Smilegate

Development Camp 2025

WebRTC Seminar

BBeBig 이지형

Contents

WebRTC 소개

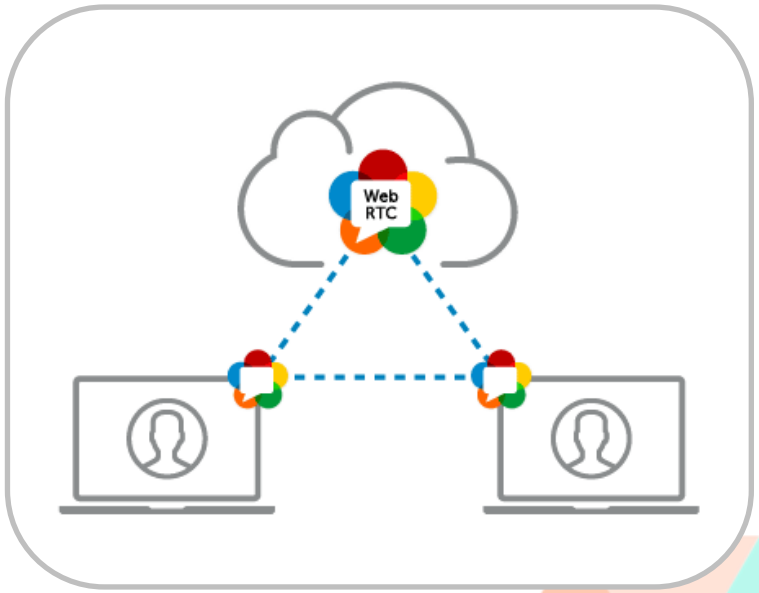
- WebRTC란?
- WebRTC의 작동 원리

WebRTC란?

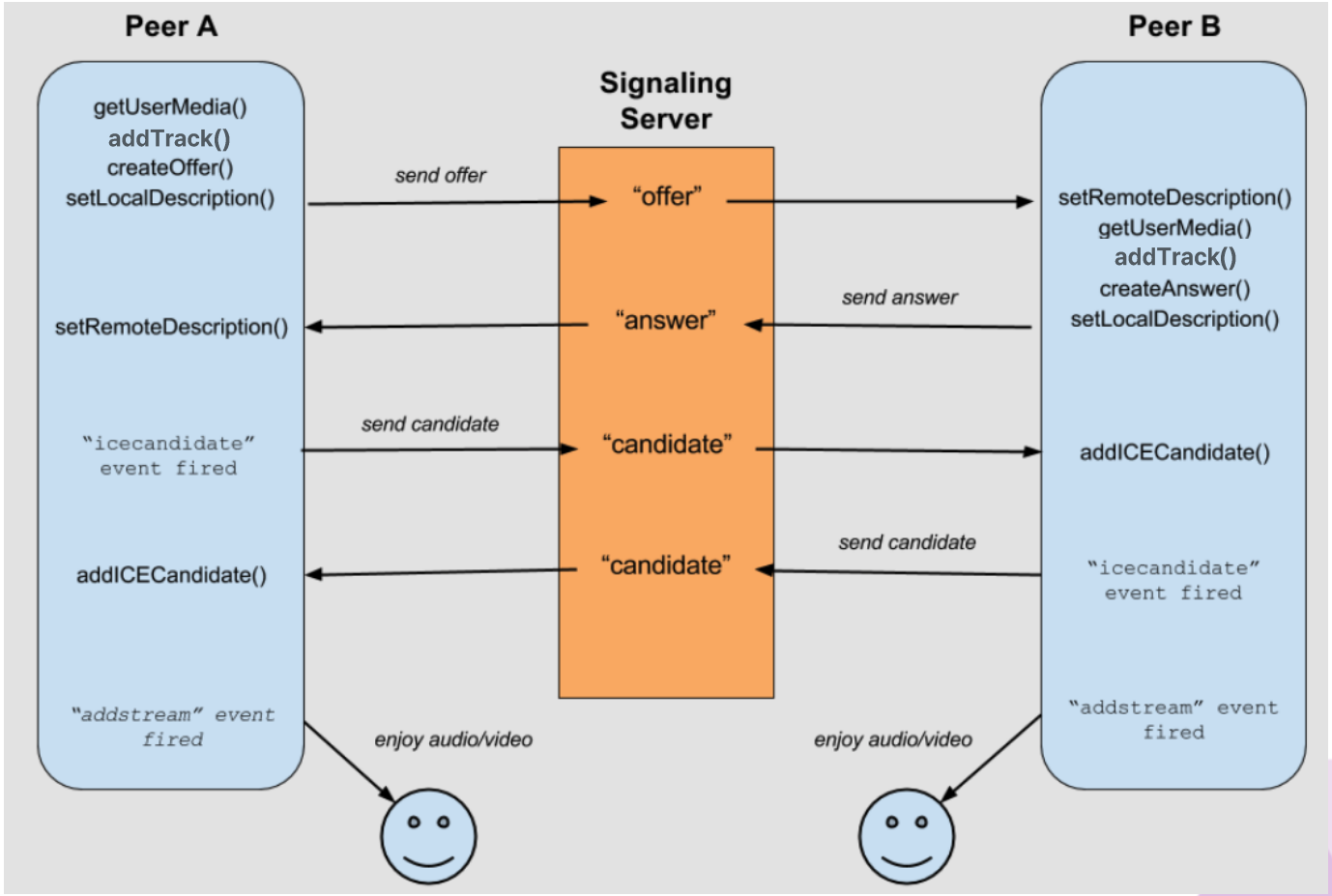
- WebRTC란 웹 브라우저 환경 및 Android, IOS에서도 사용 가능한 비디오, 음성 및 일반 데이터가 ***피어간에 실시간으로 전송**되도록 지원하는 **오픈 소스** 입니다.

  *피어(peer)는 클라이언트를 의미합니다.

- 웹 표준으로 구현되어 있으며, 모든 주요 브라우저에서 일반 JavaScript API를 제공합니다.



WebRTC의 작동 원리

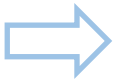


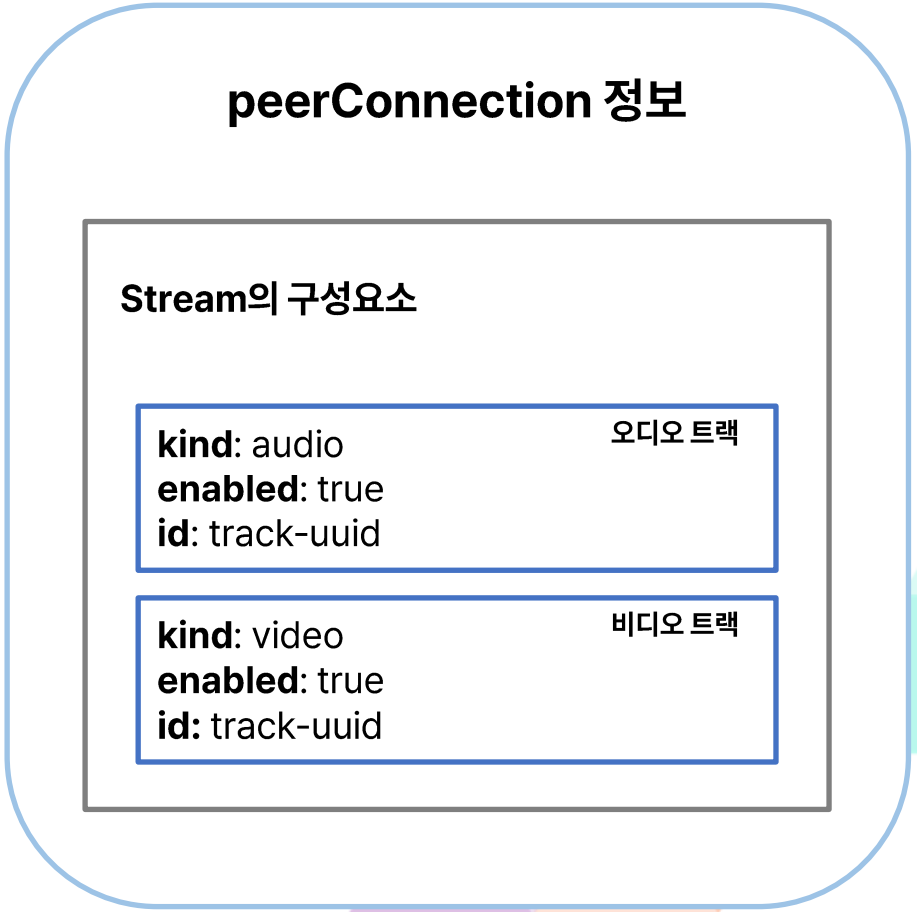
작동 원리 – getUserMedia

- 사용자의 미디어 정보를 불러오는 역할을 합니다.

```
const peerConnection = new RTCPeerConnection();
navigator.mediaDevices.getUserMedia({ video: true, audio: true })
  .then(stream => {
    stream.getTracks().forEach(track => {
      peerConnection.addTrack(track, stream); // 개별 트랙 추가
    });
  });
```

- 불러온 정보는 peerConnection에 개별 트랙으로 추가됩니다.



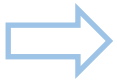


작동 원리 – offer 단계

SDP(Session Description Protocol)는 **미디어 데이터를 직접 전달하는 것이 아니라, 콘텐츠와 연결 조건에 대한 메타데이터를 설명**하는 역할을 합니다.

이 프로토콜은 **디바이스 간 미디어를 공유하기 위한 연결을 설정**하는 데 필요한 정보를 정의하며, 이를 통해 피어 간에 미디어(오디오, 비디오, 데이터)를 효율적으로 교환할 수 있도록 돕습니다.

*이전에 설정해둔 stream 데이터는 해당 단계에서 옮겨지지 않습니다.



SDP 구성 요소

- **미디어 정보**: 오디오와 비디오 코덱, 샘플링 속도, 해상도
- **네트워크 정보**: 피어 간 연결을 위한 IP, 포트 정보
- **암호화 정보**: DTLS 인증서 지문 및 설정
- **데이터 채널**: 텍스트나 파일 전송을 위한 데이터 채널 설정정보

작동 원리 – answer 단계

- 상대방에게서 받은 정보를 setRemoteDescription을 이용하여 저장하여 관리합니다.

```
// 상대방의 SDP Offer를 수신
signalingServer.onmessage = (message) => {
  if (message.type === "offer") {
    // Remote SDP 설정
    peerConnection.setRemoteDescription(new RTCSessionDescription(message.sdp))
      .then(() => {
        console.log("Remote SDP Offer set.");
        // SDP Answer 생성
        return peerConnection.createAnswer();
      })
      .then(answer => {
        // Local SDP 설정
        return peerConnection.setLocalDescription(answer);
      })
      .then(() => {
        // Signaling 서버로 Answer 전송
        signalingServer.send({ type: "answer", sdp: peerConnection.localDescription });
      })
      .catch(error => {
        console.error("Error during SDP negotiation:", error);
      });
  }
};
```

작동 원리 – candidate 단계

- Candidate 교환을 통해 P2P 간의 최적의 네트워크 경로를 설정합니다.
- 앞에서 받아온 setLocalDescription함수가 호출된 직후 Candidate를 동적으로 생성합니다.
- 이후 ICE Agent가 로컬 IP, 공인 IP(STUN), TURN 경로를 포함한 모든 Candidate를 테스트하고 최적의 경로를 선택합니다.

```
// candidate 정보를 서버로 전송합니다.
peerConnection.onicecandidate = (event) => {
  if (event.candidate) {
    signalingServer.send({ type: "candidate", candidate: event.candidate });
  }
};
```

```
// 받아온 candidate 경로를 peerConnection에 추가합니다.
signalingServer.onmessage = (message) => {
  if (message.type === "candidate") {
    peerConnection.addIceCandidate(new RTCIceCandidate(message.candidate));
  }
};
```

작동 원리 – candidate 단계

- Candidate 교환을 통해 P2P 간의 최적의 네트워크 경로를 설정합니다.
- 앞에서 받아온 setLocalDescription함수가 호출된 직후 Candidate를 동적으로 생성합니다.
- 이후 ICE Agent가 로컬 IP, 공인 IP(STUN), TURN 경로를 포함한 모든 Candidate를 테스트하고 최적의 경로를 선택합니다.

```
// candidate 정보를 서버로 전송합니다.
peerConnection.onicecandidate = (event) => {
  if (event.candidate) {
    signalingServer.send({ type: "candidate", candidate: event.candidate });
  }
};
```

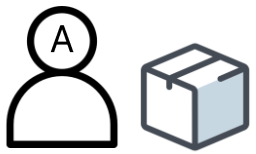
```
// 받아온 candidate 경로를 peerConnection에 추가합니다.
signalingServer.onmessage = (message) => {
  if (message.type === "candidate") {
    peerConnection.addIceCandidate(new RTCIceCandidate(message.candidate));
  }
};
```

작동 원리 – 렌더링 단계

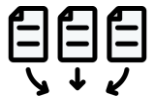
- 상대방의 트랙은 ontrack을 통해 받아올 수 있습니다.
- 해당 화면을 브라우저에 띄워줌으로써 상대방의 영상을 볼 수 있습니다.

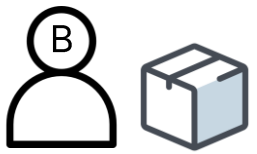
```
// 상대방의 트랙을 수신할 때 실행
peerConnection.ontrack = (event) => {
  console.log(`Received track from ${userId}:`, event.track);

  // 비디오 요소 생성
  const remoteVideo = document.createElement("video");
  remoteVideo.srcObject = event.streams[0]; // 수신한 MediaStream 설정
  remoteVideo.autoplay = true;              // 자동 재생
  remoteVideo.playsInline = true;           // iOS 대응

  // 상대방 ID를 식별할 수 있도록 데이터 속성 추가
  remoteVideo.dataset.userId = userId;

  // 비디오를 화면에 추가
  remoteContainer.appendChild(remoteVideo);
};
```

작동 원리 – 다시보기



peerConnection





peerConnection

작동 원리 – 다시보기

작동 원리 – 다시보기

작동 원리 – 다시보기

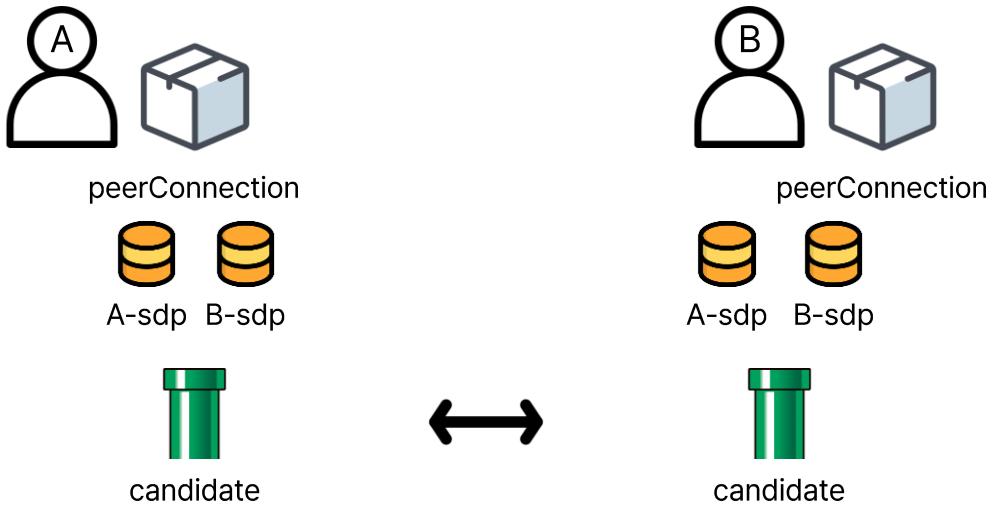
작동 원리 – 다시보기

작동 원리 – 다시보기

작동 원리 – 다시보기

작동 원리 – 다시보기

작동 원리 – 다시보기


A
peerConnection
A-sdp
B-sdp
candidate
B
peerConnection
A-sdp
B-sdp
candidate

주요 구성 요소

- RTCPeerConnection
- MediaStream
- RTCDDataChannel
- WebRTC의 사용 사례

RTCPeerConenction

- peer (클라이언트) 간의 미디어와 데이터를 전송하기 위한 WebRTC의 핵심 API
- 기능
  - 미디어 교환 (Signal Processing)
  - 보안 (DTLS/SRTP)
  - 네트워크 탐색 (NAT Traversal)
  - 대역폭 추정 및 조정

MediaStream

- WebRTC에서 미디어 장치 (카메라, 마이크 등)에 접근하고 데이터를 전송하기 위한 객체입니다.
- 미디어 스트림은 **navigator.mediaDevices API** 를 통해 가져오며, **PeerConnection**에 전달해 전송합니다.

MediaStream 주요 기능

- **사용자 미디어 장치 접근**
  - navigator.mediaDevices.getUserMedia 함수를 통해 카메라와 마이크 데이터를 가져올 수 있습니다.
  - HTTPS 환경에서만 동작하며, 로컬 개발 환경(localhost)에서는 예외적으로 허용됩니다.

```
navigator.mediaDevices.getUserMedia({ video: true, audio: true })
  .then(stream => {
    console.log("MediaStream obtained:", stream);
    // stream을 <video> 또는 <audio> 요소에 연결
    const video = document.querySelector('video');
    video.srcObject = stream;
  })
  .catch(error => {
    console.error("Error accessing media devices:", error);
  });
```

MediaStream 주요 기능

- **사용자 미디어 장치 데이터 나열**
  - navigator.mediaDevices.enumerateDevices 함수를 통해 카메라와 마이크, 스피커 목록을 가져올 수 있습니다.

```
navigator.mediaDevices.enumerateDevices()
  .then(devices => {
    devices.forEach(device => {
      console.log(`Device: ${device.kind}, Label: ${device.label}`);
    });
  })
  .catch(error => {
    console.error("Error listing media devices:", error);
  });
```

MediaStream 주요 기능

- **화면 공유**
  - navigator.mediaDevices.getDisplayMedia 함수를 통해 화면 공유 스트림을 가져올 수 있습니다.

```
navigator.mediaDevices.getDisplayMedia({ video: true })
  .then(stream => {
    console.log("Screen sharing stream:", stream);
  })
  .catch(error => {
    console.error("Error accessing screen sharing:", error);
  });
```

MediaStream 주요 기능

- **미디어 장치의 변경 감지**
  - 연결된 카메라나 마이크가 추가 / 제거 되었을 때 이벤트를 처리합니다.

```
navigator.mediaDevices.addEventListener('devicechange', () => {
  console.log("Media devices changed!");
});
```

RTCDataChannel

- **peer (클라이언트) 간 텍스트나 파일을 주고받기 위한 API**
- 특징
  - WebSocket과 유사한 인터페이스를 제공하며, 더 낮은 레벨에서 작동합니다.
  - P2P 연결을 통해 서버 부하를 감소 시킵니다.
  - 암호화된 데이터 전송을 도와줍니다.

RTCDataChannel

특징	WebRTC	WebSocket
암호화 기본 제공 여부	기본적으로 암호화됨 (DTLS, SRTP 사용)	ws://는 암호화되지 않음 wss://에서만 암호화
암호화 방식	DTLS (데이터 암호화), SRTP (미디어 암호화)	TLS (데이터 암호화)
보안 설정 필요성	별도 설정 없이도 자동으로 보안 처리	암호화를 위해 wss://를 명시적으로 사용해야 함
End-to-End Encryption	브라우저 간 P2P 연결로 완전한 End-to-End 암호화 제공	일반적으로 서버를 통해 데이터가 중계됨
주요 사용 사례	화상 통화, P2P 파일 전송, 게임 데이터 동기화 등	채팅 애플리케이션, 주식 데이터 스트리밍 등

WebRTC의 사용 사례

- 화상 회의 및 음성 통화
  - WebRTC를 사용해 고화질 화상 통화와 다자간 회의를 지원
  - 브라우저에서 별도 플러그인 설치 없이 작동
  - 연결 중 품질 저하시 WebRTC의 적응형 스트리밍(Adaptive Streaming)을 통해 품질 조정

WebRTC의 사용 사례

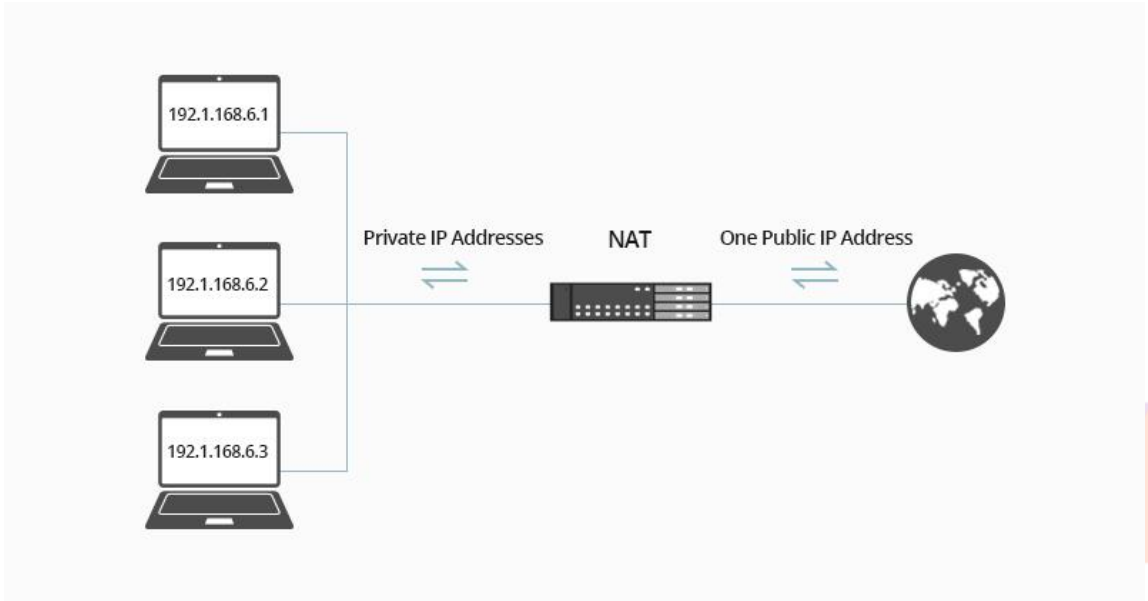
- 실시간 채팅 및 데이터, 파일 전송
  - WebRTC의 RTCDataChannel을 활용해 빠른 응답성과 낮은 지연 시간의 텍스트 채팅 제공
  - 파일 전송 시 서버를 거치지 않고 빠르게 전송 가능

용어 정리

- NAT
- STUN/TURN 서버

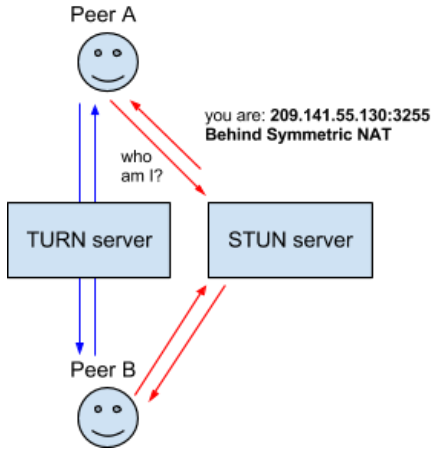
NAT (Network Address Translation)

- WebRTC는 브라우저간 **P2P 연결**을 시도합니다. 하지만 NAT는 사설 네트워크 내부 장치의 IP 주소를 숨기기 때문에 **두 장치간 직접 연결**이 어려워 질 수 있습니다.
- 집에서 사용하는 공유기의 경우 NAT을 통해 공인 주소로 변환하여 인터넷에 접속하게 합니다.
- NAT의 종류(Full Cone, Restricted Cone, Symmetric 등)에 따라 연결 가능 여부가 달라집니다.



STUN 서버

- NAT으로 가려져 있는 자신의 정보를 가져오는 역할을 합니다. 클라이언트가 STUN 서버를 통해 자신의 공인 IP를 얻고 P2P 연결을 하므로, 서버 리소스가 적게 필요합니다.
  - 만약 STUN 서버를 구현하기 힘들거나, 개발 환경에서 사용해야 한다면 무료로 제공하는 구글 STUN 서버를 사용하는 방법도 있습니다.



TURN 서버

- 데이터를 중계하여 NAT 방화벽을 우회합니다. 서버를 경유하므로 지연 시간이 증가되며 서버에 많은 트래픽 부하가 걸릴 수 있습니다.
- 손이 닿지 않는 거리의 물병은 건네 주는것 처럼 멀리 있는 물병을 건네주는 것을 예시로 들 수 있습니다.

STUN / TURN 서버 비교

특징	STUN	TURN
주요 역할	NAT 뒤의 공인 IP 주소와 포트 번호를 확인.	데이터를 중계하여 NAT와 방화벽을 우회.
작동 방식	클라이언트가 STUN 서버를 통해 자신의 공인 IP를 얻음.	클라이언트 간 데이터를 TURN 서버를 통해 중계.
성능	직접 연결(P2P)이 가능하므로 성능이 뛰어남.	서버를 경유하므로 지연 시간이 증가.
비용	서버 리소스가 적게 필요함.	서버에 많은 트래픽 부하가 걸릴 수 있음.
사용 환경	NAT 종류에 따라 연결이 가능.	P2P 연결이 불가능한 환경에서 사용.

한계와 도전 과제

- 브라우저 호환성 문제
- 네트워크 품질 문제
- 대규모 P2P 연결의 한계

브라우저 호환성 문제

- WebRTC는 다양한 브라우저에서 실시간 통신을 지원하도록 설계되었지만, 브라우저마다 WebRTC 구현 방식과 지원 수준이 다르기 때문에 호환성 문제가 발생할 수 있습니다.

네트워크 품질 문제

- P2P 통신을 기반으로 하기 때문에 네트워크 환경에 따라 품질이 크게 좌우됩니다.
  - 지연시간: 네트워크 혼잡이나 물리적 거리로 인해 지연 시간이 길어질 수 있습니다. 때문에 음성이 왜곡되거나 비디오가 동기화되지 않을 수 있습니다.
  - 패킷 손실이 발생하면 오디오 및 비디오 품질이 급격히 저하됩니다.

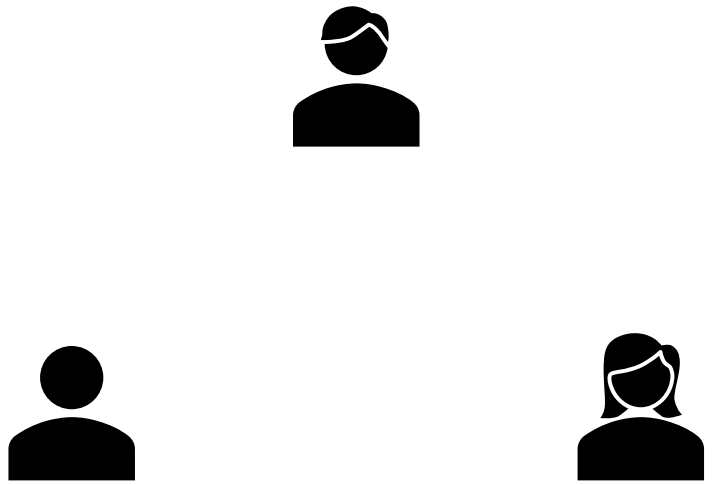
대규모 P2P 연결의 한계

- 브라우저 간의 P2P 연결을 기반으로 설계되었습니다. 이는 소규모 통신에 적합하지만, 대규모 연결을 처리할 때 한계가 있습니다.

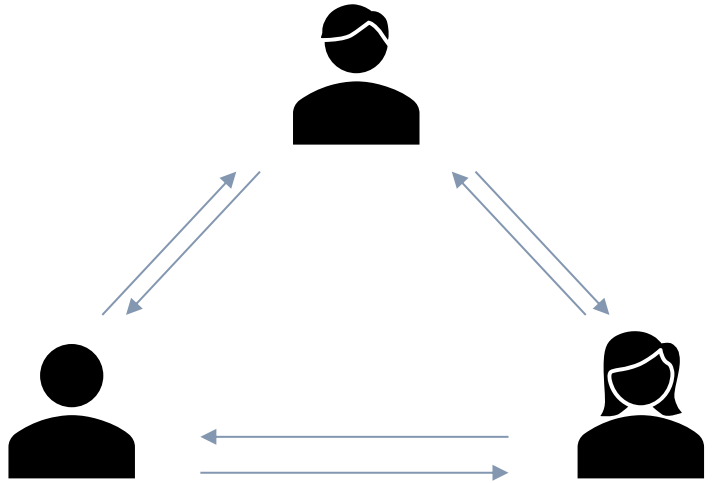
대규모 P2P 연결의 한계

- 브라우저 간의 P2P 연결을 기반으로 설계되었습니다. 이는 소규모 통신에 적합하지만, 대규모 연결을 처리할 때 한계가 있습니다.

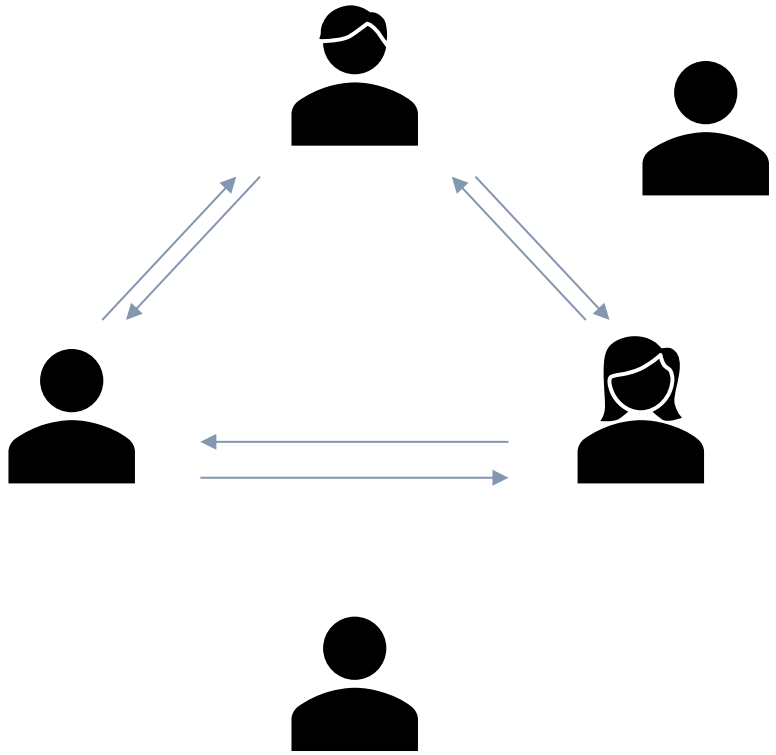
대규모 P2P 연결의 한계



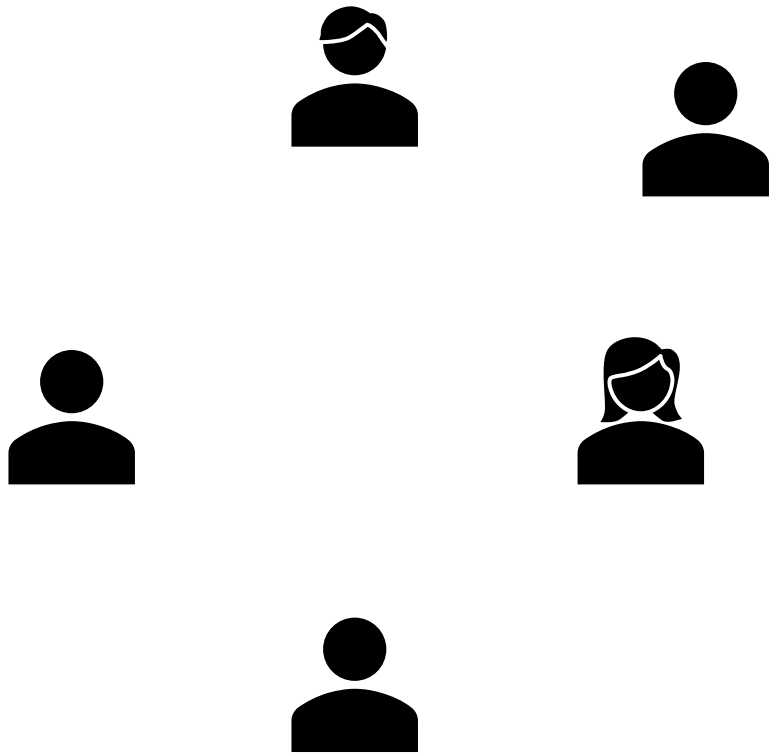
대규모 P2P 연결의 한계



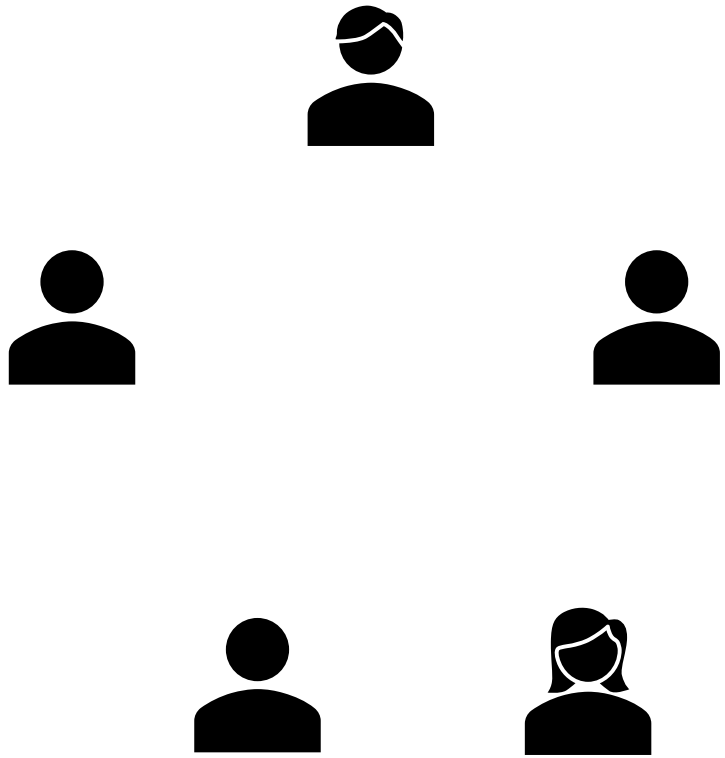
대규모 P2P 연결의 한계



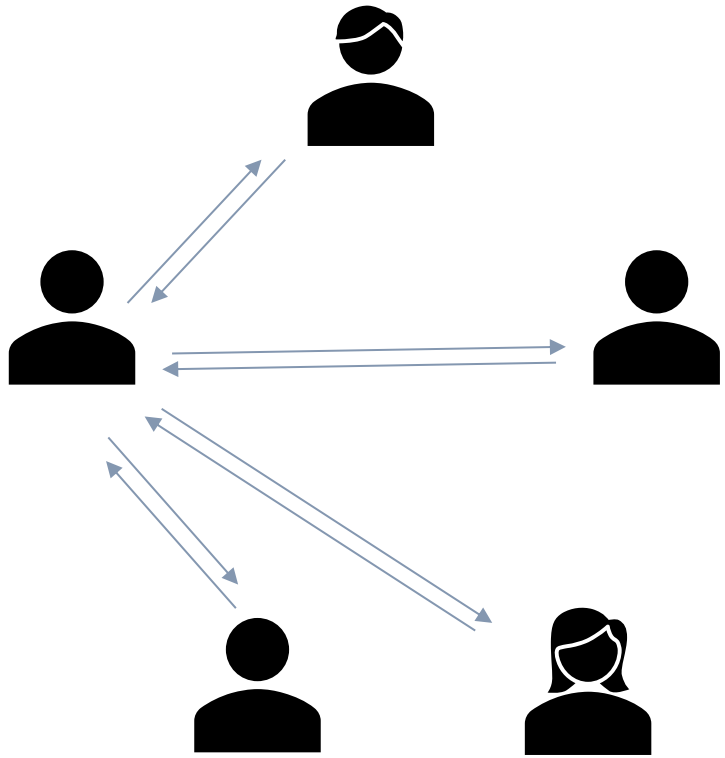
대규모 P2P 연결의 한계



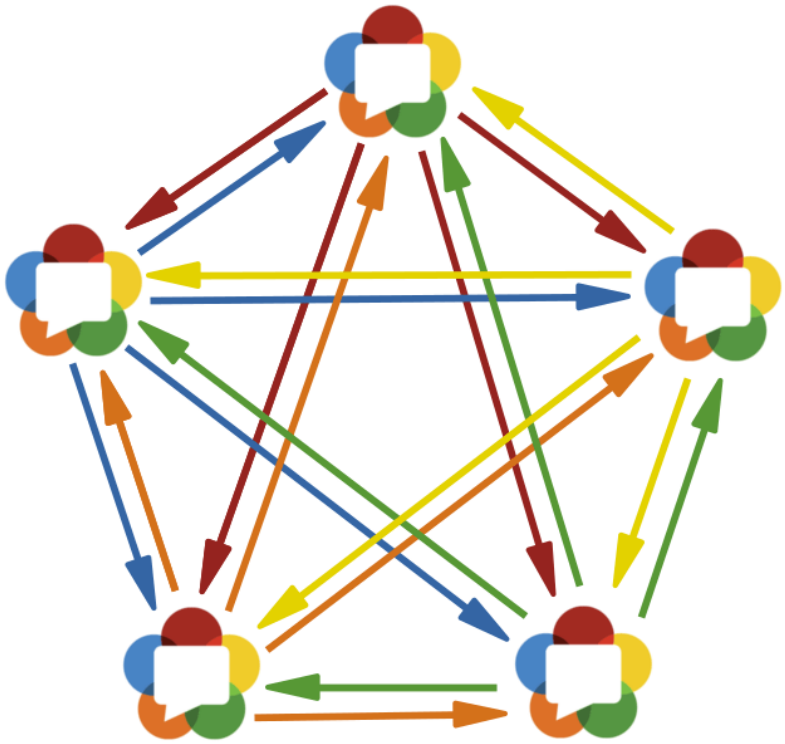
대규모 P2P 연결의 한계



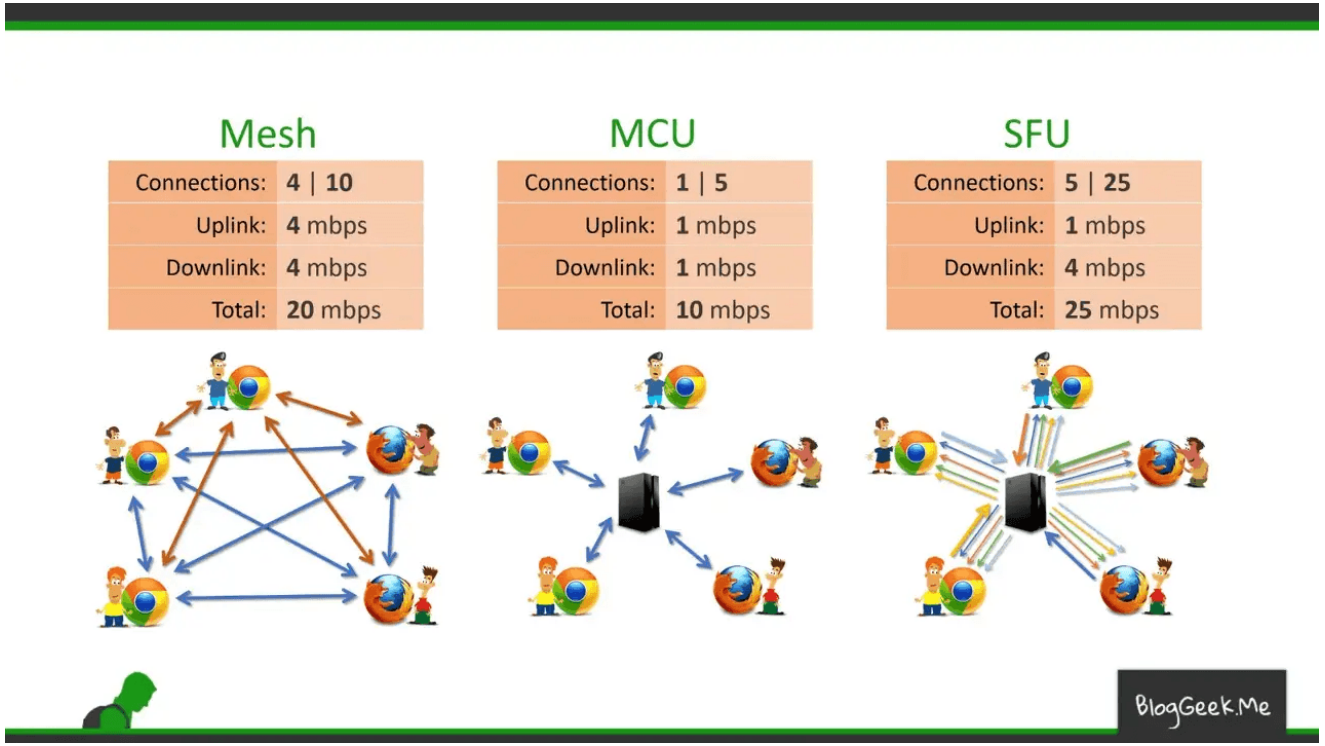
대규모 P2P 연결의 한계



대규모 P2P 연결의 한계



대규모 P2P 연결의 한계



감사합니다.