Signaling Report

Network scanning and sniffing

Kareen Raafat 55-25390
Khaled Wael 55-6548
Ganna Reda 55-20722
Hana Ahmed 55-5384
Farah Miniesy 55-2057
Rawan Miniesy 55-24969

Network Scanning:

A Scan of the private network to find the live hosts using the Ping command.

No.	Time	Source	Destination	Protocol	Length	Info
380	11.766542	192.168.1.20	192.168.1.5	ICMP	74	Echo (ping) request id=0x0001, seq=27/6912, ttl=128 (no response found!)
493	16.330863	192.168.1.20	192.168.1.5	ICMP	74	Echo (ping) request id=0x0001, seq=28/7168, ttl=128 (reply in 495)
495	16.378003	192.168.1.5	192.168.1.20	ICMP	74	Echo (ping) reply id=0x0001, seq=28/7168, ttl=64 (request in 493)
532	17.361834	192.168.1.20	192.168.1.5	ICMP	74	Echo (ping) request id=0x0001, seq=29/7424, ttl=128 (reply in 533)
533	17.388441	192.168.1.5	192.168.1.20	ICMP	74	Echo (ping) reply id=0x0001, seq=29/7424, ttl=64 (request in 532)
542	18.375510	192.168.1.20	192.168.1.5	ICMP	74	Echo (ping) request id=0x0001, seq=30/7680, ttl=128 (reply in 543)
543	18.379483	192.168.1.5	192.168.1.20	ICMP	74	Echo (ping) reply id=0x0001, seq=30/7680, ttl=64 (request in 542)

Network Sniffing:

Wire shark Sniff the TCP packets during the call.

Closing Packets:

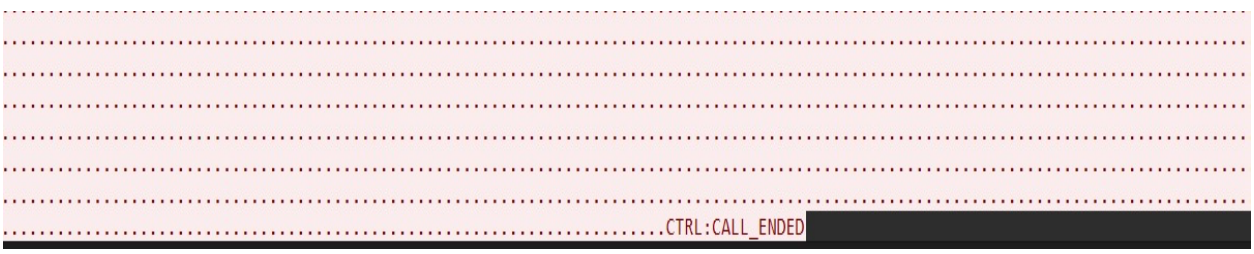
4011	25.379130	192.168.1.5	192.168.1.20	TLCP	1514	Ignored Unknown Record
4012	25.379130	192.168.1.5	192.168.1.20	TLCP	1514	Ignored Unknown Record
4013	25.379130	192.168.1.5	192.168.1.20	TLCP	718	Ignored Unknown Record
4014	25.379193	192.168.1.20	192.168.1.5	TCP	54	53087 → 50020 [RST, ACK] Seq=1725713 Ack=1711043 Win=0 Len=0

Call Started:

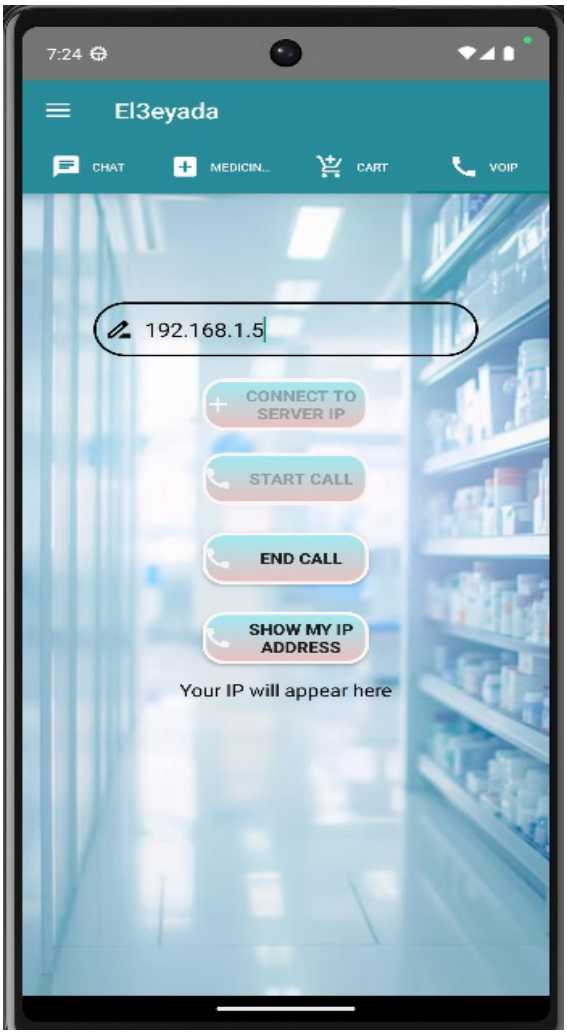
Initiating:

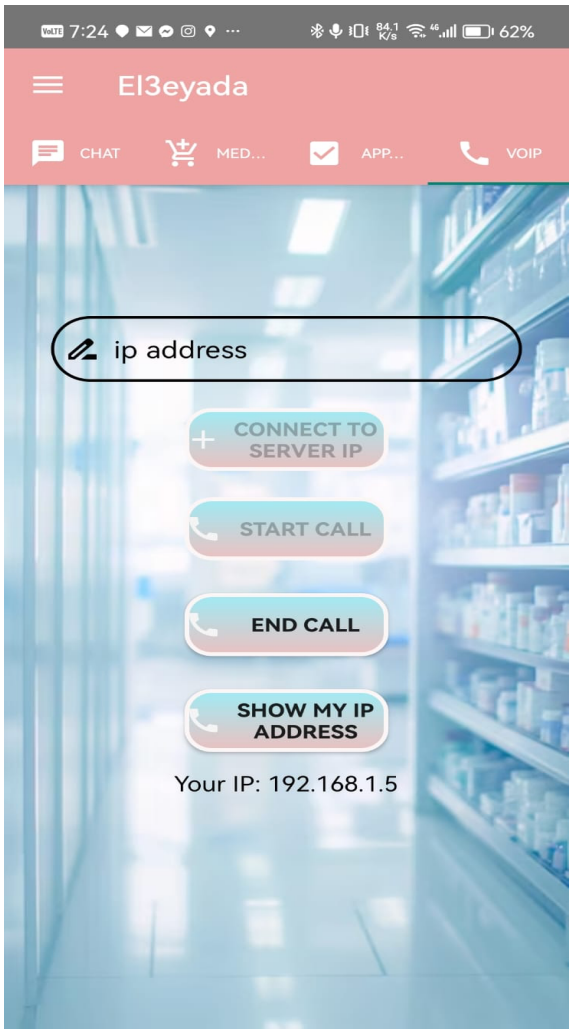
27	5.256048	192.168.1.20	192.168.1.5	TCP	66	54637 → 50020 [SYN] Seq=0 Win=65535 Len=0 MSS=1460 WS=256 SACK_PERM
28	5.257292	192.168.1.20	192.168.1.5	ECHO	123	Request
29	5.435051	192.168.1.5	192.168.1.20	TCP	66	50020 → 54637 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460 SACK_PERM WS=512
30	5.435173	192.168.1.20	192.168.1.5	TCP	54	54637 → 50020 [ACK] Seq=1 Ack=1 Win=65280 Len=0

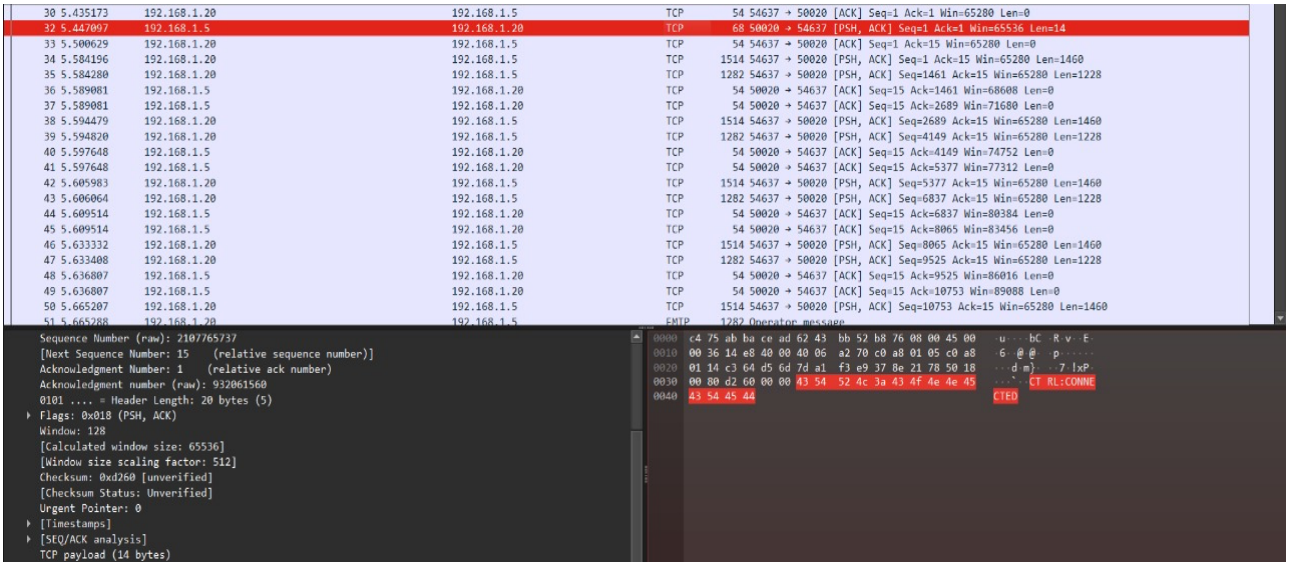
Call Ended:

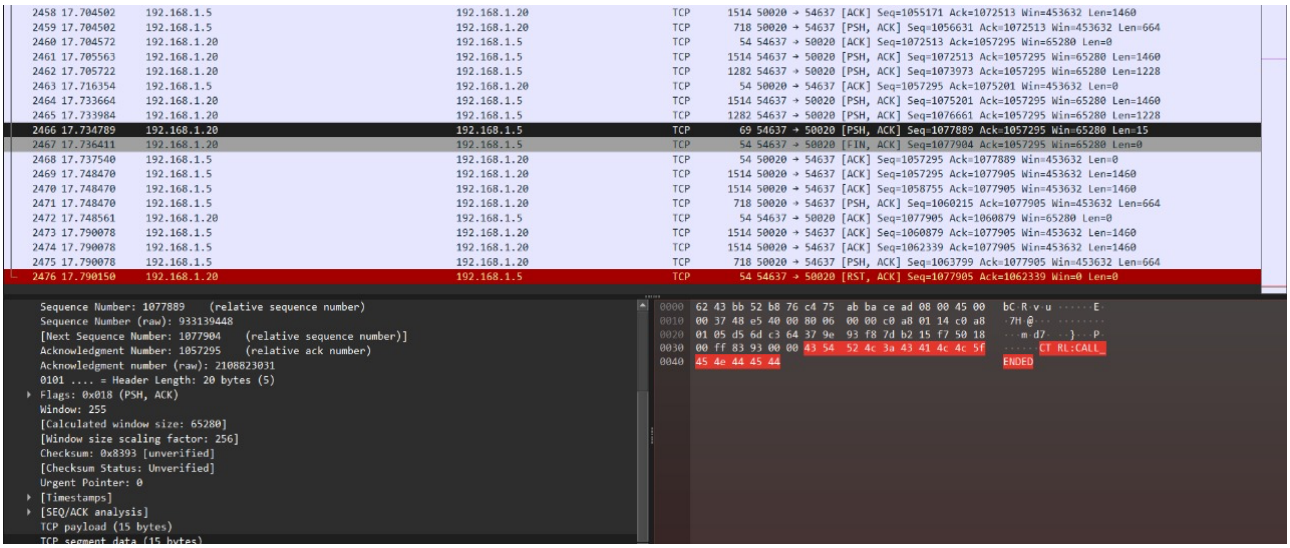


IP address of live host:









Reasons why RTP doesn't appear:

SIP and RTP protocols are not being used explicitly; instead, the application relies on custom communication over TCP (port 50020). Below are the reasons why Wireshark does not show SIP and RTP packets:

Absence of SIP Implementation

SIP (Session Initiation Protocol) is a signaling protocol used to establish, manage, and terminate VoIP calls. In the current implementation, SIP is not utilized for signaling purposes. Instead, custom control messages (e.g., CTRL:CONNECTED) are sent over TCP.

Typically, SIP messages are transmitted on ports like 5060 (default for SIP), but since SIP protocols or libraries are not incorporated, Wireshark cannot detect SIP packets.

Lack of RTP for Media Transmission

RTP (Real-time Transport Protocol) is the standard for transmitting audio and video streams in VoIP communications. The application uses raw socket communication and Android's Audio Record and Audio Track APIs to handle audio data.

Unlike RTP, which operates over UDP, this implementation transmits audio over TCP. Since it does not adhere to RTP standards, Wireshark does not recognize these packets as RTP.

Packet Flow During the VoIP Session:

1. **Connection Setup**
   - **The server starts by listening for incoming TCP connections on port 50020.**
     When the client initiates a connection using the server's IP address and port, a TCP handshake occurs, establishing the communication channel.
2. **Control Messages**
   - **After the TCP connection is established, the server sends a control message (CTRL:CONNECTED) to the client to confirm the connection.**
   - **Throughout the session, additional control messages such as CTRL:CALL_ENDED may be exchanged to signal events like call termination whether the user choose to terminate it or any other reason that lead to call termination.**
3. **Audio Data Transmission**
   - **Capture and Transmission:**
     The client's microphone captures audio using the Audio Record API, and the data is read into a buffer. The buffered audio data is then sent over the established TCP connection to the server in real-time.
   - **Reception and Playback:**
     On the server side, the audio data is received from the TCP connection, buffered, and played back using the Audio Track API.
4. **Unidirectional/Bidirectional Flow**
   - **The packet flow for audio data is bidirectional: both the client and server send and receive audio packets simultaneously to enable real-time communication.**
   - **However, these packets are transmitted as raw byte streams over TCP, without using a standardized media transport protocol like RTP.**
5. **Call Termination**
   - **When either the client or server decides to end the session, a control message (CTRL:CALL_ENDED) is sent to the other party, signaling the termination of the call.**
   - **The TCP connection is then gracefully closed by both sides.**