Internet engineering mini project report
Hossein Khalili, Bardia Heydarinejad, Erfun Ghalibaf
January 9, 2018

# فاز سوم ( استفاده از ICE )

## Deploy on a server and configure system to work behind NATs

برنامه نهایی برروی سرور Deploy شده و از آدرس http://hossein-khalili.me قابل دسترسی میباشد. همچنین فایل های برنامه برروی github و در آدرس https://github.com/hossein-khalili/simplevideochat قرار دارند

همانطور که در مستندات دو فاز قبل پروژه توضیح داده شد. یک سرویس videoCall بسیار ساده با قابلیت ایجاد room تولید کردیم. در این فاز قصد داریم تا این برنامه را روی یک سرور واقعی deploy کرده و ضمناً با استفاده از ICE امکان اتصال دو کاربر از دو شبکه ( NAT ) مختلف را فراهم آوریم. برای اینکار در فایل main.js تعریف متغیر pc که مخفف Peer Connection است را به شکل زیر تغییر میدهیم.

```
pc = new RTCPeerConnection({
    iceServers: [     // Information about ICE servers
{
        urls: "turn:138.197.154.24:3478",
        username: "test",  // A TURN server
        credential: "test"      }
    ]
```

با اضافه کردن تنظیمات iceServers به RTCPeerConnection در واقع به برنامه میگوییم که سرور TURN در چه آدرسی واقع شده است و تنظیمات اتصال به آن چیست. در مثال فوق این سرور در آدرس ۱۳۸.۱۹۷.۱۵۴.۲۴ و پورت ۳۴۷۸ قرار دارد و نام کاربری آن test و رمز عبور نیز test میباشد.

توضیحات : این TURN server را با استفاده از coturn برروی vps شخصی ایجاد کردیم.

برای تولید آن بر بستر ubuntu ابتدا بایست با دستور sudo apt-get install coturn این برنامه را نصب کرد. سپس با تغییر مقادیر فایل /etc/turnserver.conf تنظیمات سرور را شخصی سازی میکنیم. در نهایت با دستور turnserver -o -v -u test:test آن را اجرا مینماییم. پرچم o- باعث اجرای این برنامه در پس زمینه سیستم میشود (daemon) همچنین پرچم v- به برنامه میگوید تا اطلاعات (log) را در فایلی ذخیره کند و پرچم u- مشخص کننده نام کاربری و رمزعبور است.

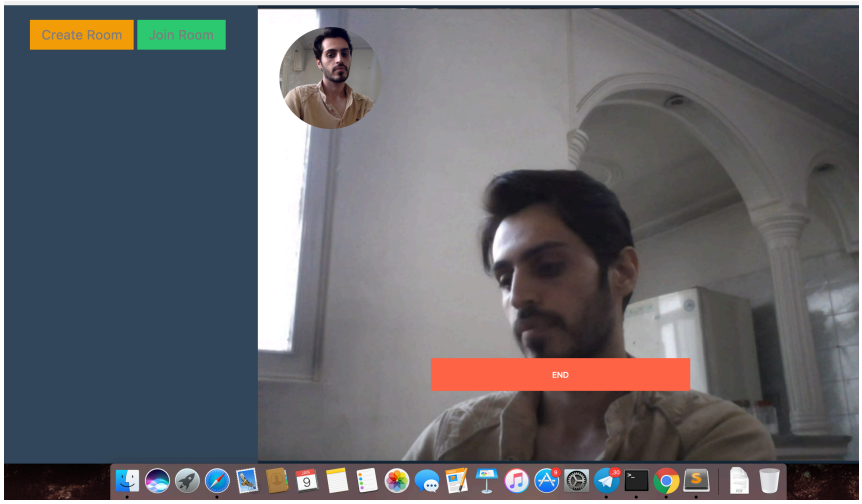
حال باید به برنامه بگوییم که هنگامی که icecandidate وجود داشت چه کاری باید انجام دهد. برای این منظور تابع onicecandidate متغیر pc ( RTC Peer Connection ) را برابر تابع handleIceCandidate قرار دادیم و آن را اینگونه تعریف نمودیم.

```
function handleIceCandidate(event) {
  console.log('handleIceCandidate event: ', event);
  if (event.candidate) {
    sendMessage({
      type: 'candidate',
      label: event.candidate.sdpMLineIndex,
      id: event.candidate.sdpMid,
      candidate: event.candidate.candidate});
  } else {
    console.log('End of candidates.');
  }
}
```

این تابع هنگامی که یک ice candidate ایجاد میشود (یعنی دستگاه پشت NAT قرار داشته باشد) پیغامی مبنی بر تقاضای client برای ایجاد یک ارتباط مبتنی بر ICE را به سرور ارسال میکند. این پیغام حاوی شماره خط sdp کاربر و نوع دیتایی که میفرستد (id) و اطلاعات candidate مورد نظر شامل پروتوکل (tcp/udp) و شماره نوع آن ( type number ) است.

سرور پس از دریافت این پیام آن را برای سمت دیگر ارتباط ارسال میکند. پس از دریافت این پیام در سمت دیگر شرط زیر در تابع socket.on('message') اجرا میشود.

```
if (message.type === 'candidate' && isStarted) {
  console.log("This Peer Got ICE Candidate.");
  var candidate = new RTCIceCandidate({
    sdpMLineIndex: message.label,
    candidate: message.candidate
  });
  pc.addIceCandidate(candidate);
```

در اینجا برنامه یک ارتباط از نوع RTCIceCandidate ایجاد کرده و آن را به عنوان پارامتر به تابع addIceCandidate میدهد و تابع را فراخوانی میکند. این تابع candidate دریافت شده را به تعریف RTCPeerConnection ها اضافه میکند که مشخص کننده آخرین وضعیت ارتباط است. در طول برنامه تعداد زیادی candidate دریافت میشود که برنامه بر اساس آنها اقدام به ایجاد ارتباط بین دوطرف میکند. پس از این مرحله ارتباط بین دو کاربر ایجاد شده و دو طرف شروع به ارسال میدیای مورد نظر میکنند.

تصویر مورد نظر مربوط به ارتباط دو لپتاپ با استفاده از دو ISP مختلف است.