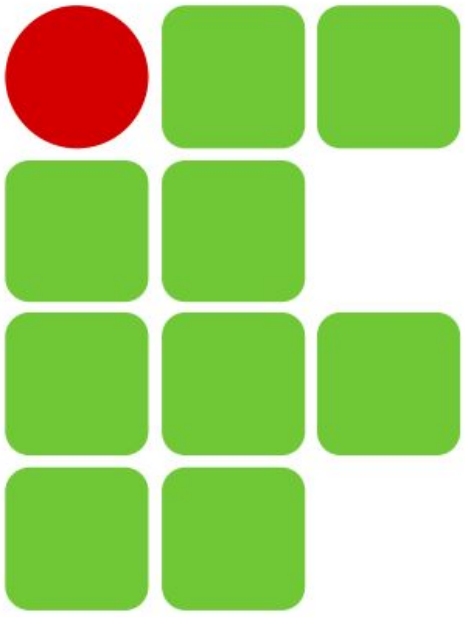

INSTITUTO FEDERAL
SANTA CATARINA
Campus São José

Projeto Final de Sistema Multimídia

Análise de Protocolo em Servidor Easyrtc

Disciplina: Sistemas Multimídia
Professor: Ederson Torresini
Alunos: Guilherme Medeiros e Victor Cesconeto

Julho, 2019

1) Do cenário construído:

Um servidor WebRTC foi criado usando a ferramenta EasyRTC. À este servidor foram conectados dois dispositivos utilizando o navegador Mozilla Firefox 66.0.3.

Uma troca de mídia em áudio foi realizada e os dados dessa troca foram extraídos.

2) Sobre o WebRTC e o EasyRTC

O EasyRTC é uma ferramenta que cria um servidor WebRTC pronto de maneira simplificada e sem muitas configurações. Neste experimento, foi utilizada a ferramenta do EasyRTC que cria um servidor para as comunicações em tempo real. O servidor, neste experimento, foi criado em uma instância de máquina virtual da Google Cloud, servidor este que age como terceiro na comunicação entre duas entidades.

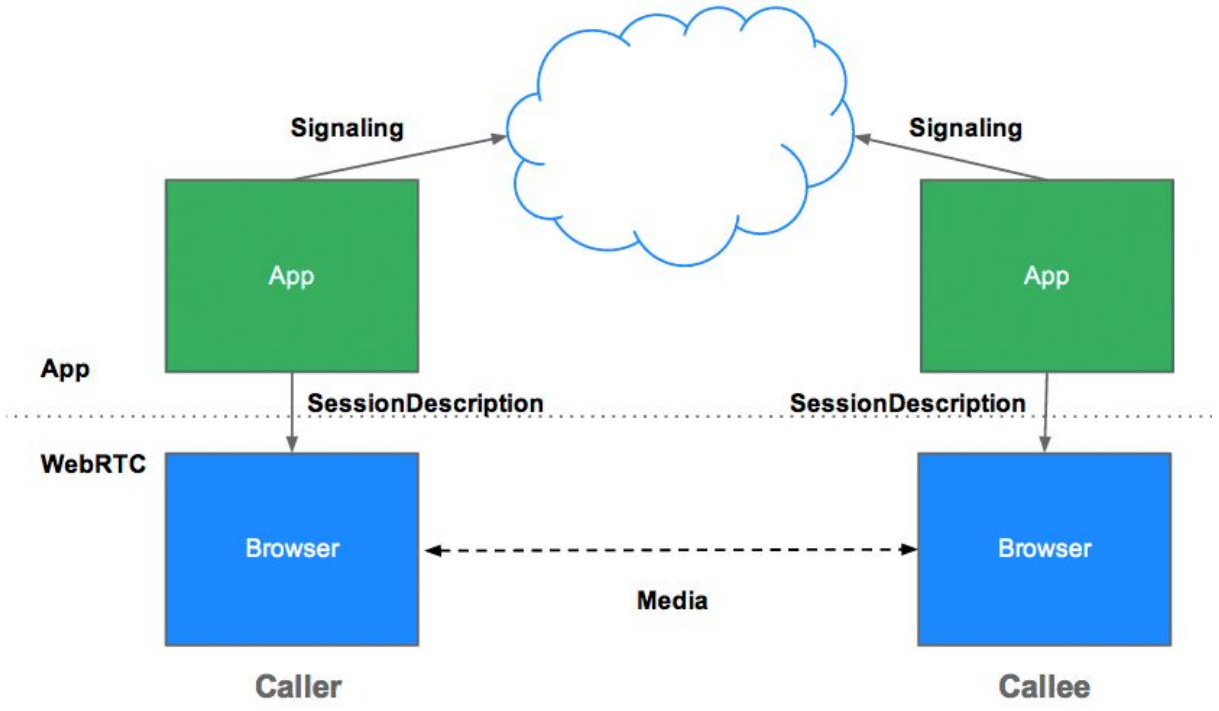

Figura 1 - Esquema representativo de um servidor WebRTC com dois UAs

3) Sinalização e Análise de toda a comunicação:

Registro.

As mensagens seguem, basicamente, dois formatos: GET e POST. A primeira define mensagens que o UA recebe do servidor EasyRTC e a segunda define mensagens que o UA manda ao servidor EasyRTC. Toda a comunicação obedece esse padrão e as mensagens serão analisadas neste escopo.

Figura 2 - Conjunto de GETs e POSTs que constituem a comunicação.

A primeira ação de um USER AGENT em um servidor WebRTC é fazer o registro no servidor. Usando a ferramenta EasyRTC e colocando o servidor online, a conexão é feita totalmente pelo navegador. Quando há a conexão, o UA se conecta ao servidor, começando por pedir uma SID, uma identificação com o método GET.

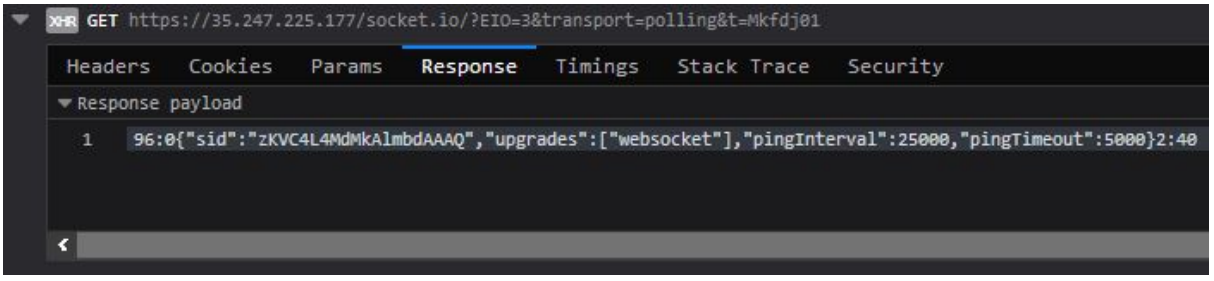

Figura 3 - Identificação com o SID enviada pelo método GET ao User Agent A

Após receber a SID do servidor, ele pede uma autenticação, e recebe um token que o permite utilizar os serviços do servidor, caso algum usuário já esteja conectado, neste token o SID desses usuários são também recebidos pelo UA.

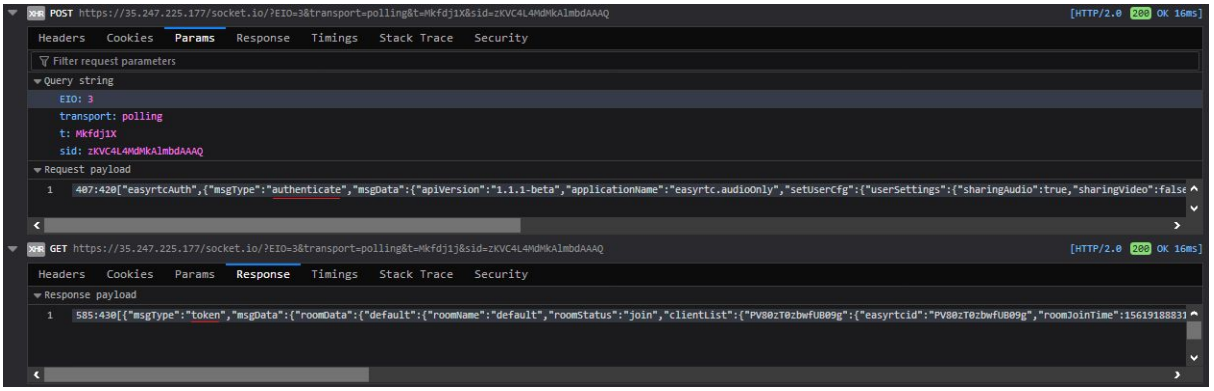

Figura 4 - Transação de autenticação e token

Um POST é feito pelo UA com a flag “stillAlive”, indicando que ele está ativo no servidor, no GET feito em seguida o servidor manda um “ack”, sinalizando que recebeu a mensagem do post anterior do UA.

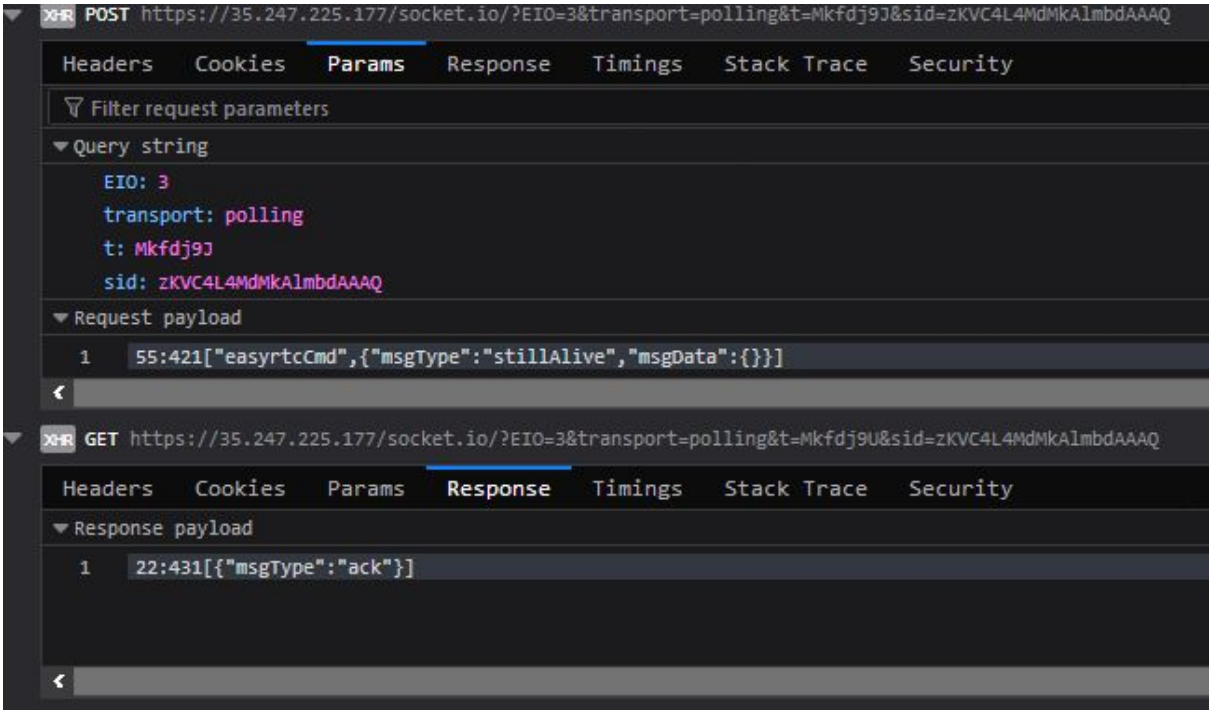

Figura 5 - Transação de StillAlive e Ack, indicando que o UA está ativo

O UA, em seguida, realiza um “setRoomApiField”, registrando o usuário na sala. O próximo GET retorna do servidor com a flag “roomData” com todas as informações da sala (tempo de entrada, quantos e quais UAs estão conectados, nome da sala, entre outros). Por fim mais um GET é realizado, onde o UA recebe do servidor um “ack”. Ao final deste ACK, todo o registro foi feito e o User Agent está pronto para se conectar à outro usuário utilizando o servidor WebRTC.

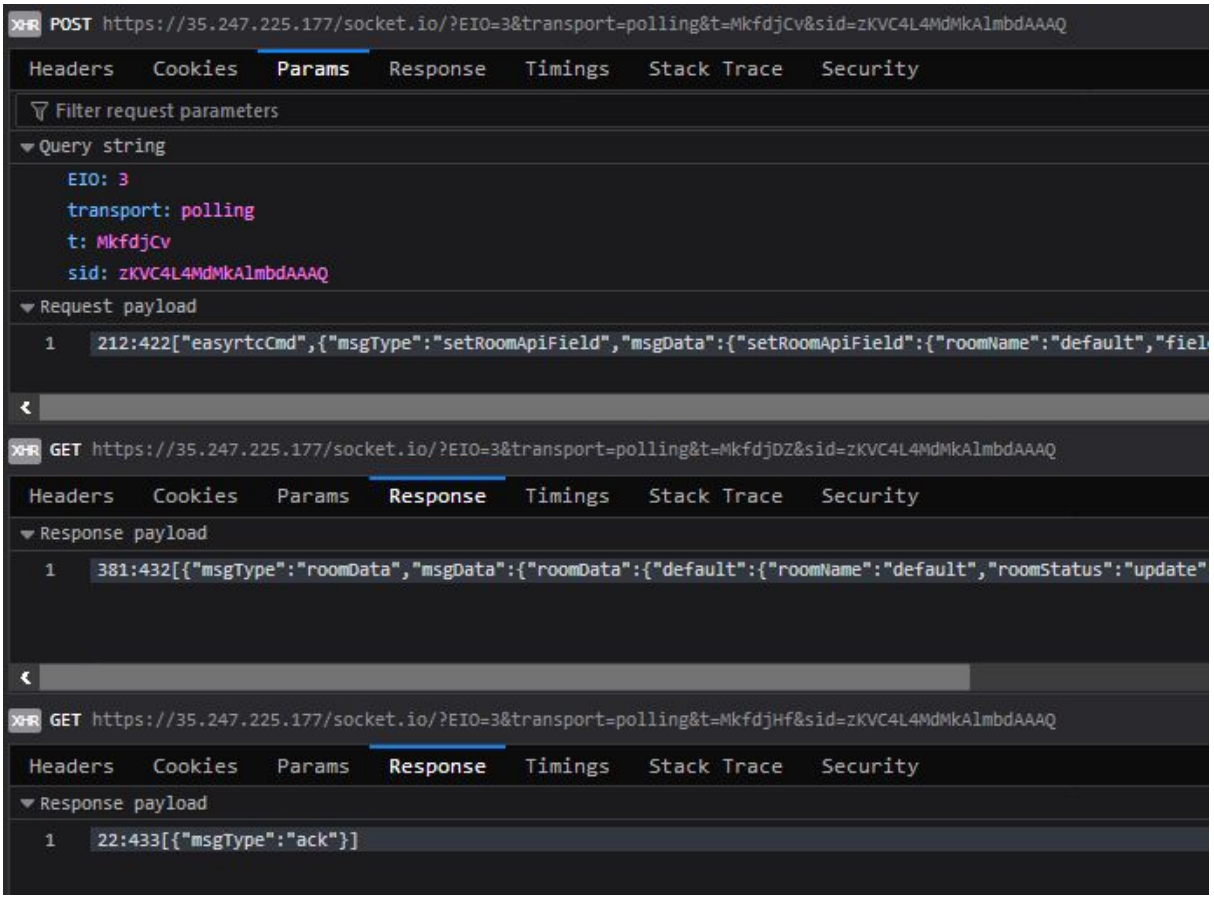

Figura 6 - Transações setRoomApiField, msgData e Ack, onde o usuário pede registro na sala específica e recebe informações sobre a mesma.

Daqui pra frente o usuário fica mandando “stillAlive” com o objetivo de não deixar o timeout do servidor esgotar. Aqui ele só se mantém em standby na sala. Cada POST com “stillAlive” recebe um GET com “ack”.

Estabelecendo Sessão com outro UA:

Um UA “A” faz um POST com a flag “offer”. Neste post, todas as informações sobre codecs e tipo de mídia são mencionadas. Após o post, é feito um GET pelo mesmo UA “A”, que recebe um “ack” do servidor, dizendo que ele recebeu o “offer” anterior.

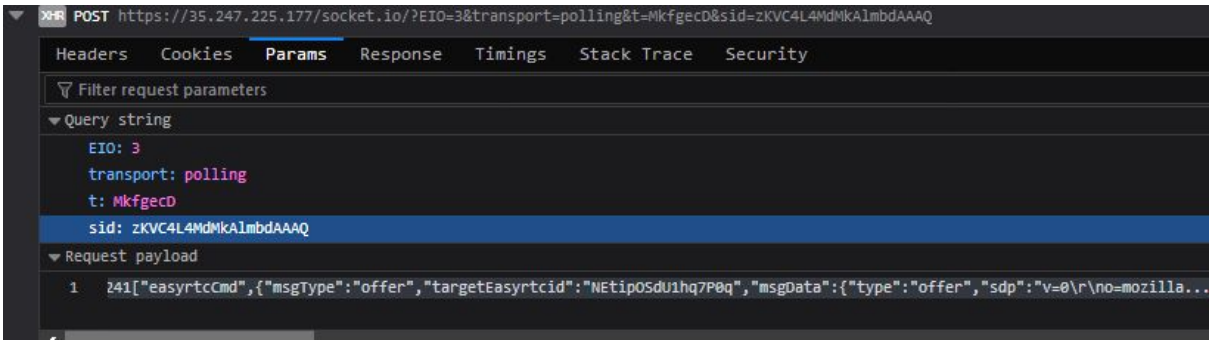

Figura 7 - Offer do User Agent A

Neste ponto é interessante analisar a mensagem que vem como payload:

[HTTP/2.0 200 OK 23ms]

Query string
EIO 3
transport polling
t MkfgecD
sid zKVC4L4MdMkAImbdAAAQ
Request payload

*1143:4241["easyrtcCmd",{"msgType":"offer","targetEasyrtcid":"NEtipOSdU1hq7P0q","msgData":{"type":"offer","**sdp**":"v=0\r\no=mozilla...THIS_IS_SDPARTA-67.0.4 9191153030254923835 0 IN IP4 0.0.0.0\r\ns=-\r\nt=0 0\r\na=fingerprint:sha-256 1B:B6:87:EC:D9:57:F9:6A:73:9E:80:6F:67:DE:C0:07:C4:F5:31:43:85:5A:4D:BC:05:7B:11:91:64:69:62:66\r\na=group:BUNDLE 0\r\na=ice-options:trickle\r\na=msid-semantic:WMS *\r\nm=audio 9 **UDP/TLS/RTP/SAVPF 109 9 0 8 101\r\nc=IN IP4** 0.0.0.0\r\na=sendrecv\r\na=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level\r\na=extmap:2/recvonly urn:ietf:params:rtp-hdrext:csrc-audio-level\r\na=extmap:3 urn:ietf:params:rtp-hdrext:sdes:mid\r\na=fmtp:109 maxplaybackrate=48000;stereo=1;useinbandfec=1\r\na=fmtp:101 **0-15\r\na=ice-pwd:857d94b7aa7f4cb34c13ba9f543ecafa\r\na=ice-ufrag:ba41e18a\r\na=mid:0\r\na=msid:{b4ed4f44-8353-40f3-b5a1-65b4492d54cd}** {f2cd7d4b-6ac8-4475-bdb5-10f81ca28114}\r\na=**rtcp**-mux\r\na=rtpmap:109 opus/48000/2\r\na=rtpmap:9 G722/8000/1\r\na=rtpmap:0 PCMU/8000\r\na=rtpmap:8 PCMA/8000\r\na=rtpmap:101 telephone-event/8000/1\r\na=setup:actpass\r\na=ssrc:2092661310 cname:{827ab836-06fc-4c98-bfa7-2b21c88b3039}\r\n"}}]*

Em vermelho temos a informação sobre o transporte da mensagem, temos o sid com a identificação de A e B, temos o uso do protocolo SDP, a oferta de codecs feita na transação e a informação sobre o servidor ICE.

Em azul vemos o protocolo da camada de transporte (RTC over TCP).

É feito, na sequência, por A, um POST com a flag “stillAlive”, no GET realizado após isso, é recebida a flag “easyRTCmsg”, que o UA responde com um POST contendo a flag “ack”.

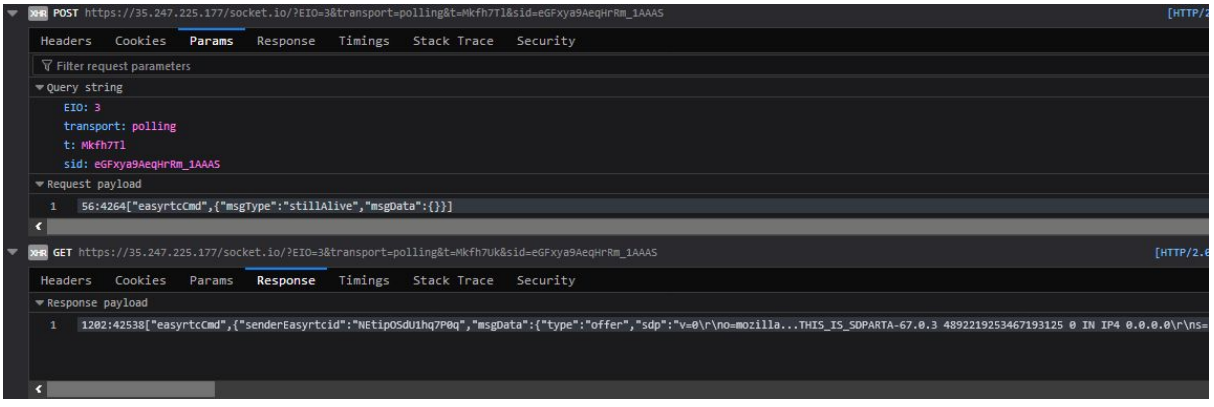

Figura 8 - Transação stillAlive + recepção “EasyRTCmsg”

No GET realizado após isso, vem a flag “answer”, referente à resposta do User Agent “B”, ao qual “A” quer se conectar. Nesta mensagem vem todas as informações do User Agent “B”, como codecs aceitos, formato de mídia, e quais condições do “offer” ofertado por “A” foram aceitas por “B”.

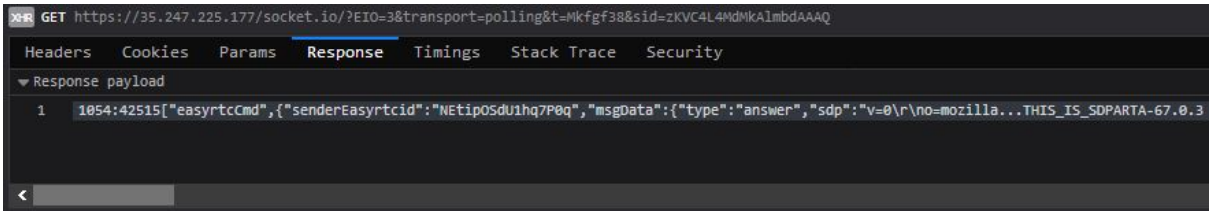

Figura 9 - Transação com flag “answer”

A análise do payload também é interessante neste ponto:

[HTTP/2.0 200 OK 31ms]

1054:42515["easyrtcCmd",{"senderEasyrtcid":"NEtipOSdU1hq7P0q","msgData":{"type":"answer","sdp":"v=0\r\no=mozilla...THIS_IS_SDPARTA-67.0.3 4687195596229414246 0 IN IP4 0.0.0.0\r\ns=-\r\nt=0 0\r\na=fingerprint:sha-256 C0:72:51:EA:EA:38:97:9F:5B:28:B2:52:54:15:2E:4D:72:8A:CF:07:CD:21:8D:DC:E2:68:DE:0B:FC:52:0B:5B\r\na=group:BUNDLE 0\r\na=**ice-options:trickle\r\na=msid-semantic:WMS *\r\nm=audio 9 UDP/TLS/RTP/SAVPF 109 101\r\nc=IN IP4 0.0.0.0\r\na=sendrecv\r\na=extmap:1** urn:ietf:params:rtp-hdrext:ssrc-audio-level\r\na=extmap:3 **urn:ietf:params:rtp-hdrext:sdes:mid\r\na=fmtp:109 maxplaybackrate=48000;stereo=1;useinbandfec=1\r\na=fmtp:101 0-15\r\na=ice-pwd:55aab0438f98bb7038d9c0eb459cabd4\r\na=ice-ufrag:668a8bb3\r\na=mid:0\r\na=msid:{2fb501ee-4a25-4fb7-adae-43d97a781709} {f9e3955f-dcb8-4d3c-8df7-de7d3db9e435}\r\na=rtcp-mux\r\na=rtpmap:109 opus/48000/2\r\na=rtpmap:101** telephone-event/8000/1\r\na=setup:active\r\na=ssrc:67158520 cname:{2ae7edba-bd0e-47e8-b050-a9ff84182770}\r\n"},"easyrtcid":"PV80zT0zbwfUB09g","msgType":"answer","serverTime":1561919653165}]257:42516["easyrtcCmd",{"senderEasyrtcid":"NEtipOSdU1hq7P0q","msgData":{"type":**"candidate",**"label":0,"id":"0","candidate":"candidate:0 1 **UDP 2122252543 10.0.0.100 46503** typ host"},"easyrtcid":"PV80zT0zbwfUB09g","msgType":"candidate","serverTime":1561919653165}]273:42517["easyrtcCmd",{"senderEasyrtcid":"NEtipOSdU1hq7P0q","msgData":{"type":"candidate","label":0,"id":"0","candidate":"candidate:6 1 TCP 2105508095 10.0.0.100 63919 typ host tcptype passive"},"easyrtcid":"PV80zT0zbwfUB09g","msgType":"candidate","serverTime":1561919653166}]268:42518["easyrtcCmd",{"senderEasyrtcid":"NEtipOSdU1hq7P0q","msgData":{"type":"candidate","label":0,"id":"0","candidate":"candidate:6 1 TCP 2105524479 10.0.0.100 9 typ host tcptype active"},"easyrtcid":"PV80zT0zbwfUB09g","msgType":"candidate","serverTime":1561919653166}]

Em vermelho vemos as informações do servidor ICE, a oferta e aceitação dos codecs que serão utilizados.

Em azul vemos o protocolo a anunciação do método “candidate” (explicado logo abaixo), e o protocolo da camada de transporte (UDP).

A próxima mensagem é um POST com a flag “candidate”. Este post tem todas as informações do endereço do User Agent A. IP, porta, e protocolo de transporte utilizado. Em seguida A faz um GET e recebe uma mensagem com flag “candidate” com todas as informações de endereço de B. IP, porta e protocolo da camada de transporte.

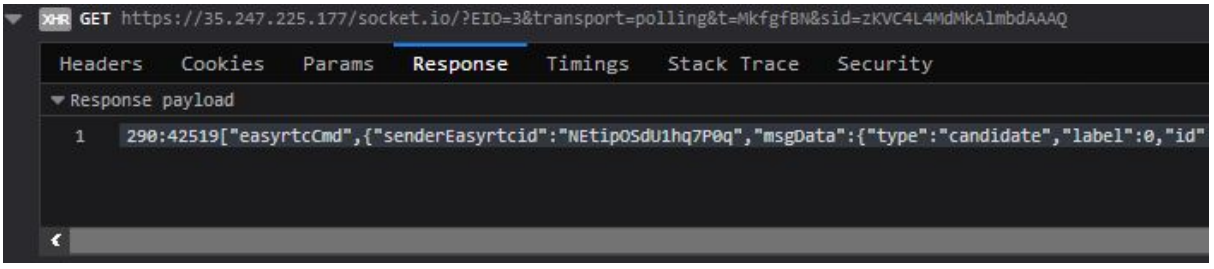

Figura 10 - Get com flag “candidate”

O GET seguinte retorna uma série de “ack”, referentes a cada endereço mandado no POST anterior.

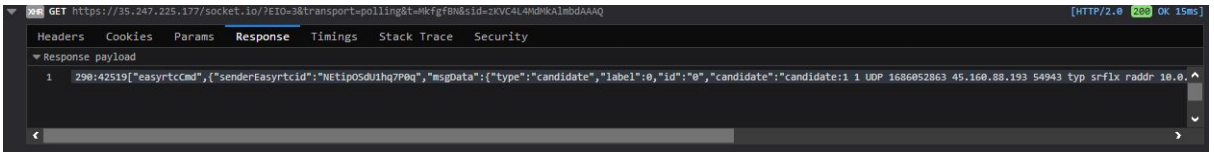

Figura 11 - GET com série de “ack”.

Em seguida é feito um POST, contendo “ack”.

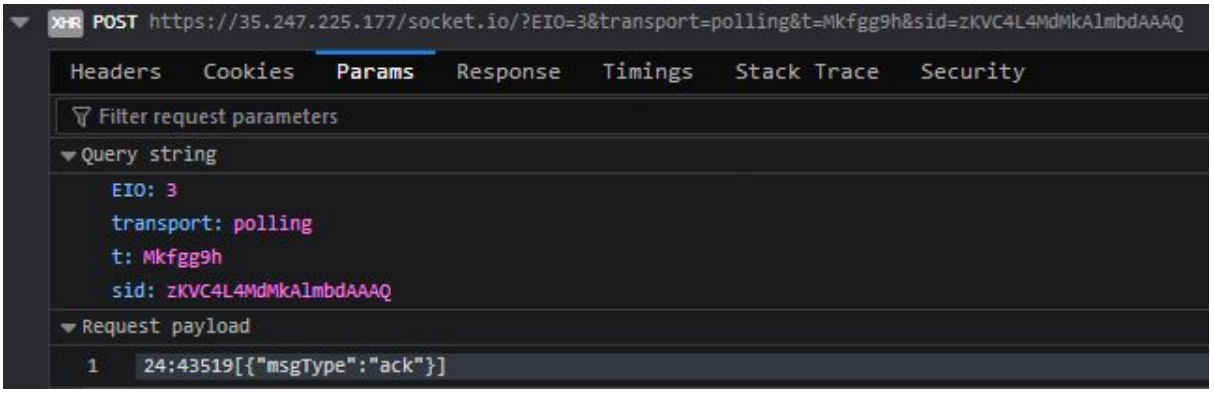

Figura 12 - Último POST contendo “ack”.

Após esse ponto, tudo está certo entre os dois UAs e eles entram em estado de comunicação, aqui ficam sendo feitos POSTs com a flag “stillAlive” com objetivo de dizer que ainda estão em sessão, e no GET após isso é recebido um “ack”. Até acontecer a mensagem com a flag “hangup”, onde a comunicação se desfaz mas o registro com o servidor se mantém.

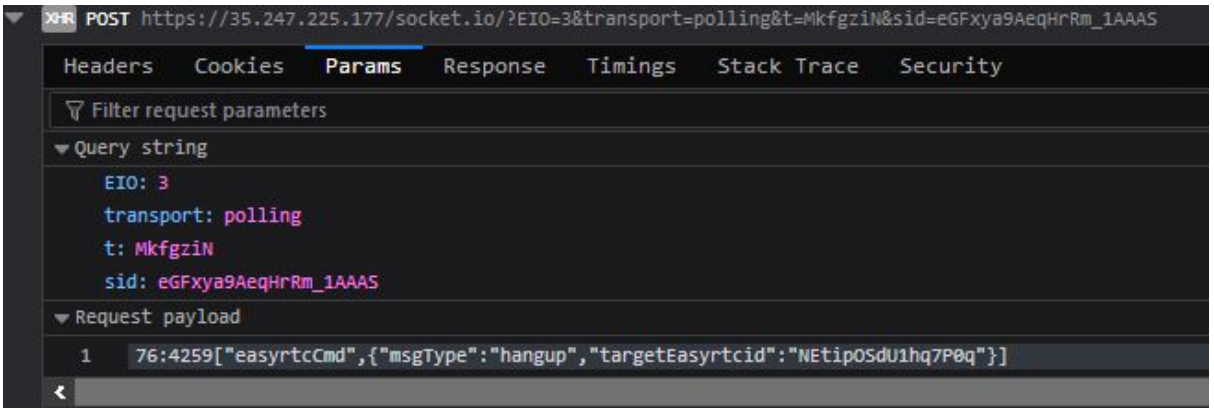

Figura 12 - POST com flag “hangup” sinalizando o fim da conexão.

4) Protocolos de transporte utilizados:

Depende da oferta de cada UA, mas podem ser utilizados UDP/RTP e RTCP. No experimento acima, o User Agent que fez a análise usa RTCP (em um firefox rodando em um microsoft Windows), o outro User Agent usa UDP (em um firefox rodando em um Android).

5) Mecanismos de Controle de Atraso e Jitter:

Há, em cada requisição GET e POST, um campo com o atraso em relação à mensagem anterior, dado em milisegundos. Aqui é feito o controle de tempo de toda a transação.

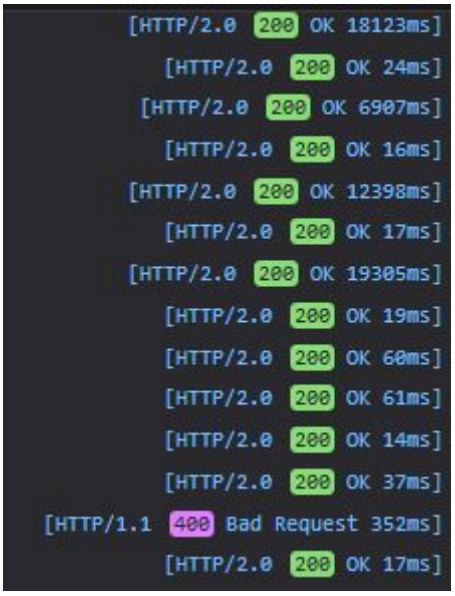

Figura 13 - Método de controle do jitter e atraso.

6) Transações e Diálogos:

Cada GET/POST é uma transação neste tipo de conexão. O conjunto de transações com mesma identificação de SID (sid=*sid_do_UA*) é um diálogo.

7) Escolha de Caminho:

O protocolo de escolha de caminho utilizado vai depender de cada conexão. No caso do experimento, foi utilizado um servidor ICE em ambos UAs e a escolha de caminho segue o protocolo de rede até o outro ponto. Isto pode ser visto em ambas as análises de Payload realizadas.

8) Referências:

WebRTC:
<https://webrtc.org/start/>, acessado em 29/06/2019 às 16:23.

Eventos em EasyRTC:
<https://easyrtc.com/docs/easyrtc_server_events.php>, acessado em 21/06 às 17:35.

Documentação de client para EasyRTC:
<https://easyrtc.com/docs/easyrtc_client_tutorial.php>, acessado em 23/06 às 17:50

Código fonte da implementação do EasyRTC:
<https://easyrtc.com/docs/client-api/easyrtc_int.js.php#line3077>, acessado em 29/06/2019 às 19:30