Rete P2P

–

Progettare e implementare un sistema di comunicazione peer-to-pee

Reti e Comunicazione Digitale

a.a. 2024-2025

Corso di laurea in

Informatica e Comunicazione Digitale – Sede di Taranto

Team “I Carlini”		
Cognome & Nome	**Matricola**	**Email**
Carlucci Manuel	775438	m.carlucci69@studenti.uniba.it
Nitti Vittorio	775161	v.nitti19@studenti.uniba.it

Docente: Lopez Ugo

Sommario

Panoramica Teorica **4**
Rete Peer to Peer 4
WebRTC 4
Server STUN e TURN 5
ICE Candidates 6
Algoritmo di Connessione WebRTC 7
Web Sockets VS WebRTC 8

Caso di Studio **9**
Stakeholder 9
Item Funzionali 10
Diagramma dei Casi d'uso 10
Specifiche dei Casi d'uso 11

Scelta Progettuale **16**
Connessione fra 2 peer o multi-peer 16
Soluzioni Tecniche – Firebase Vs Agora.io 17
Firebase 17
Firebase per connessioni Peer-to-Peer 17
Agora.io 17
Agora.io per connessioni Peer-to-Peer 18
Confronto tra Firebase e Agora.io 18
Scelta Effettuata 18

Implementazione **19**
Configurazione della connessione P2P 19
Gestione della Chat 20
Invio e Ricezione di File 20
Gestione Videochiamata 20
Disconnessione dei Peer 21
Distribuzione 21

Informazioni di installazione **22**
Passaggi per l'installazione 22
Installazione e Avvio 22
Risoluzione Problemi 22

Sviluppi Futuri **23**

Glossario **24**
Acronimi 24
Definizioni 25

Riferimenti **26**

Panoramica Teorica

In questa sezione esploreremo quelli che sono i dettagli teorici della soluzione da noi implementata per sviluppare un sistema Peer to Peer per scambiare informazioni e file.

Rete Peer to Peer

Nel contesto delle telecomunicazioni, il termine **peer-to-peer** (spesso abbreviato in **P2P**) si riferisce a un modello architetturale di rete informatica basato su una distribuzione paritetica delle risorse e delle responsabilità tra i nodi partecipanti. A differenza delle tradizionali reti client-server, in cui i ruoli di client e server sono distinti e staticamente assegnati, nelle reti **P2P** i nodi (detti *peer*, ovvero "pari" in inglese) possono agire contemporaneamente sia come client che come server. Questo significa che ogni nodo non si limita a richiedere dati o servizi, ma può anche offrirli agli altri nodi della rete.

WebRTC

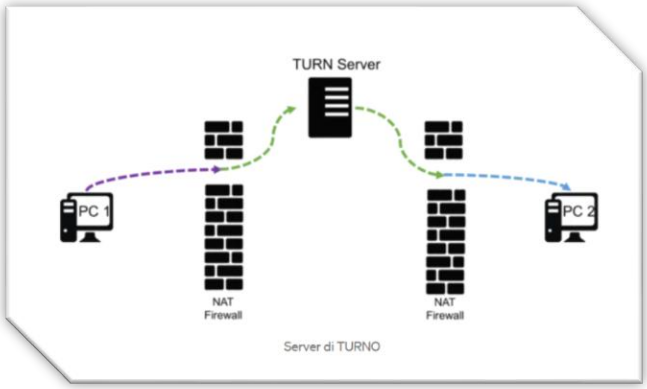
WebRTC (Web Real-Time Communication) è un'API di comunicazione in tempo reale che utilizza una connessione peer-to-peer diretta. È un progetto open source con specifiche pubblicate da W3C e IETF. WebRTC consente la trasmissione di audio, video e dati tra dispositivi senza la necessità di un server intermedio per il trasferimento dei media.

Per stabilire una connessione peer-to-peer, i peer devono scambiarsi informazioni sugli indirizzi IP e concordare il formato per la trasmissione dei dati, come risoluzione video e codec.

- Il primo peer genera **un'offerta (SDP** - Session Description Protocol), che contiene dettagli come codec supportati, velocità di trasmissione e indirizzi IP disponibili.
- L'SDP viene inviato al **server di segnalazione**, che funge da intermediario per il trasferimento delle informazioni tra i peer.
- Il secondo peer riceve l'SDP e genera una risposta (**SDP Answer**) con i dettagli della connessione accettata.
- La risposta viene quindi inviata al primo peer tramite il server di segnalazione, che facilita l'incontro tra i due dispositivi ma non gestisce il traffico multimediale. Questo processo è noto come **signaling**.

Server STUN e TURN

Molti dispositivi si trovano dietro un router o un firewall, il che significa che hanno un **indirizzo IP privato** e non sono direttamente raggiungibili da Internet. Il **server STUN** aiuta a scoprire l'IP pubblico, ma questo non garantisce automaticamente che una connessione diretta sia possibile.
Quando un peer si connette a un server STUN, questo risponde restituendo l'IP pubblico e la porta attraverso cui il peer è visibile su Internet. Con queste informazioni, WebRTC può generare un **SDP (Session Description Protocol)** corretto, che viene utilizzato nel processo di negoziazione della connessione tra i peer.
Tuttavia, questo IP pubblico appartiene al router/NAT del peer e non direttamente al suo dispositivo. Di conseguenza, se Peer 1 prova a inviare un file all'indirizzo IP pubblico e alla porta di Peer 2, il pacchetto verrà bloccato se il NAT non sa dove inoltrarlo.



Se i peer non riescono a comunicare direttamente a causa di una configurazione di rete troppo restrittiva (ad esempio, firewall aziendali o NAT complessi), entra in gioco il **server TURN**.
Il server TURN funge da **relay** intermedio: invece di stabilire una connessione diretta tra i peer, entrambi inviano i loro dati al server TURN, che li instrada al destinatario corretto.
L'uso di TURN garantisce la connettività anche nelle situazioni più difficili, ma introduce **maggiore latenza** e **costi di banda** perché tutto il traffico passa attraverso il server intermedio.

WebRTC utilizza STUN per tentare di superare le limitazioni NAT e TURN come fallback quando la connessione diretta non è possibile.

ICE Candidates

Gli indirizzi IP possono variare a causa della NAT (Network Address Translation), rendendo complicata la connessione diretta tra peer. Per superare questo ostacolo, WebRTC utilizza **ICE (Interactive Connectivity Establishment)**, un framework che facilita la scoperta e la connessione tra peer. Ogni peer genera una lista di **ICE candidates**, combinazioni di indirizzi IP e porte disponibili, che vengono scambiati attraverso il server di segnalazione per trovare il percorso di connessione più efficiente.

Un ICE candidate contiene

- un indirizzo IP (che può essere privato, pubblico o associato a un server TURN),
- una porta di ricezione dei dati,
- il protocollo utilizzato (UDP o TCP),
- il tipo di candidato (host, server reflexive o relay).

Esistono tre tipi principali di ICE candidates:

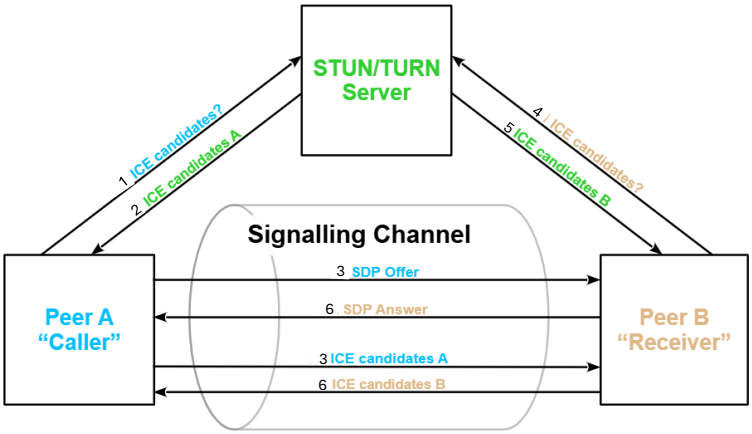
- **Host Candidates:** sono indirizzi IP locali che identificano il dispositivo nella sua rete privata. Funzionano solo per connessioni nella stessa rete locale e non sono utili se i peer si trovano dietro NAT diversi.
- **Server Reflexive Candidates (srflx):** ottenuti tramite un server STUN, rappresentano l'IP pubblico e la porta assegnata dal NAT. Sono utilizzati per connessioni dirette tra peer dietro NAT, se le condizioni di rete lo permettono.
- **Relay Candidates:** forniti da un server TURN, vengono utilizzati quando la connessione diretta non è possibile, ad esempio in presenza di un NAT simmetrico o di firewall restrittivi. Tuttavia, l'uso di TURN introduce una maggiore latenza poiché il traffico passa attraverso il server intermediario.

Il processo di ICE si articola in quattro fasi principali.

1. Nella fase di **raccolta (gathering)**, ogni peer raccoglie tutti i possibili ICE candidates disponibili e li invia progressivamente all'altro peer.
2. Segue la fase di **scambio (signaling)**, in cui i peer si trasmettono i candidates attraverso un server di signaling.
3. A questo punto, viene eseguito il **test di connettività (connectivity check)**, in cui i peer provano a stabilire una connessione utilizzando diverse combinazioni di ICE candidates, scegliendo quella funzionante.
4. Infine, avviene la **selezione del percorso migliore (nominazione)**, in cui viene scelto il candidato più efficiente in base a latenza e stabilità. Se una connessione diretta è possibile, viene preferita rispetto all'uso di un server TURN per garantire prestazioni ottimali.

Algoritmo di Connessione WebRTC

Il processo di connessione inizia con lo scambio degli SDP tra i peer tramite il server di segnalazione. Successivamente, i candidati ICE vengono generati e scambiati per individuare il percorso di connessione ottimale. Se possibile, WebRTC stabilisce una connessione **peer-to-peer diretta**. Se invece firewall o NAT simmetrici impediscono questa connessione, entra in gioco un server TURN che funge da relay intermedio. Una volta stabilita la connessione, i dati vengono trasmessi in tempo reale e, se necessario, la connessione può essere aggiornata per adattarsi ai cambiamenti della rete.



Passaggi di connessione riassunti:

1. **Peer 1 contatta il server STUN**

- Ottiene il suo **IP pubblico e la porta assegnata dal NAT** (server reflexive candidate).
- Raccoglie anche altri ICE candidates (host, relay se necessario)

2. **Peer 1 crea una SDP offer**
    - La SDP (Session Description Protocol) contiene informazioni sulla sessione multimediale, i codec supportati e gli **ICE candidates** disponibili.

3. **Peer 1 invia la SDP offer a Peer 2 tramite il server di segnalazione**
    - Peer 1 **non conosce direttamente l'IP** di Peer 2 all'inizio.
    - Usa un **server di signaling** (WebSocket, WebRTC signaling server, o un backend personalizzato) per inviare la SDP offer a Peer 2.

4. **Peer 2 riceve la SDP offer e risponde con una SDP answer**
    - Peer 2 raccoglie i suoi **ICE candidates** e li aggiunge alla risposta.
    - Invia la SDP answer a Peer 1 sempre tramite il **server di signaling**.

5. **Entrambi i peer iniziano gli ICE Connectivity Checks**
    - Testano le combinazioni di ICE candidates per trovare la connessione migliore.
    - Se possibile, stabiliscono una connessione **diretta (P2P)**.
    - Se il NAT è restrittivo, viene usato un **server TURN** per inoltrare il traffico.

La SDP offer non viene mandata direttamente a Peer 2, perché Peer 1 **non conosce ancora il suo indirizzo IP**. È il **server di signaling** che permette ai due peer di scambiarsi queste informazioni iniziali.

Web Sockets VS WebRTC

WebSockets e WebRTC sono tecnologie simili, in quanto entrambe permettono la condivisione di informazioni in tempo reale. Tuttavia, si differenziano nel modo in cui gestiscono la comunicazione tra i peer.

Con **WebSockets**, i dati passano sempre attraverso un **server intermedio**, creando un modello di comunicazione **Peer-to-Server**. Invece, **WebRTC** consente una connessione **diretta (Peer-to-Peer)** tra i browser, evitando il passaggio dei dati attraverso un server.

Entrambe le tecnologie supportano la comunicazione in tempo reale, ma con WebSockets, se Peer 1 vuole inviare un messaggio a Peer 2, il messaggio viene instradato dal server, così come la risposta di Peer 2. Questo introduce una certa **latenza**, che potrebbe non essere evidente nel trasferimento di file, ma diventa significativa nel caso di **audio e video in tempo reale**. WebRTC, eliminando il passaggio attraverso un server, permette uno scambio di dati più veloce ed efficiente.

WebRTC utilizza il **protocollo UDP** per la trasmissione dei dati, che è molto veloce ma **non garantisce l'affidabilità** del trasferimento. UDP non verifica se i dati sono stati ricevuti correttamente: in uno **streaming video**, la perdita di alcuni frame è accettabile, ma nel **trasferimento di file**, la perdita di anche pochi byte può corrompere l'intero file. Per questo motivo, WebRTC non è la scelta ideale per la trasmissione di dati critici che richiedono un'affidabilità elevata.

Un'altra differenza importante è che **WebRTC non include un meccanismo di signaling integrato**. Per avviare la connessione tra i peer, è necessario un **canale di signaling** separato, che tipicamente utilizza **WebSockets**. Questo processo di signaling permette ai due peer di scambiarsi le informazioni necessarie per stabilire la connessione, dopodiché WebRTC prende il controllo della comunicazione diretta.

Caso di Studio

Lo scopo di questo progetto è lo sviluppo di un sistema **peer-to-peer** che consenta agli utenti di scambiare file e messaggi in modo diretto, senza la necessità di un server centrale per instradare i dati.

Il sistema è progettato per essere utilizzato su diversi dispositivi, consentendo una comunicazione fluida e affidabile tra utenti che desiderano condividere informazioni in tempo reale.

Stakeholder

Nella realizzazione di questo sistema sono coinvolti diversi attori, ciascuno con un ruolo specifico nel processo di sviluppo e utilizzo del sistema.

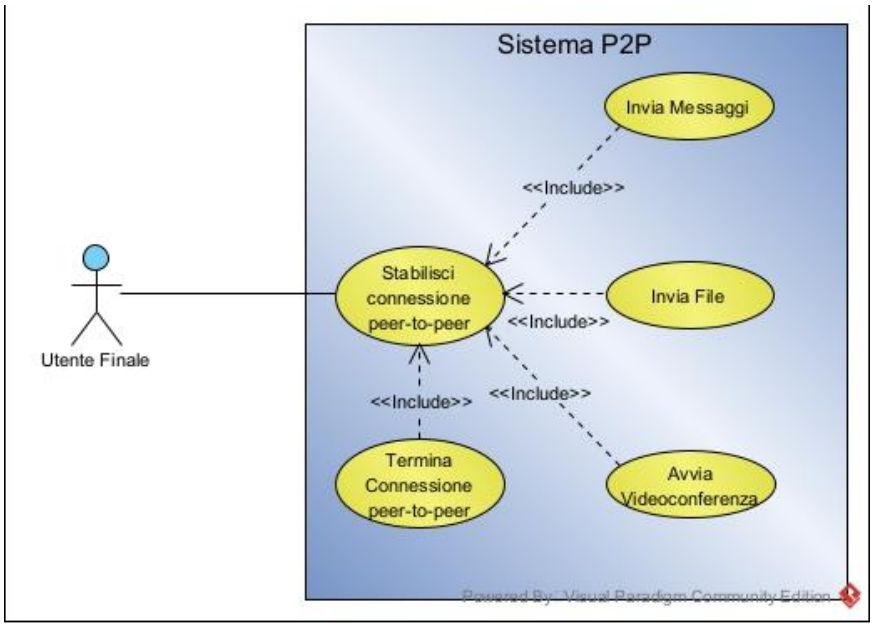
- Committente:
    - Docente: Ugo Lopez
    - Il docente svolge il ruolo di supervisore e valutatore del progetto, fornendo linee guida, requisiti e specifiche per l'implementazione del sistema. Ha il compito di verificare che il progetto soddisfi gli obiettivi prefissat
- Analisti dei requisiti, progettisti, sviluppatori del sistema e technical writer:
    - Carlucci Manuel
    - Nitti Vittorio
    - Questo gruppo è responsabile dell'analisi dei requisiti definiti dal docente, della progettazione dell'architettura del sistema, della scrittura del codice e della documentazione tecnica. Ogni membro del team contribuisce attivamente allo sviluppo del progetto.
- Utente finale:
    - Qualsiasi individuo che desideri utilizzare il sistema per la condivisione di file e messaggi.
    - L'utente finale è la figura centrale per la quale il sistema è stato progettato. Deve poter interagire con l'applicazione in modo intuitivo, senza doversi preoccupare di configurazioni complesse.

Item Funzionali

CODICE	DESCRIZIONE	PRIORITÀ
IF-01	**Come** Utente Finale **Voglio** avere la possibilità di stabilire una connessione peer-to-peer con un secondo utente, **Così che** possa condividere informazioni col secondo utente direttamente, senza passare da un server centrale	**Alta**
IF-02	**Come** Utente Finale **Voglio** avere la possibilità di terminare una connessione peer-to-peer precedentemente stabilita, **Così che** possa interrompere la comunicazione con l'altro peer garantendo che nessun dato venga più trasmesso o ricevuto.	**Alta**
IF-03	**Come** Utente Finale **Voglio** avere la possibilità di inviare messaggi in tempo reale a un altro peer, **Così che** la comunicazione avvenga in modo diretto e sicuro senza dipendere da un'infrastruttura server	**Alta**
IF-04	**Come** Utente Finale **Voglio** poter trasferire file a un altro peer in maniera diretta **Così che** lo scambio di dati sia rapido e senza congestione di rete causata da un server centrale.	**Alta**
IF-05	**Come** Utente Finale **Voglio** poter avviare una videoconferenza con un secondo peer **Così che** possa comunicare in tempo reale tramite audio e video senza dipendere da un server centrale.	**Bassa**

Diagramma dei Casi d'uso

Di seguito il diagramma dei casi d'uso del sistema che intendiamo sviluppare.



Specifiche dei Casi d'uso

Di seguito andremo a dettagliare i casi d'uso precedentemente definiti nel diagramma dei casi d'uso.

CU-01

NOME	**STABILISCI CONNESSIONE PEER-TO-PEER**
ID	CU-01
BREVE DESCRIZIONE	Un utente avvia una connessione peer-to-peer con un altro utente del sistema attraverso WebRTC, utilizzando un meccanismo di signaling basato su Firebase.
PRE-CONDIZIONI	Aver fatto l'accesso al sistema indicando il proprio nome.
ATTORI PRIMARI	Utente
ATTORI SECONDARI	Sistema (gestione signaling tramite Firebase)
SEQUENZA PRINCIPALE DEGLI EVENTI	1. Il primo utente (Peer A) genera una offer SDP per avviare la comunicazione. 2. Il sistema memorizza l'offer SDP nel database. 3. Il sistema raccoglie gli ICE candidates di Peer A e li salva nel database. 4. Il sistema genera un ID univoco di connessione e lo associa alla sessione. 5. Peer A condivide l'ID con il secondo peer (Peer B) attraverso un canale esterno (chat, email, ecc.). 6. Peer B inserisce l'ID nel sistema e recupera l'offer SDP. 7. Peer B genera un'answer SDP e la salva nel database. 8. Il sistema raccoglie gli ICE candidates di Peer B e li salva nel database. 9. Peer A recupera l'answer SDP e gli ICE candidates di Peer B. 10. Il sistema stabilisce la connessione WebRTC diretta tra Peer A e Peer B.
POST-CONDIZIONI	La connessione fra i due peer è stata stabilita
SEQUENZA ALTERNATIVA DEGLI EVENTI	Nessuna

CU-02

NOME	TERMINA CONNESSIONE PEER
ID	CU-02
BREVE DESCRIZIONE	Un utente chiude una connessione peer-to-peer precedentemente stabilita.
PRE-CONDIZIONI	• La connessione peer-to-peer è stata stabilita con successo (CU-01).
ATTORI PRIMARI	Utente
ATTORI SECONDARI	Sistema (WebRTC per la trasmissione dei messaggi)
SEQUENZA PRINCIPALE DEGLI EVENTI	Include “CU:01 – Stabilisci Connessione peer-to-peer” 1. L’utente chiede la terminazione della connessione. 2. Il sistema invia un messaggio di chiusura all'altro peer. 3. Il sistema chiude il RTCDataChannel e la RTCPeerConnection.
POST-CONDIZIONI	La connessione peer-to-peer è stata terminata e non è più attiva.

CU-03

NOME	INVIA MESSAGGIO
ID	CU-03
BREVE DESCRIZIONE	Un utente invia un messaggio testuale a un altro utente attraverso una connessione peer-to-peer stabilita.
PRE-CONDIZIONI	• La connessione peer-to-peer è stata stabilita con successo (CU-01). • Il RTCDataChannel è stato creato e aperto.
ATTORI PRIMARI	Utente
ATTORI SECONDARI	Sistema (WebRTC per la trasmissione dei messaggi)
SEQUENZA PRINCIPALE DEGLI EVENTI	Include “CU:01 – Stabilisci Connessione peer-to-peer” 4. L’utente scrive un messaggio nel campo di input della chat e lo invia. 5. Il sistema invia il messaggio tramite il RTCDataChannel. 6. Il secondo utente riceve il messaggio e il sistema lo visualizza nella chat.
POST-CONDIZIONI	Il messaggio è stato inviato e visualizzato nella chat del destinatario.

CU-04

NOME	INVIA FILE
ID	CU-04
BREVE DESCRIZIONE	Un utente invia un file a un altro utente attraverso la connessione peer-to-peer.
PRE-CONDIZIONI	• La connessione peer-to-peer è stata stabilita con successo (CU-01). • Il RTCDataChannel è attivo e funzionante.
ATTORI PRIMARI	Utente
ATTORI SECONDARI	Sistema (WebRTC per il trasferimento dei file)
SEQUENZA PRINCIPALE DEGLI EVENTI	Include “CU:01 – Stabilisci Connessione peer-to-peer” 1. L’utente seleziona un file dal proprio dispositivo. 2. Il sistema frammenta il file in pacchetti per il trasferimento. 3. Il sistema invia i pacchetti di dati attraverso il RTCDataChannel. 4. Il secondo utente riceve i pacchetti e il sistema li riassembla. 5. Il sistema genera un link per il download e lo mostra nella chat del destinatario.
POST-CONDIZIONI	Il file è stato trasferito e il destinatario può scaricarlo.

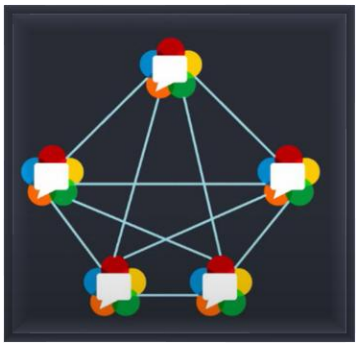
CU-05

NOME	INVIA FILE
ID	CU-05
BREVE DESCRIZIONE	Un utente avvia una videochiamata con un altro utente attraverso WebRTC.
PRE-CONDIZIONI	• La connessione peer-to-peer è stata stabilita con successo (CU-01). • Il browser ha concesso i permessi per accedere alla webcam e al microfono.
ATTORI PRIMARI	Utente
ATTORI SECONDARI	Sistema (WebRTC per la trasmissione video e audio)
SEQUENZA PRINCIPALE DEGLI EVENTI	Include “CU:01 – Stabilisci Connessione peer-to-peer” 1. L’utente Avvia la Videochiamata. 2. Il sistema richiede l’accesso alla webcam e al microfono. 3. Il sistema cattura i flussi audio e video e li trasmette all'altro peer. 4. Il secondo utente riceve i flussi e visualizza la videochiamata.
POST-CONDIZIONI	La videochiamata è attiva tra i due peer.

Scelta Progettuale

Connessione fra 2 peer o multi-peer

Per implementare una connessione **peer-to-peer** con più dispositivi contemporaneamente, possiamo seguire tre approcci principali:

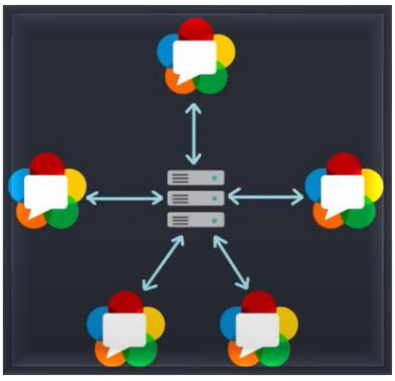


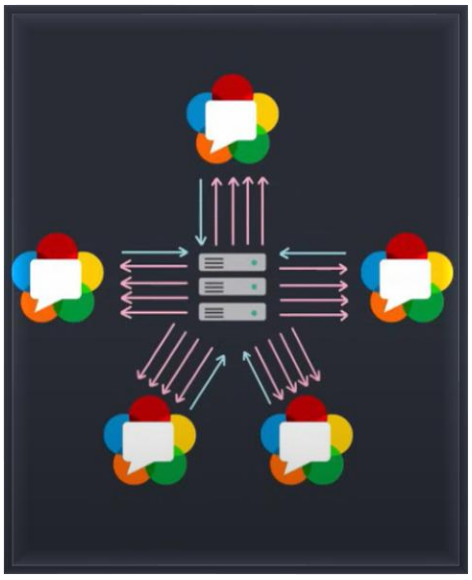
1. Mesh Network
In questo modello, ogni peer stabilisce una connessione diretta con tutti gli altri peer del gruppo. Ad esempio, con 5 peer, ogni dispositivo dovrà mantenere 4 connessioni separate. Quando un nuovo peer si unisce alla rete, deve connettersi a tutti gli altri, e viceversa.

Vantaggio: Comunicazione **completamente peer-to-peer**.
Svantaggio: Scalabilità limitata: Il numero di connessioni cresce esponenzialmente con l'aumentare dei peer.

2. MCU (Multipoint Control Unit)
Questo approccio si avvicina di più a una Peer-to-Server. Tutti i peer si connettono a un server centrale, che raccoglie i flussi in entrata, li elabora e li mixa prima di inviarli agli altri peer.

Vantaggio: Maggiore scalabilità, poiché i peer non gestiscono direttamente più connessioni.
Svantaggio: Elevati costi di elaborazione, poiché il server deve processare e mixare tutti i flussi.





3. SFU (Selective Forwarding Unit)
Anche in questo caso si utilizza un server intermedio, ma a differenza dell'MCU, il server non mixa i flussi, limitandosi a inoltrarli ai peer che li richiedono. Il server funge quindi da router, dirigendo i flussi video/audio ai destinatari appropriati.

Vantaggi:

- **Meno carico sul server** rispetto all'MCU, poiché non deve elaborare i flussi.
- **Migliore scalabilità** rispetto alla Mesh Network, poiché ogni peer carica solo un singolo stream, riducendo il consumo di banda in upload.

Svantaggio: Maggiore richiesta di banda in download per i peer, poiché devono ricevere più stream separati.

Considerazioni Finali
Per semplificare l'implementazione, possiamo utilizzare **servizi di terze parti** che offrono soluzioni WebRTC pronte all'uso. Molti di questi servizi sono a pagamento, ma esistono anche opzioni **gratuite o con limitazioni**.

Soluzioni Tecniche – Firebase Vs Agora.io

Firebase

Firebase è una piattaforma di sviluppo per applicazioni mobili e web fornita da Google. Funziona come **Backend-as-a-Service (BaaS)**, offrendo una suite di strumenti che semplificano la gestione del backend, dei database, dell'autenticazione, dell'hosting e di altre funzionalità, senza la necessità di gestire direttamente un'infrastruttura server-side.

Di seguito le funzionalità principali di Firebase

- **Realtime Database & Firestore**: Database NoSQL che consente la sincronizzazione dei dati in tempo reale.
- **Authentication**: Fornisce un sistema di login con email/password, Google, Facebook e altri provider.
- **Cloud Storage**: Permette l'archiviazione e il recupero di file tramite Google Cloud.
- **Cloud Messaging (FCM)**: Sistema di notifiche push per Android, iOS e Web.
- **Cloud Functions**: Esecuzione di codice serverless in risposta a eventi specifici.

Firebase per connessioni Peer-to-Peer

Firebase non fornisce direttamente connessioni P2P, poiché è basato su un'architettura client-server. Tuttavia, può essere utilizzato per stabilire e coordinare connessioni P2P attraverso meccanismi di **signaling**.

Uno dei problemi principali nelle connessioni P2P è la necessità di un **server di signaling**, il cui compito è permettere ai peer di scambiarsi informazioni per avviare una connessione diretta. Firebase può essere utilizzato in questa fase, consentendo ai dispositivi di:

- Registrarsi nel database con un identificatore univoco.
- Inviare il proprio indirizzo IP e altre informazioni di connessione.
- Recuperare i dettagli del peer con cui vogliono connettersi.

Dopo lo scambio di queste informazioni, la connessione P2P può avvenire tramite **WebRTC**, consentendo la trasmissione diretta dei dati tra i dispositivi.

Agora.io

Agora.io è una piattaforma **CPaaS (Communication Platform-as-a-Service)** che fornisce strumenti per la comunicazione in tempo reale, inclusi **streaming audio/video, videochiamate e messaggistica interattiva**. Il suo SDK permette di integrare facilmente funzionalità di comunicazione in applicazioni web e mobili.

Agora.io per connessioni Peer-to-Peer

Agora.io utilizza WebRTC per la trasmissione dei dati, ma a differenza di una pura connessione P2P, si appoggia a server intermedi per garantire una connessione stabile e ottimizzata. Questo permette di:

- **Garantire connessioni affidabili** anche in condizioni di rete instabili.
- **Ottimizzare la latenza** grazie ai suoi algoritmi di gestione della rete.
- **Supportare più partecipanti,** utile, per esempio, per gruppi di peer senza che ogni peer debba connettersi a tutti gli altri direttamente.

Agora.io non è una soluzione completamente peer-to-peer nel senso tradizionale, poiché la sua architettura utilizza server relay per migliorare l'esperienza utente. Tuttavia, per applicazioni che necessitano di **alta qualità audio/video e scalabilità**, Agora.io è una scelta più efficace rispetto a una pura connessione P2P.

Confronto tra Firebase e Agora.io

Caratteristica	***Firebase***	***Agora.io***
Tipo di servizio	BaaS (Backend-as-a-Service)	CPaaS (Communication Platform-as-a-Service)
Approccio alla connessione	Basato su un'architettura client-server, può essere usato per il signaling in WebRTC	Fornisce API per streaming audio/video in tempo reale e comunicazione interattiva
Supporto P2P	Non supporta direttamente P2P, ma può facilitare il signaling	Supporta direttamente WebRTC, ma usa server intermedi per garantire stabilità
Scalabilità	Altamente scalabile per backend e database	Scalabile per applicazioni di streaming e videoconferenza
Qualità della connessione	Dipende dalla connessione diretta tra peer (P2P)	Offre ottimizzazioni per reti instabili, garantendo bassa latenza
Sicurezza	Gestisce autenticazione, database e storage con regole di accesso granulari	Protezione con crittografia end-to-end e controllo delle sessioni
Costo	Modello di pricing basato sull'uso di database, storage e funzioni cloud	Modello di pricing basato sui minuti di utilizzo per streaming audio/video

Scelta Effettuata

Nel contesto della realizzazione di una connessione **peer-to-peer (P2P) diretta** tramite **WebRTC**, Firebase può essere impiegato per la fase di **signaling**, facilitando lo scambio delle informazioni necessarie tra i peer per stabilire la connessione. D'altra parte, **Agora.io** offre un'infrastruttura ottimizzata per la gestione dei flussi multimediali in tempo reale, risultando particolarmente utile in scenari con reti instabili o sistemi multi-peer.

Poiché l'obiettivo principale del nostro sistema non prevede la gestione di **streaming video** né la necessità di supportare **connessioni multi-peer**, la soluzione adottata sarà basata su **un'architettura P2P pura**, utilizzando **Firebase esclusivamente come server di signaling** per coordinare l'inizializzazione della connessione tra i dispositivi.

Implementazione

Di seguito illustriamo i passi implementativi applicati nello sviluppo.

Configurazione della connessione P2P

Per la realizzazione di una connessione peer-to-peer (P2P), abbiamo utilizzato le funzionalità offerte da **WebRTC**, in combinazione con **Firestore** per il **signaling**.

Il processo è iniziato con la creazione di un account su Firebase e la configurazione di un progetto, denominato "Rete PTP". Una volta impostato il **Cloud Firestore**, è stato predisposto un database per la gestione dei messaggi di chat e lo scambio di informazioni necessarie alla connessione tra i peer.

La prima fase ha riguardato la configurazione della connessione P2P, sfruttando gli ICE candidates forniti dagli STUN server gratuiti di Google. Dopo aver ottenuto l'accesso alle tracce audio e video del peer locale tramite navigator.mediaDevices.getUserMedia, queste sono state aggiunte alla peerConnection, garantendo i permessi per webcam e microfono. Queste tracce sono essenziali per generare correttamente l'SDP, questo perché l'SDP descrive i media stream da trasmettere. Senza tracce audio o video attive, l'SDP risulta incompleto o non valido, rendendo impossibile la negoziazione della chiamata. Per evitare l'inoltro accidentale delle tracce prima dell'effettiva connessione, l'audio e il video del peer locale sono stati inizialmente disabilitati.

Successivamente, è stato creato un oggetto MediaStream per il peer remoto, con l'evento ontrack della peerConnection configurato per ricevere le tracce provenienti dall'altro peer. Contestualmente, è stato implementato un DataChannel denominato chatChannel, che consente la comunicazione di messaggi di testo tra i peer.

Per il signaling server, Firestore è stato utilizzato per scambiare i dati di connessione. Nella collezione calls è stato creato un documento contenente due sottocollezioni: offerCandidates e answerCandidates, deputate alla gestione degli ICE candidates. Il browser, attraverso l'evento onicecandidate, salva questi candidati su Firestore, consentendo ai peer di recuperarli e completare il processo di negoziazione della connessione.

Una volta creata l'offerta SDP con createOffer, questa è stata impostata come descrizione locale tramite setLocalDescription e memorizzata su Firestore. Un listener monitorava eventuali risposte (answer) per aggiornare la descrizione remota con setRemoteDescription una volta ricevuta la risposta dall'altro peer. Parallelamente, è stata implementata la gestione degli ICE candidates della risposta, trasmettendoli tra i peer sempre tramite Firestore.

Terminata la fase di connessione, gli stream video sono stati collegati agli elementi del DOM per consentire la visualizzazione delle webcam dei peer. Per migliorare l'esperienza utente, è stata inoltre implementata una funzione che permette agli utenti di inserire il proprio nome e visualizzarlo nell'interfaccia.

In maniera analoga è stata gestita la risposta del secondo peer, usando come fulcro della connessione l'ID generato da firestore.

Gestione della Chat

Una volta stabilita la connessione, la gestione della chat è avvenuta tramite il **DataChannel**. Quando un peer riceve un canale dati dall'altro, l'evento ondatachannel lo intercetta e invoca la funzione setupDataChannel, responsabile della sua configurazione. Se la connessione viene stabilita correttamente, viene mostrato un messaggio di conferma all'interno della chat.

Il DataChannel è stato configurato per gestire diversi stati. All'apertura, viene registrato un log nella console per indicare che la comunicazione è attiva. In caso di errore, il problema viene segnalato tramite onerror, mentre la chiusura viene intercettata dall'evento onclose.

L'invio dei messaggi avviene attraverso un campo di input che, una volta compilato e confermato con il tasto "Invio" o tramite il pulsante di invio, aggiorna l'interfaccia con il messaggio inviato e lo trasmette al peer remoto utilizzando dataChannel.send().

Invio e Ricezione di File

Oltre ai messaggi di testo, la chat supporta anche l'invio di file. Quando l'utente seleziona un file tramite l'input dedicato, il file viene acquisito e un link di download viene generato all'interno della chat. Il file viene frammentato in pacchetti di dimensione pari a BYTES_PER_CHUNK e trasmesso attraverso il DataChannel.

Il trasferimento dei file presenta un problema quando si deve inviare un file di grandi dimensioni, poiché il browser ha un buffer massimo di 256 kB. Per gestire questa limitazione, i file più grandi vengono suddivisi in piccoli pacchetti da 64 kB e inviati sequenzialmente. Viene utilizzato il metodo arrayBuffer per dividere il file in un buffer array e un flag speciale segnala l'invio dell'ultimo chunk del file. Inoltre, viene trasmessa anche la dimensione totale del arrayBuffer.

Per garantire l'integrità e l'ordine dei dati, tutti i chunk devono arrivare e rispettare la sequenza corretta. Tuttavia, inviare i blocchi in sequenza potrebbe superare il buffer del canale di uscita. Per evitare questo problema, i blocchi vengono accodati e inviati progressivamente quando il buffer scende sotto una soglia specifica. Questo processo viene monitorato tramite l'evento bufferedamountlow.

Il processo di ricezione è gestito analizzando il contenuto dei messaggi in arrivo. Il lato ricevente imposta un array per memorizzare i blocchi ricevuti e utilizza un messaggio speciale (Channel.LAST_DATA_OF_FILE) per identificare la fine del trasferimento. Una volta ricevuti tutti i blocchi, questi vengono assemblati in un nuovo buffer array e inseriti in un Blob, che viene poi reso disponibile per il download tramite un link nella chat. Infine, il canale viene chiuso.

Gestione Videochiamata

Quando si avvia una videochiamata premendo il pulsante callButton, la sezione del video viene resa visibile, e i flussi video vengono assegnati agli elementi HTML. Vengono aggiornate anche le icone della webcam e del microfono.

Se si termina la chiamata, tutti i flussi vengono fermati, e gli elementi HTML vengono resettati. Inoltre, vengono aggiornate le icone per riflettere lo stato disattivato della webcam e del microfono.

Gli utenti possono attivare e disattivare la webcam e il microfono premendo i rispettivi pulsanti. Il sistema aggiorna automaticamente l'icona corrispondente per riflettere lo stato attuale e gestendo gestiscono l'attivazione e disattivazione delle tracce video e audio del peer locale. Se la webcam o il microfono sono attivi, vengono disattivati e viceversa.

Disconnessione dei Peer

Infine, è stata implementata una funzionalità per gestire la disconnessione tra i peer. Quando un utente clicca sul pulsante di disconnessione, viene inviato un messaggio attraverso il DataChannel per notificare l'interruzione della connessione. Il messaggio viene visualizzato nella chat e il canale dati, insieme alla peerConnection, viene chiuso. Per evitare interazioni non valide dopo la disconnessione, il pulsante di disconnessione viene disabilitato.

Questa soluzione garantisce una comunicazione efficace tra i peer, fornendo una gestione completa della videochiamata, della chat e del trasferimento file in un ambiente peer-to-peer basato su WebRTC e Firestore.

Distribuzione

Il **deployment** dell'applicazione su Firebase permette di distribuire il progetto su un hosting scalabile e accessibile via web. Dopo aver completato lo sviluppo e la configurazione dell'ambiente Firebase, il processo di distribuzione prevede la costruzione del progetto e il suo caricamento sui server Firebase. Grazie al comando firebase deploy, tutti i file statici vengono distribuiti nell'hosting Firebase, rendendo l'applicazione disponibile online. In caso di modifiche al codice, è sufficiente eseguire nuovamente npm run build seguito da firebase deploy per aggiornare la versione pubblicata.

Informazioni di installazione

Passaggi per l'installazione

1. Creare un account Firebase se non lo si possiede già.
2. Creare un nuovo progetto Firebase.
3. Recuperare le variabili di configurazione del progetto Firebase, definite come:

```
const firebaseConfig = {
apiKey: "Your API Key",
authDomain: "Your Auth Domain",
projectId: "Your Project ID",
storageBucket: "Your Storage Bucket",
messagingSenderId: "Your Messaging Sender ID",
appId: "Your App ID",
measurementId: "Your Measurement ID"
};
```

4. Clonare la repository GitHub:

```
git clone https://github.com/Manuel-Cplusplus/Rete-P2P.git
```

5. Inserire le variabili di configurazione all'interno di un file .env, seguendo l'esempio fornito nella repository.

Installazione e Avvio

Dopo aver clonato la repository, eseguire i seguenti comandi nella directory del progetto:

- npm install
- npm install -g firebase-tools
- firebase login

Per avviare il progetto in locale:

- npm run dev

Per costruire e distribuire il progetto su Firebase:

- npm run build
- firebase deploy

Il sistema di build utilizzato è **Vite**, usato per gestire l'ambiente di sviluppo e la build del progetto prima della distribuzione su Firebase.

Risoluzione Problemi

Se si verificano problemi durante la build, provare i seguenti passaggi:

1. Eliminare la cartella node_modules, la cartella dist e il file package-lock.json.
2. Eseguire nuovamente l'installazione dei pacchetti: npm install

Sviluppi Futuri

Abbiamo preso in considerazione l'implementazione di un sistema multi-peer, ma per realizzare questa soluzione non sarebbe stato possibile adottare un sistema peer-to-peer puro, come richiesto dal caso di studio.

Per integrare un sistema multi-peer, una possibile soluzione sarebbe l'adozione di una rete mesh, mantenendo però l'architettura peer-to-peer. Tuttavia, tale approccio risulterebbe inefficiente e scarsamente scalabile.

Pertanto, suggeriamo l'uso di servizi di terze parti con soluzioni gratuite, seppur limitate, come Agora.io. Questa piattaforma consente una gestione semplificata del sistema multi-peer e offre vantaggi in termini di ottimizzazione, grazie all'utilizzo di server intermedi (i server di Agora.io).

È importante sottolineare che questa soluzione non rispetta più il modello peer-to-peer puro, ma tale compromesso è necessario per garantire la scalabilità, come descritto nella sezione "Scelta Progettuale: Connessione fra 2 peer o multi-peer".

Un altro elemento da definire negli sviluppi futuri è quello di prevedere la possibilità di effettuare più connessioni peer-to-peer contemporaneamente.

Glossario

Acronimi

TERMINE	DESCRIZIONE
API	Application Programming Interface, insieme di strumenti e protocolli per lo sviluppo di software.
P2P	Peer-to-Peer, modello di rete in cui i nodi possono agire sia come client che come server.
MCU	Multipoint Control Unit
SFU	Selective Forwarding Unit
CPAAS	Communication Platform-as-a-Service
BAAS	Backend-as-a-Service
FCM	Firebase Cloud Messaging
SDK	Software Development Kit
WEBRTC	Web Real-Time Communication, tecnologia che consente la comunicazione audio, video e dati in tempo reale tra peer.
SDP	Session Description Protocol, protocollo per la negoziazione dei parametri di comunicazione tra peer.
STUN	Session Traversal Utilities for NAT, protocollo che aiuta un peer a scoprire il proprio indirizzo IP pubblico.
TURN	Traversal Using Relays around NAT, protocollo che permette la trasmissione di dati tramite un server relay quando la connessione diretta non è possibile.
ICE	Interactive Connectivity Establishment, framework per la gestione della connessione tra peer attraverso diversi percorsi di rete.
NAT	Network Address Translation
UDP	User Datagram Protocol, protocollo di trasmissione veloce ma non affidabile, utilizzato da WebRTC.

Definizioni

TERMINE	DESCRIZIONE
RETE PEER-TO-PEER	Modello architetturale di rete in cui i nodi possono agire contemporaneamente come client e server.
WEBRTC	Tecnologia che consente comunicazioni in tempo reale su browser e dispositivi mobili.
SERVER STUN	Server che aiuta un peer a ottenere il proprio indirizzo IP pubblico per facilitare la connessione P2P.
SERVER TURN	Server che funge da relay quando la connessione diretta tra peer non è possibile.
SERVER DI SIGNALING	Server utilizzato per lo scambio iniziale di informazioni tra peer affinché possano stabilire una connessione diretta.
ICE CANDIDATES	Indirizzi IP e porte disponibili che vengono utilizzati per determinare il percorso di connessione ottimale tra peer.
MESH NETWORK	Modello di rete in cui ogni peer si connette direttamente a tutti gli altri peer.
MULTIPOINT CONTROL UNIT	Un server centrale che raccoglie, elabora e inoltra i flussi dati tra peer.
SELECTIVE FORWARDING UNIT	Server intermedio che inoltra i flussi ai peer senza elaborarli.
FIREBASE	Piattaforma BaaS di Google che offre database, autenticazione e cloud storage.
AGORA.IO	Piattaforma CPaaS per streaming audio/video e comunicazioni in tempo reale.
BACKEND-AS-A-SERVICE	Modello di servizio cloud che fornisce funzionalità di backend senza la necessità di un'infrastruttura server-side.
COMMUNICATION PLATFORM-AS-A-SERVICE	Piattaforma che fornisce API per comunicazioni in tempo reale.
LATENZA	Tempo di ritardo nella trasmissione dei dati in una comunicazione di rete.
WEBSOCKETS	Protocollo di comunicazione bidirezionale tra client e server che mantiene una connessione persistente
CODEC	Algoritmo di compressione e decompressione utilizzato per la trasmissione di audio e video in WebRTC.

Riferimenti

Agora.io. (s.d.). *How developers build real-time experiences*. Tratto da https://www.agora.io/en/

BaaS-Wiki. (s.d.). *Backend as a service*. Tratto da https://it.wikipedia.org/wiki/Backend_as_a_service

Firebase. (s.d.). *Firebase Documentation*. Tratto da https://firebase.google.com/docs

Itnext. (s.d.). *WebRTC for beginners; How calls work from the outside!* Tratto da https://itnext.io/webrtc-for-beginners-how-it-all-works-from-the-outside-3c806f582229

Medium. (s.d.). *P2P connection with WebRTC*. Tratto da https://medium.com/@avocadi/p2p-connection-with-webrtc-8557461cae2d

PTP-Wikipedia. (s.d.). *Peer-to-peer*. Tratto da https://it.wikipedia.org/wiki/Peer-to-peer

WebRTC. (s.d.). *WebRTC*. Tratto da https://webrtc.org/?hl=it