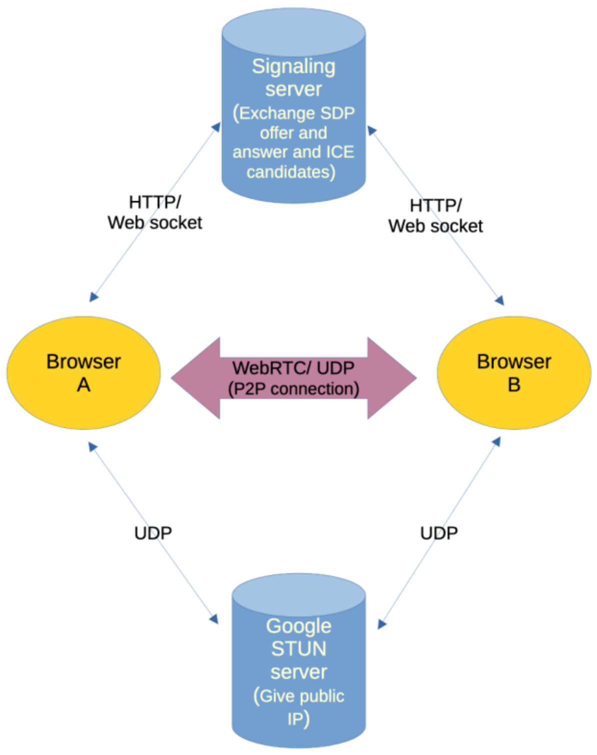
System Goals

We had one main goal for our system – to develop an application that can be used for one to one live video chat using WebRTC with only basic functionality and very simple UI.

System architecture


Signaling server (Exchange SDP offer and answer and ICE candidates)
HTTP/ Web socket
HTTP/ Web socket
Browser A
WebRTC/ UDP (P2P connection)
Browser B
UDP
UDP
Google STUN server (Give public IP)

Our system consists of 4 modules: Signaling server, STUN server and the 2 peers that communicate via browser. WebRTC connection establishment requires a server to exchange connection messages between peers. Such a server is called the signaling server. The signaling server informs the peers about incoming calls and answers, by relaying the SDP offer and answer and ICE candidate messages. Signaling server is also used to reject the call when there is no WebRTC channel between the peers.

The peers need to exchange their public IP address within the ICE candidate message, since they cannot use their local IP addresses. Peers ask for their public IP address from the public STUN server. We are not using TURN servers since it is hard to implement, and we wanted to focus on simpler situation.

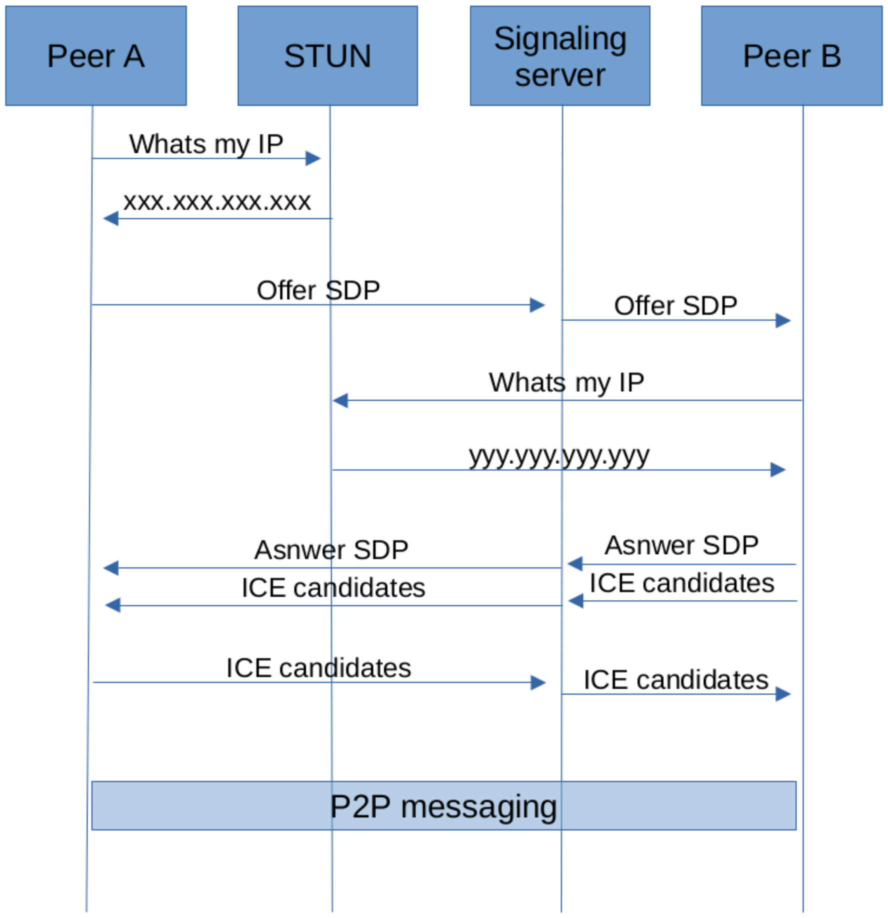
Once the peers have enough information about the other peer's location and what sort of data they want to send, they can establish a P2P connection.

Components / Module description including the interfaces exposed between the modules

We have two primary modules, the first one is authentication module which primarily handles the user account creation and login. The interfaces associated with this module is login page and user registration pages. Similarly, the second one is chat module which handles the RTCPeerconnection from the server side. The interface associated with the chat module is user dashboard. Besides this we have used JS component for making Peer connection offer and answer from the client side.

Communication channel between the modules

The peers establish their connection to the signaling server via HTTPS, and after logging in, they communicate through web sockets. The peers use STUN protocol to get the public IP address response from STUN server. STUN protocol runs over UDP transport protocol, similarly as the WebRTC protocol, that the peers use to communicate directly with each other. All the protocols used in this project are secure and the data is encrypted.



User (browser) uses signaling server to contact other user and transfer needed data to request p2p connection (send SDP offer). SDP offer [1] contains information of what kind of media is being sent and its format, transfer protocol being used, IP addresses and ports to be used. Other user gets the call request (Offer) and respond by sending (SDP answer along with ICE candidates). SDP answer [1] contains similar information as the offer but from the callable user. Interactive Connectivity Establishment (ICE) candidates describe the protocols and routing needed for WebRTC connection. For generating ICE candidates, we need the public IP information from STUN server.

Pros and cons of the open-source components/modules used for developing the system, and the modules/components you have built

Pros of the open-source components/modules used for developing the system and the modules/components built:

- They already have handling the WebRTC connection establishments.
- We do not need to completely develop from scratch. We only need to call their endpoint or predefined functions wherever it is needed.
- Due to open-source module, the development duration of the project was comparatively faster.

Cons of the open-source components/modules used for developing the system and the modules/components built:

- One of the pre-defined functions used in creating peer connections was already depreciated. Hence we had to find an alternative to that function, because many version conflicts may arise.
- Any support from the publisher were not available for the implementation of that code in the systems. Thus, in the initial we struggled a lot.
- We cannot completely rely on open-source components, they do not always perform well as expected.

Which of the fallacies of the distributed system does your system violate, and how

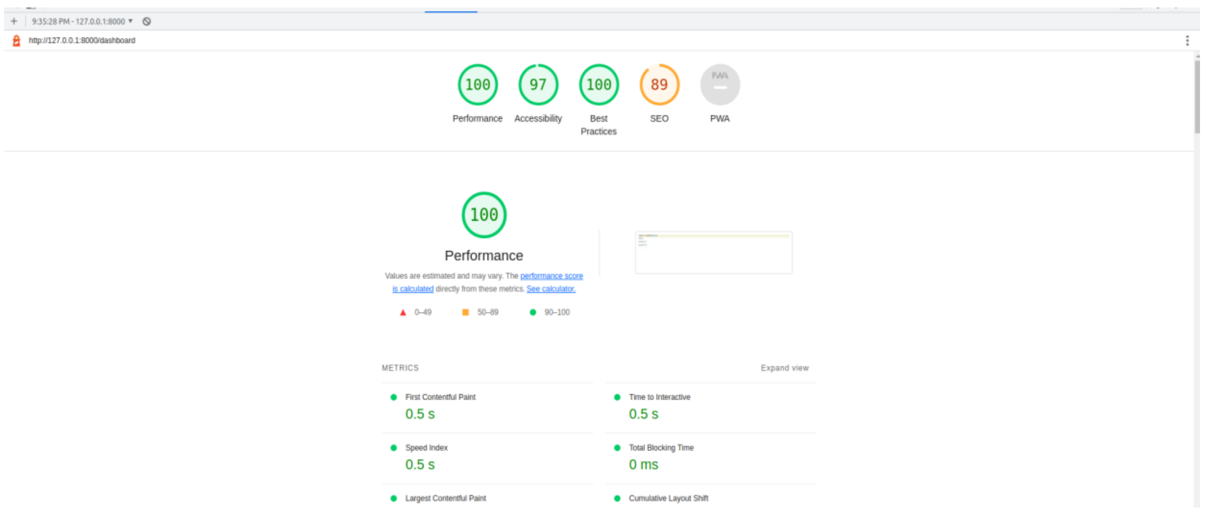
The system violated the network security fallacy of the distributed system fallacies, in the following ways: everything is handled through third party channels which in this case means Django Daphne.

What needs to be added to your system be used to be integrated/extended by another system

To be integrated/extended by another system the following additions to our system should be done: our signaling server should be public such that users in distinct networks can create accounts and contact each other. We need to configure external STUN and TURN servers so that systems from different networks can communicate. Deploy the application in Docker container and fix bugs.

Evaluation. Methodology used for evaluating the system performance, and the key results

We tried multiple methodologies to evaluate the performance of our system. Firstly, we did some manual testing by logging and checking the response and found that the overall performance is as expected. Secondly, we used the third-party application to analyze the performance, and found out that the overall performance of our system is pretty good.



9:35:28 PM - 127.0.0.1:8000

http://127.0.0.1:8000/dashboard

100	97	100	89	PWA
Performance	Accessibility	Best Practices	SEO	PWA

100

Performance

Values are estimated and may vary. The performance score is calculated directly from these metrics. See calculator.

▲ 0–49 ■ 50–89 ● 90–100

METRICS Expand view

● First Contentful Paint 0.5 s	● Time to Interactive 0.5 s
● Speed Index 0.5 s	● Total Blocking Time 0 ms
● Largest Contentful Paint	● Cumulative Layout Shift

Avenues for future work

To dive deeper into this subject, one could extend the application to multi person calls, add a chat functionality, mute button, screen sharing, video stream hiding and as mentioned previously have a public server. At the moment when the connection is closed the browser still has access to your camera; so to close your local video stream on browser would be a good addition. Currently we are sending synchronous requests to the server – this could be changed to asynchronous so that we can send parallel requests.

[1] An Offer/Answer Model with the Session Description Protocol (SDP) 'https://www.ietf.org/rfc/rfc3264.txt'