	17.35		✓
287	49857	0.000000	0.000000	0.000000	19.35		✓
288	49858	0.000000	0.000000	0.000000	21.44		✓
295	49859	69.293000	0.000000	0.000000	23.30		✓
296	49860	0.000000	0.000000	0.000000	25.21		✓
297	49861	0.000000	0.000000	0.000000	27.12		✓
298	49862	6.455000	0.000000	0.000000	28.98		✓
306	49863	72.586000	0.000000	0.000000	30.54		✓
308	49864	16.508000	0.000000	0.000000	31.77		✓
309	49865	0.000000	0.000000	0.000000	32.82		✓
310	49866	0.717000	0.000000	0.000000	34.57		✓
322	49867	97.900000	0.000000	0.000000	36.62		✓
323	49868	0.000000	0.000000	0.000000	38.50		✓
324	49869	0.000000	0.000000	0.000000	40.37		✓
325	49870	6.372000	0.000000	0.000000	42.49		✓
336	49871	55.992000	0.000000	0.000000	44.40		✓
340	49872	20.231000	0.000000	0.000000	46.37		✓
341	49873	0.000000	0.000000	0.000000	48.34		✓
342	49874	3.919000	0.000000	0.000000	50.27		✓
374	49875	210.314000	0.000000	0.000000	52.13		✓
375	49876	0.000000	0.000000	0.000000	54.22		✓
376	49877	0.000000	0.000000	0.000000	56.30		✓
381	49878	0.000000	0.000000	0.000000	58.31		✓
383	49879	0.000000	0.000000	0.000000	60.27		✓

0 streams, G: Go to packet, N: Next problem packet

Figure 1: Analysis of the RTP stream from a Discord call, showing packet loss and jitter statistics.

Screenshot 2: TCP Stream Analysis



```
tcp.stream eq 2

No.   Time      Source          Destination     Protocol Length Info
  157 1.565365  172.16.196.182  150.171.22.12   TCP        66 64870 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM
  158 1.586500  150.171.22.12   172.16.196.182  TCP        66 443 → 64870 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1440 WS=256 SACK_PERM
  159 1.586548  172.16.196.182  150.171.22.12   TCP        54 64870 → 443 [ACK] Seq=1 Ack=1 Win=263424 Len=0
  160 1.587634  172.16.196.182  150.171.22.12   TCP      1494 64870 → 443 [ACK] Seq=1 Ack=1 Win=263424 Len=1440 [TCP PDU reassembled in 161]
  161 1.587634  172.16.196.182  150.171.22.12   TLSv1.3   375 Client Hello (SNI=px.ads.linkedin.com)
  162 1.589267  150.171.22.12   172.16.196.182  TCP        60 443 → 64870 [ACK] Seq=1 Ack=1441 Win=1198080 Len=0
  163 1.601815  150.171.22.12   172.16.196.182  TCP        66 443 → 64870 [ACK] Seq=1 Ack=1762 Win=1197568 Len=0 SLE=1 SRE=1762
  164 1.601815  150.171.22.12   172.16.196.182  TLSv1.3   147 Hello Retry Request
  165 1.601815  150.171.22.12   172.16.196.182  TLSv1.3    60 Change Cipher Spec
  166 1.601849  172.16.196.182  150.171.22.12   TCP        54 64870 → 443 [ACK] Seq=1762 Ack=100 Win=263168 Len=0
  167 1.603741  172.16.196.182  150.171.22.12   TLSv1.3   661 Change Cipher Spec, Client Hello (SNI=px.ads.linkedin.com)
  171 1.632866  150.171.22.12   172.16.196.182  TCP        60 443 → 64870 [ACK] Seq=100 Ack=2369 Win=1197824 Len=0
  172 1.632866  150.171.22.12   172.16.196.182  TLSv1.3   246 Server Hello
  173 1.632866  150.171.22.12   172.16.196.182  TCP      1514 443 → 64870 [PSH, ACK] Seq=292 Ack=2369 Win=1198080 Len=1460 [TCP PDU reassembled in 176]
  174 1.632866  150.171.22.12   172.16.196.182  TCP      1514 443 → 64870 [PSH, ACK] Seq=1752 Ack=2369 Win=1198080 Len=1460 [TCP PDU reassembled in 176]
  175 1.632866  150.171.22.12   172.16.196.182  TCP      1514 443 → 64870 [PSH, ACK] Seq=3212 Ack=2369 Win=1198080 Len=1460 [TCP PDU reassembled in 176]
  176 1.632866  150.171.22.12   172.16.196.182  TLSv1.3   378 Application Data
  177 1.632925  172.16.196.182  150.171.22.12   TCP        54 64870 → 443 [ACK] Seq=2369 Ack=4996 Win=263424 Len=0
  178 1.635083  172.16.196.182  150.171.22.12   TLSv1.3    78 Application Data
  179 1.635339  172.16.196.182  150.171.22.12   TCP        54 64870 → 443 [FIN, ACK] Seq=2393 Ack=4996 Win=263424 Len=0
  182 1.653172  150.171.22.12   172.16.196.182  TCP        60 443 → 64870 [ACK] Seq=4996 Ack=2394 Win=4193792 Len=0
  183 1.653172  150.171.22.12   172.16.196.182  TCP        60 443 → 64870 [RST, ACK] Seq=4996 Ack=2394 Win=0 Len=0

▸ Frame 161: 375 bytes on wire (3000 bits), 375 bytes captured (3000 bits) on interface \Device\NPF_{12054A84-40/
▸ Ethernet II, Src: AzureWaveTec_f7:b7:64 (28:d0:43:f7:b7:64), Dst: Fortinet_9a:41:94 (04:01:a1:9a:41:94)
▸ Internet Protocol Version 4, Src: 172.16.196.182, Dst: 150.171.22.12
▸ Transmission Control Protocol, Src Port: 64870, Dst Port: 443, Seq: 1441, Ack: 1, Len: 321
▸ [2 Reassembled TCP Segments (1761 bytes): #160(1440), #161(321)]
▸ Transport Layer Security

0000  04 01 a1 9a 41 94 28 d0  43 f7 b7 64 08 00 45 00
0010  01 69 a4 02 40 00 80 06  38 0e ac 10 c4 b6 96 ab
0020  16 0c fd 66 01 bb e4 b8  48 3c c2 77 60 ee 50 18
0030  04 05 69 9a 00 00 f8 b2  1a d8 6b 0c ef 16 c1 44
0040  2a c5 01 e7 9c 40 02 33  90 a1 05 ae 55 86 a8 62
0050  66 d3 e3 82 f4 fb ab 92  53 04 03 96 38 ab c6 5d
0060  5b a1 72 99 72 34 f0 d9  7b e9 db cd 34 98 85 0c

Frame (375 bytes)   Reassembled TCP (1761 bytes)

tcp_capture.pcapng    Packets: 124582 · Displayed: 22 (0.0%) · Dropped: 0 (0.0%)    Profile: Default
```

Figure 2: Analysis of the filtered TCP stream from a file download, showing the three-way handshake and data packets.

7. Observations

From the captured data and screenshots, we made the following observations:

- **RTP/UDP Traffic:**
  - The traffic consisted of a large number of small, continuous packets.
  - The RTP Stream Analysis tool (Figure 1) showed a packet loss of **-3 (-0.16%)**, which is extremely low and acceptable for a real-time call.
  - The protocol did not show any connection setup (like a handshake); it began sending data immediately.
- **TCP Traffic:**
  - The conversation (Figure 2) clearly began with the **TCP three-way handshake** ([SYN], [SYN, ACK], [ACK]), which establishes the connection.
  - The data was sent in large, efficient "Application Data" packets.
  - The TCP stream also included acknowledgment ([ACK]) packets being sent back to the server to confirm receipt.

8. Result / Output

We successfully used Wireshark to capture and analyze two distinct types of network traffic. We were able to:

1. Isolate a UDP stream, decode it as RTP, and analyze its real-time performance metrics (packet loss, jitter).
2. Isolate a TCP stream and identify its key reliability features, most notably the three-way handshake.

9. Conclusion

This experiment successfully provided a practical demonstration of the core differences between UDP and TCP. The results confirm the theoretical concepts:

- **VoIP (RTP/UDP)** prioritizes speed and low latency by sacrificing the overhead of guaranteed delivery. This is essential for a real-time call.
- **File Download (TCP)** prioritizes 100% data integrity by using a robust, connection-oriented protocol that performs handshakes and acknowledgments. This is essential for ensuring a file is not corrupted.

We conclude that the choice of transport protocol is fundamentally dictated by the specific needs and goals of the application.

10. Viva Questions

1. What is the main difference between TCP and UDP?

2. Why can't you use TCP for a real-time voice call?

3. What is the purpose of the three-way handshake in TCP?

4. What is RTP, and why is it used on top of UDP?

5. In our TCP capture (Figure 2), what does [SYN] stand for, and what is its purpose?