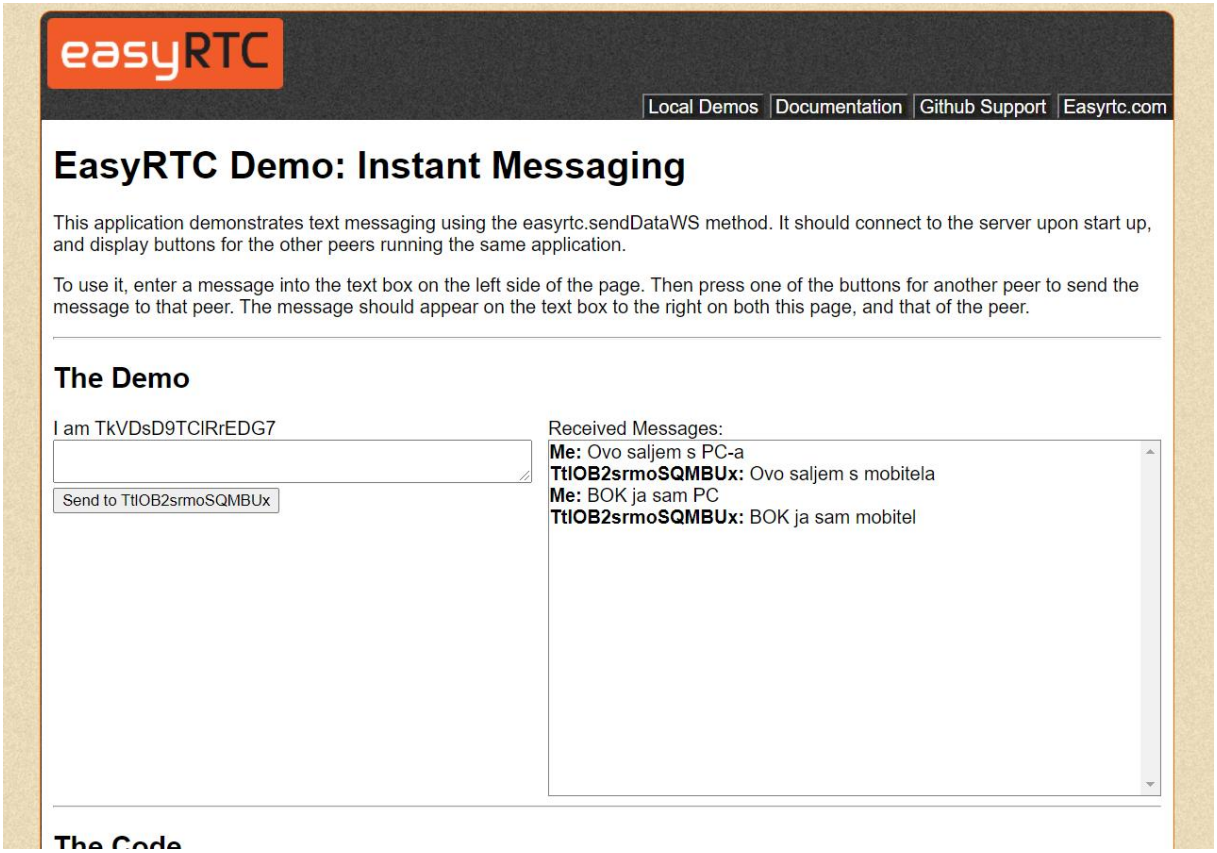
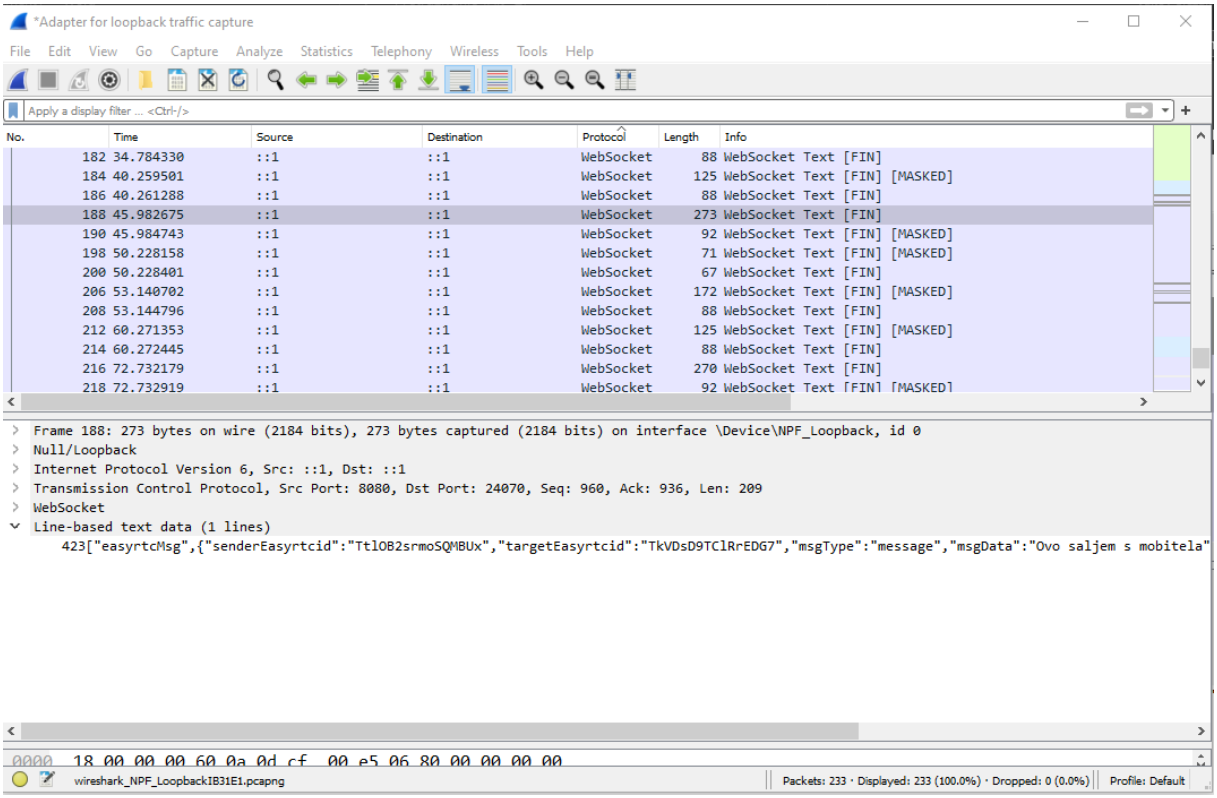
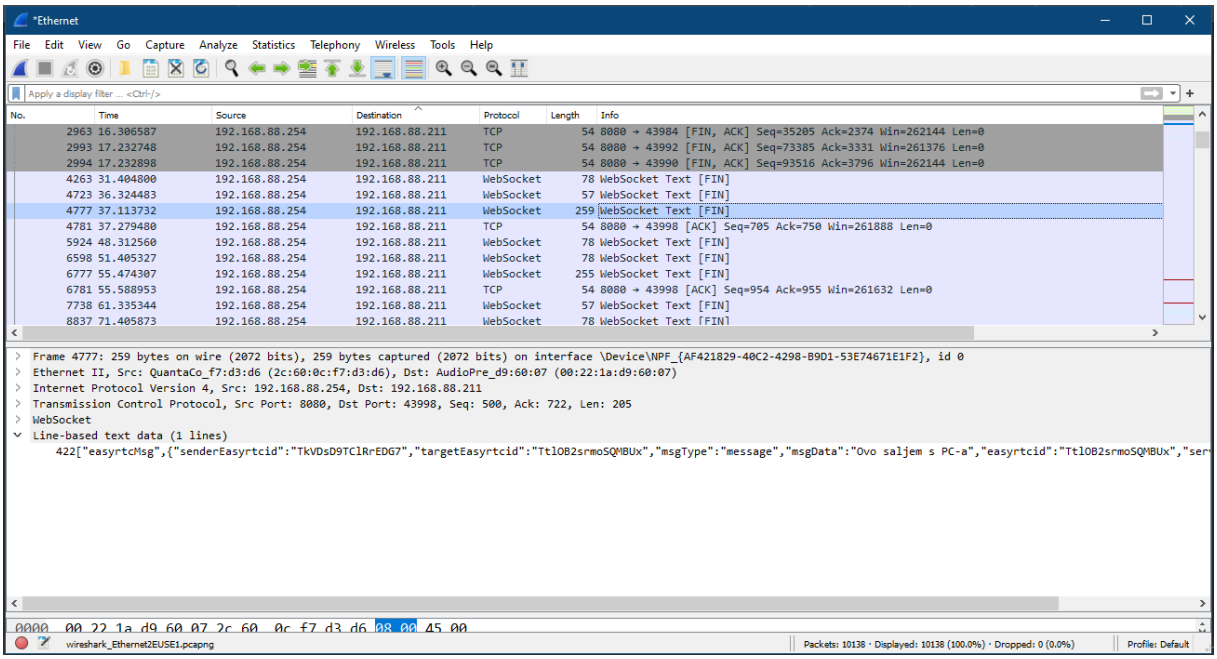
1.)


easyRTC
Local Demos
Documentation
Github Support
Easyrtc.com
EasyRTC Demo: Instant Messaging
This application demonstrates text messaging using the easyrtc.sendDataWS method. It should connect to the server upon start up, and display buttons for the other peers running the same application.
To use it, enter a message into the text box on the left side of the page. Then press one of the buttons for another peer to send the message to that peer. The message should appear on the text box to the right on both this page, and that of the peer.
The Demo
I am TkVDsD9TClRrEDG7
Send to TtlOB2srmoSQMBUx
Received Messages:
Me: Ovo saljem s PC-a
TtlOB2srmoSQMBUx: Ovo saljem s mobitela
Me: BOK ja sam PC
TtlOB2srmoSQMBUx: BOK ja sam mobitel
The Code


*Adapter for loopback traffic capture
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
No. Time Source Destination Protocol Length Info
182 34.784330 ::1 ::1 WebSocket 88 WebSocket Text [FIN]
184 40.259501 ::1 ::1 WebSocket 125 WebSocket Text [FIN] [MASKED]
186 40.261288 ::1 ::1 WebSocket 88 WebSocket Text [FIN]
188 45.982675 ::1 ::1 WebSocket 273 WebSocket Text [FIN]
190 45.984743 ::1 ::1 WebSocket 92 WebSocket Text [FIN] [MASKED]
198 50.228158 ::1 ::1 WebSocket 71 WebSocket Text [FIN] [MASKED]
200 50.228401 ::1 ::1 WebSocket 67 WebSocket Text [FIN]
206 53.140702 ::1 ::1 WebSocket 172 WebSocket Text [FIN] [MASKED]
208 53.144796 ::1 ::1 WebSocket 88 WebSocket Text [FIN]
212 60.271353 ::1 ::1 WebSocket 125 WebSocket Text [FIN] [MASKED]
214 60.272445 ::1 ::1 WebSocket 88 WebSocket Text [FIN]
216 72.732179 ::1 ::1 WebSocket 270 WebSocket Text [FIN]
218 72.732919 ::1 ::1 WebSocket 92 WebSocket Text [FIN] [MASKED]
Frame 188: 273 bytes on wire (2184 bits), 273 bytes captured (2184 bits) on interface \Device\NPF_Loopback, id 0
Null/Loopback
Internet Protocol Version 6, Src: ::1, Dst: ::1
Transmission Control Protocol, Src Port: 8080, Dst Port: 24070, Seq: 960, Ack: 936, Len: 209
WebSocket
Line-based text data (1 lines)
423["easyrtcMsg",{"senderEasyrtcid":"TtlOB2srmoSQMBUx","targetEasyrtcid":"TkVDsD9TClRrEDG7","msgType":"message","msgData":"Ovo saljem s mobitela"
0000 18 00 00 00 60 0a 0d cf 00 e5 06 80 00 00 00 00
wireshark_NPF_LoopbackIB31E1.pcapng
Packets: 233 · Displayed: 233 (100.0%) · Dropped: 0 (0.0%)
Profile: Default



a)

Komunikacija se vrši između mobitela i PC-a, tj. entiteta označenih sa TkVDsD9TClRrEDG7 i TtlOB2srmoSQMBUx, ali poruke idu preko (nodejs) servera koji se isto nalazi na PC-u. Zato se TCP veza uspostavlja između mobitela i servera, te između PC-a (tj. browsera) i servera.

PC – 192.168.88.254

Mobitel – 192.168.88.211

Server (na PC-u) – 192.168.88.254

b)

Kao što se vidi u slikama priloženima gore, koristi se protokol WebSocket za komunikaciju, kojim se onda šalju poruke (nakon što se inicijalno učitao HTML). Naravno ispod njega se koristi TCP, IPv4, te dalje ovisno o korištenoj tehnologiji Ethernet i WiFi.

c)

Klijenti komuniciraju preko poslužitelja. To se može vidjeti i na gore priloženim slikama, jer svaka npr. odlazna poruka sa mobitela odlazi na IP 192.168.88.254:**8080** , te se onda sa tog porta poruka dalje pošalje na već neki proizvoljni port koji je otvorio browser. Isto je i obrnuto, browser šalje sa svojeg proizvoljnog porta na loopback IP adresu na port 8080, a onda se sa 192.168.88.254:**8080** šalje poruka na mobitel na IP 92.168.88.211.

d)

Slike su priložene gore. U jednom wireshark prozoru možemo vidjeti poruku koju server šalje na browser preko loopback IP adrese, dok na drugom koju server šalje mobitelu.

2.)

a)

WebRTC internals je jedan od predefiniranih stranica (kod mene u chrome-u) gdje su ispisane sve statistike trenutnih pokrenutih WebRTC instanci, tj. sesija.

b)

Statistics RTCInboundRTPAudioStream_1413389400

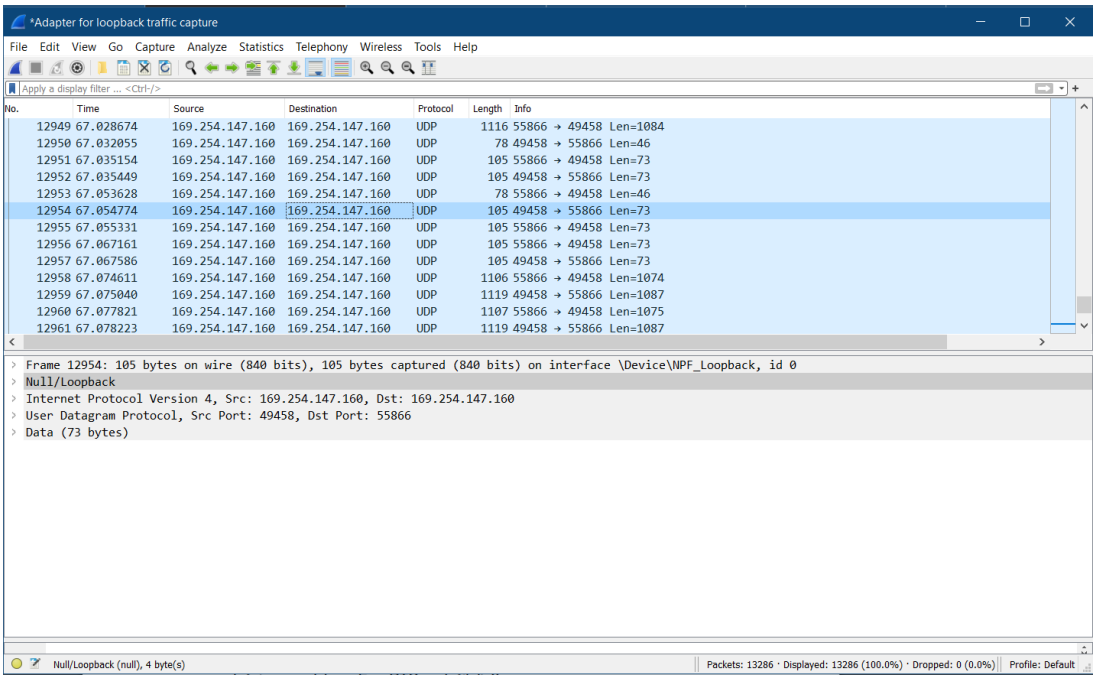
timestamp	21. 12. 2021. 14:46:29
ssrc	1413389400
kind	audio
trackId	RTCMediaStreamTrack_receiver_1
transportId	RTCTransport_0_1
codecId	RTCCodec_0_Inbound_111
[codec]	opus (111, minptime=10;useinbandfec=1)
mediaType	audio
jitter	0
packetsLost	0
packetsDiscarded	0
packetsReceived	23273
[packetsReceived/s]	50.05005836363684
fecPacketsReceived	0
fecPacketsDiscarded	0
bytesReceived	816807
[bytesReceived_in_bits/s]	14014.016341818315
headerBytesReceived	651644

▼ RTCInboundRTPVideoStream_2089870811 (inbound-rtp)

Statistics RTCInboundRTPVideoStream_2089870811

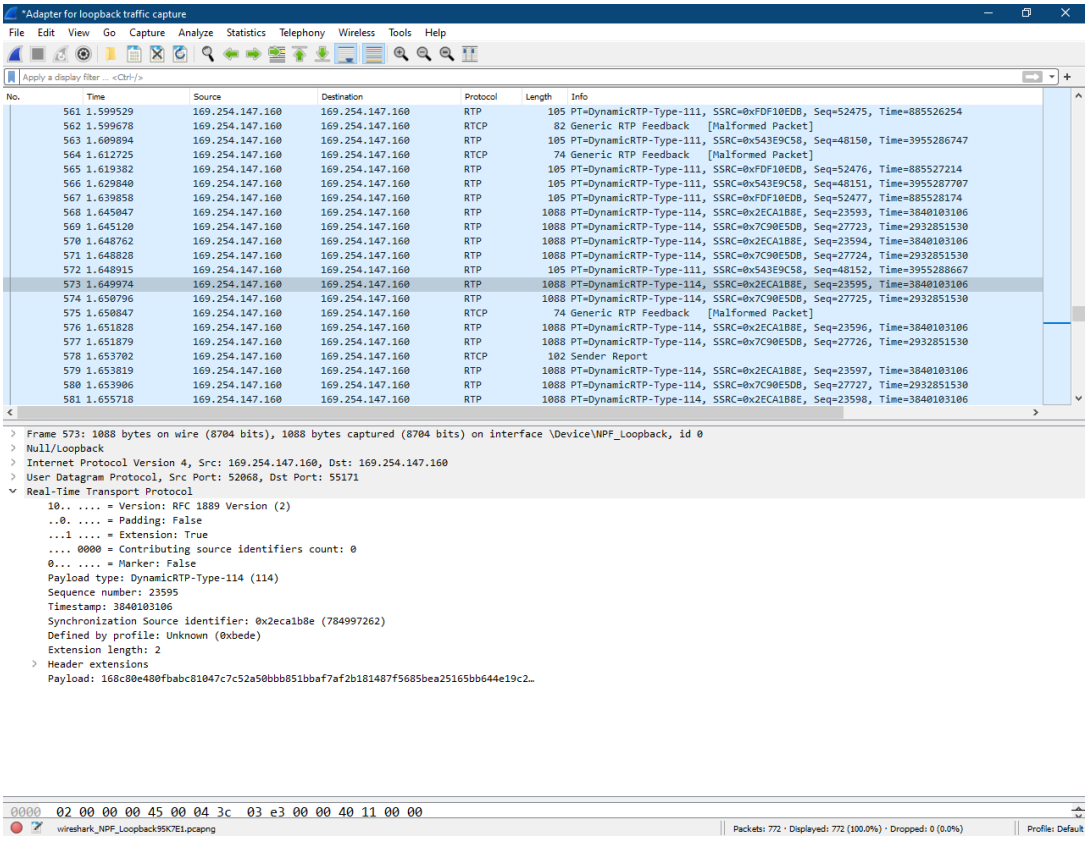
timestamp	21. 12. 2021. 14:45:40
ssrc	2089870811
kind	video
trackId	RTCMediaStreamTrack_receiver_2
transportId	RTCTransport_0_1
codecId	RTCCodec_1_Inbound_96
[codec]	VP8 (96)
mediaType	video
jitter	0.018
packetsLost	0
packetsReceived	36535
[packetsReceived/s]	90.361462741345
bytesReceived	37897243
[bytesReceived_in_bits/s]	746329.4573333595
headerBytesReceived	902620
[headerBytesReceived_in_bits/s]	17863.457167711225
lastPacketReceivedTimestamp	1640094340434
[lastPacketReceivedTimestamp]	21. 12. 2021. 14:45:40
jitterBufferDelay	224.371
[jitterBufferDelay/jitterBufferEmittedCount_in_ms]	61.000000000001364
jitterBufferEmittedCount	4175
framesReceived	4176
[framesReceived/s]	10.040162526816111
[framesReceived-framesDecoded]	0
frameWidth	640
frameHeight	480
framesPerSecond	10
framesDecoded	4176
[framesDecoded/s]	10.040162526816111

c)



Kao što se može vidjeti na slici, koristi se UDP transportni protokol za prijenos audio i video struja. No u 'višem' sloju se koriste RTP i RTCP.

d)



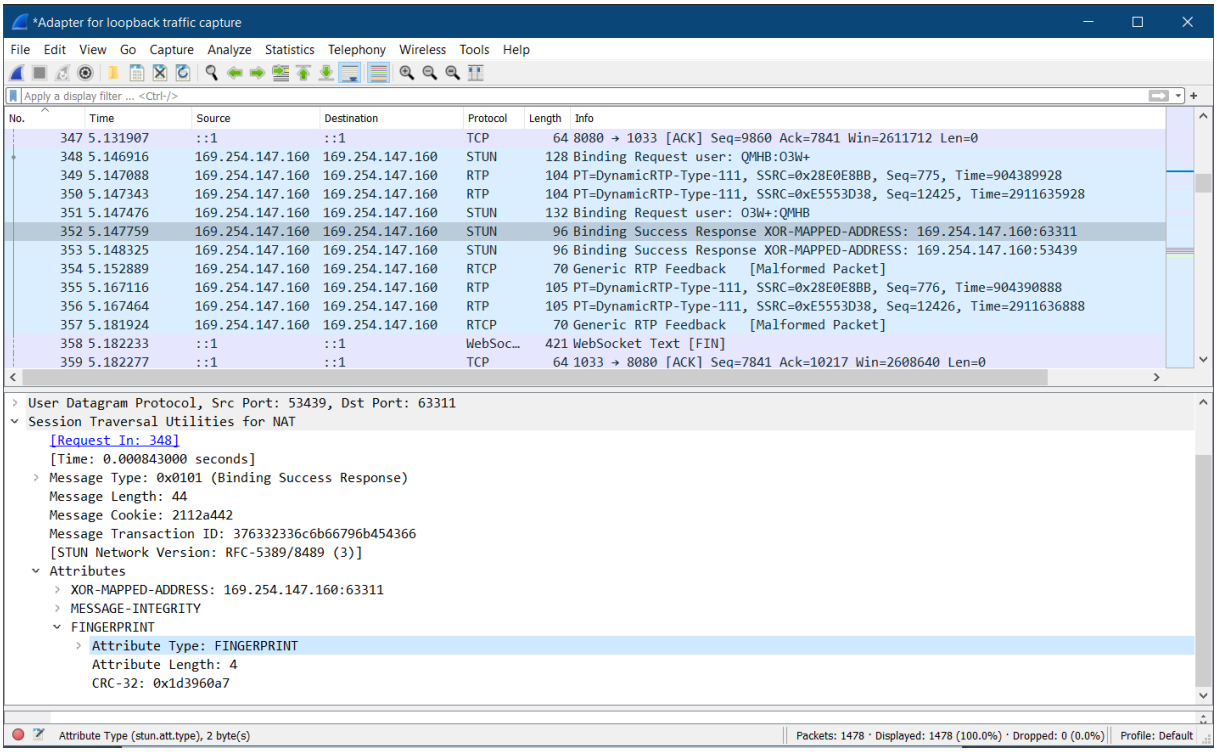
RTP je protokol dizajniran za konzistentni prijenos (najčešće real-time zvukovnih i video) podataka preko IP mreža. RTCP je kontrolni protokol koji koegzistira sa RTP-om te nadzire

kvalitetu usluge i prenosi podatke o sudionicima i raznim parametrima u trenutno pokrenutoj RTP sesiji.

e)

Kao što se može vidjeti iz prethodne slike, ovaj put se podaci ne prenose posredstvom EasyRTC poslužitelja, nego direktno između spojenih entiteta. To se vidi po tome da niti izvorišni niti odredišni port u prijenosu podataka nije 8080 (već su portovi **52068** i **55171**)

f)



Jeidni protokol koji se u ovom slučaju može uočiti (barem koje sam ja uočio) jest STUN. On služi kako bi host (tj. u našem slučaju entitet koji pokreće WebRTC) otkrio nalazi li se iza NAT-a na način da kotaktira nekog predefiniranog poslužitelja koji se nalazi na javnoj IP adresi i pošalje mu upit s koje IP adrese i porta je došao taj upit. Host zna IP adresu koja je dodijeljena njegovom interface-u, te ukoliko se razlikuje od one koje vrati STUN poslužitelj, znači da je host iza NAT-a. Time se ujedno i otkrije javna IP adresa i port. Na slici iznad se može vidjeti slijed razmijenjenih paketa gdje oba browser tab-a pitaju za svoju IP adresu i port, te dobe odgovore.