UNIVERSITÀ DEGLI STUDI DI MILANO

FACOLTÀ DI SCIENZE E TECNOLOGIE

Corso di Laurea in Informatica per la comunicazione digitale

Streaming decentralizzato di contenuti audiovisivi

Relatore: Trentini Andrea

Tesi di Laurea di:
Mirko Milovanovic
Matricola: 870671

Anno Accademico 2023-2024

Questo lavoro è dedicato ai miei genitori,
a mia sorella, suo marito e i miei nipoti.

"Do you see how you've grown? Don't be sorry then"

– Porter Robinson

Ringraziamenti

Desidero esprimere la mia più sincera gratitudine a tutte le persone che hanno contribuito, direttamente o indirettamente, alla realizzazione di questa tesi.

Innanzitutto, ringrazio il Professor Andrea Trentini per la sua guida esperta, i preziosi consigli e il supporto costante durante tutto il percorso di ricerca e stesura di questo lavoro.

Un ringraziamento speciale va alla comunità di PeerTube e Framasoft, non solo per aver creato una piattaforma innovativa che è stata oggetto di questo studio, ma anche per il loro impegno nel promuovere soluzioni open source e decentralizzate.

Vorrei esprimere la mia gratitudine tutti gli amici e colleghi volontari dell'Osservatorio Astronomico 'Giovanni Virginio Schiaparelli' che mi hanno accompagnato durante il percorso universitario, per il loro supporto morale, l'incoraggiamento e i momenti di confronto che hanno arricchito questa esperienza.

Un profondo ringraziamento va alla mia famiglia, per il loro incondizionato sostegno e la enorme pazienza dimostrata durante tutto il mio percorso accademico. Senza il loro supporto, questo traguardo non sarebbe stato possibile.

Infine, un pensiero di gratitudine a tutti coloro che, pur non essendo citati esplicitamente, hanno contribuito in vari modi alla realizzazione di questo lavoro.

Indice

Capitolo 1

Introduzione

Fin dai suoi albori Internet, nato da ARPAnet, svolge un ruolo importantissimo nella vita di tutti noi, che sia per condividere informazioni militari, utilizzare servizi erogati via Web o, come sta accadendo in questi ultimi anni, guardare contenuti audiovisivi "on-the-go", connettendo cioè le persone intorno al globo nei più svariati modi possibili attraverso l'utilizzo di calcolatori come computer e dispositivi mobili "connessi".

Negli anni, e per come è stato concepito Internet stesso architetturalmente, si sono venuti a formare veri metodi di comunicazione tra gli utilizzatori e gli erogatori di servizi Internet, più o meno sicuri, efficaci o resilienti rispetto a tematiche come la privacy dei dati, la centralizzazione e costi di operazione.

Il modello Client/Server è certamente il più diffuso, reso famoso dall'imponente World Wide Web, il modo più intuitivo di utilizzare Internet ma certamente non l'unico[44]. Questa architettura, in particolare nell'ambito della trasmissione di contenuti audiovisivi in streaming, presenta notevoli sfide tecniche ed economiche. L'infrastruttura centralizzata richiede server potenti e costosi che devono gestire simultaneamente migliaia di connessioni, con un consumo di banda che cresce linearmente con il numero di spettatori.

Negli ultimi anni, diverse piattaforme hanno cercato di risolvere questo problema attraverso l'adozione di tecnologie peer-to-peer (P2P) per la distribuzione dei contenuti. Tra queste, PeerTube si distingue come una delle soluzioni più promettenti, combinando una struttura federata con tecnologie di streaming P2P che promettono di ridurre in modo significativo il carico sui server centrali.

Questa tesi si propone di analizzare criticamente le affermazioni fatte dagli sviluppatori di PeerTube riguardo l'efficienza e l'efficacia del loro approccio P2P nel contesto dello streaming video. Partiremo da un'analisi del panorama attuale dello streaming centralizzato, ne evidenzieremo i limiti per poi esplorare le alternative decentralizzate e concentrarci su PeerTube, ricreando e ampliando i test condotti dai suoi sviluppatori per verificarne empiricamente le prestazioni.

1.1 Il problema dello streaming centralizzato

Un sistema Client/Server tradizionale per lo streaming video presenta diverse criticità:

- **Scalabilità limitata**: I server centrali devono gestire tutto il traffico in uscita, con costi di banda che crescono linearmente con il numero di spettatori.
- **Single Point of Failure**: Se il server centrale subisce un'interruzione, tutti i client perdono l'accesso al servizio.
- **Costi infrastrutturali elevati**: Mantenere server capaci di gestire picchi di traffico richiede investimenti significativi.
- **Centralizzazione del controllo**: Poche entità controllano le piattaforme più popolari, con potenziali implicazioni per la libertà di espressione e la privacy.

1.2 Le alternative decentralizzate e P2P

Le tecnologie peer-to-peer offrono un approccio alternativo che potrebbe risolvere molti di questi problemi:

- **Distribuzione del carico**: Ogni client può anche fungere da distributore per altri client, riducendo il carico sul server originale.
- **Maggiore resilienza**: L'assenza di un punto centrale di fallimento rende il sistema più robusto.
- **Riduzione dei costi**: I costi di banda vengono distribuiti tra i partecipanti anziché essere sostenuti da un unico fornitore.
- **Decentralizzazione del controllo**: Nessuna entità singola può controllare totalmente il flusso di informazioni.

Tra le varie piattaforme che utilizzano questo approccio, PeerTube ha attirato la nostra attenzione per la sua architettura ibrida che combina federazione e P2P, e per le affermazioni dei suoi sviluppatori riguardo l'efficienza di questo approccio. In particolare, un articolo pubblicato dal team di PeerTube sostiene che la loro implementazione P2P può ridurre il carico sul server di origine fino all'80%, una cifra che merita un'analisi approfondita e una verifica indipendente.

Capitolo 2

Stato dell'arte

In questo capitolo analizzeremo il panorama attuale delle tecnologie di streaming, confrontando approcci centralizzati e decentralizzati per comprendere i loro punti di forza e debolezza.

2.1 Modelli di comunicazione in rete

2.1.1 Client/Server e il Web 2.0

Un sistema Client/Server è un tipo di computazione distribuita in cui i clienti effettuano delle richieste verso un server che, a sua volta, risponde con i dati/servizi richiesti restando in attesa. I client possono essere di vario tipo e trovarsi ovunque nel globo e, in generale, integrano una parte hardware (smartphone, PCs, ...) a una software (applicazioni GUI, web browser, ...). I server sono invece macchine specializzate spesso raggruppate assieme in grandi data center, interconnesse tra di loro per fornire uno o più servizi a molteplici client contemporaneamente.

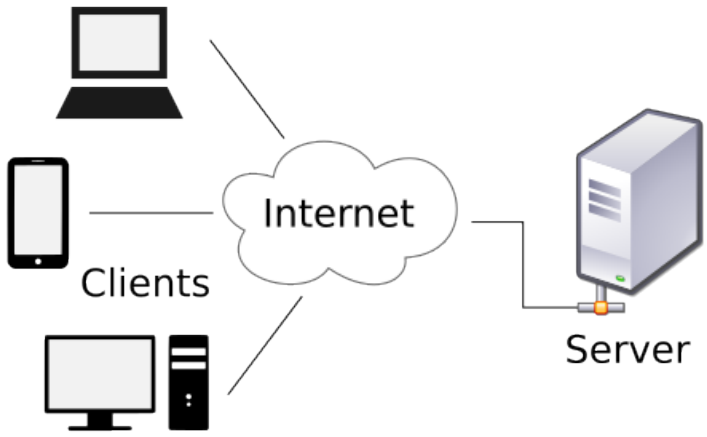

Figura 2.1. Modello Client/Server tradizionale [43].

Questo modello ha dominato l'era del Web 2.0, caratterizzata da:

- Piattaforme centralizzate con contenuti generati dagli utenti
- Elevata interattività e social networking
- Controllo dei dati da parte delle grandi aziende tecnologiche
- Monetizzazione tramite pubblicità e raccolta dati

Nonostante i vantaggi in termini di semplicità d'uso e accessibilità, questo modello ha portato a:

- Forte centralizzazione del potere da parte di poche "Big Tech"
- Elevati costi di infrastruttura
- Problemi di privacy e controllo dei dati personali
- Vulnerabilità a censura e manipolazione dei contenuti

2.1.2 Peer-to-Peer

Il modello peer-to-peer rappresenta un approccio alternativo in cui ogni nodo della rete può fungere sia da client che da server, eliminando la necessità di un'infrastruttura centralizzata. Questo modello è alla base di molte tecnologie del cosiddetto Web 3.0, caratterizzato da:

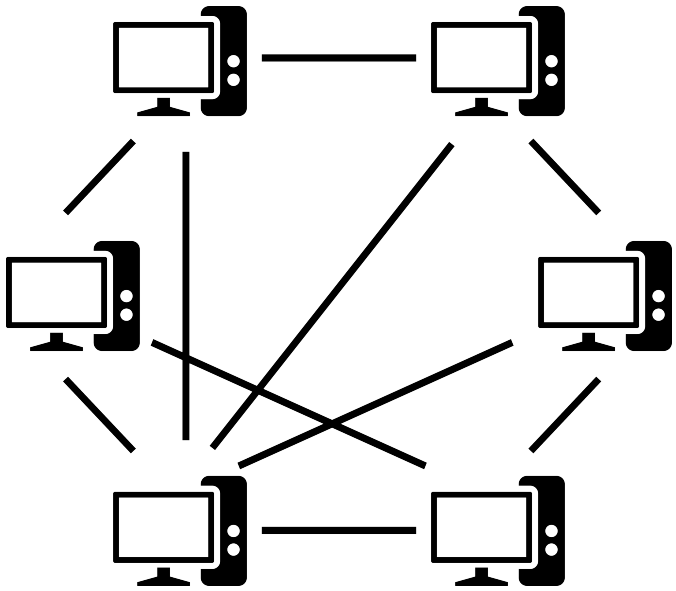

Figura 2.2. Modello P2P [12].

- Decentralizzazione del controllo e distribuzione del carico
- Maggiore privacy e sovranità sui dati personali
- Resilienza a censura e fallimenti di singoli nodi
- Riduzione dei costi infrastrutturali

Tuttavia, le soluzioni P2P presentano anche alcune sfide:

- Maggiore complessità tecnica
- Possibili problemi di latenza e qualità del servizio
- Difficoltà nel garantire contenuti legali e moderazione
- Sfide tecniche come il NAT traversal

2.2 Modelli di distribuzione dei dati in rete

La distribuzione dei dati nelle reti di computer può seguire diversi modelli, ciascuno con caratteristiche specifiche e casi d'uso ottimali. I tre principali modelli di distribuzione sono: unicast, multicast e broadcast. Comprendere questi modelli è essenziale per analizzare le soluzioni di streaming video, poiché ciascuno offre vantaggi e svantaggi in termini di efficienza, scalabilità e facilità di implementazione.

2.2.1 Unicast

Il modello unicast consiste nella trasmissione di dati da un mittente a un singolo destinatario specifico. Questo è il modello predominante su Internet e viene utilizzato dalla maggior parte dei servizi di streaming video tradizionali.

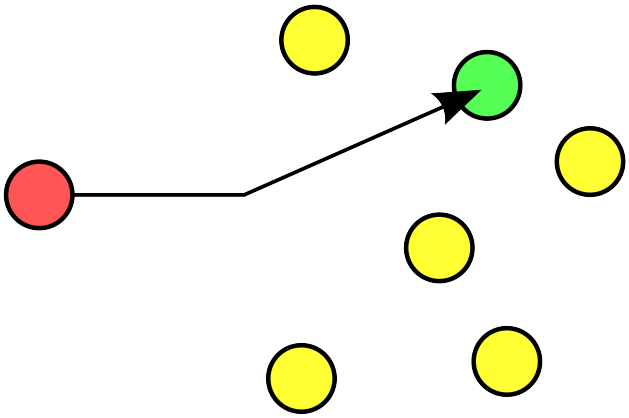

Figura 2.3. Modello di comunicazione Unicast.

Caratteristiche principali:

- **Connessione uno-a-uno:** Ogni client stabilisce una connessione dedicata con il server.
- **Consumo di banda:** La banda richiesta sul server aumenta linearmente con il numero di client.
- **Controllo individuale:** Ogni client può controllare indipendentemente la riproduzione (pausa, riavvolgimento, ecc.).
- **Personalizzazione:** Permette contenuti e pubblicità personalizzati per ogni utente.

Implicazioni per lo streaming video: Il modello unicast è semplice da implementare e garantisce un elevato controllo sull'esperienza utente, ma diventa estremamente inefficiente quando molti utenti richiedono lo stesso contenuto contemporaneamente. Un server che trasmette un video in HD a 1000 spettatori deve inviare 1000 copie identiche dello stesso flusso dati, con conseguente consumo elevato di banda e risorse del server [53].

2.2.2 Multicast

Il multicast è un modello di distribuzione in cui i dati vengono inviati a un gruppo specifico di destinatari simultaneamente. Anziché inviare più copie dei dati, il server invia una singola copia che viene replicata dai router di rete solo quando necessario.

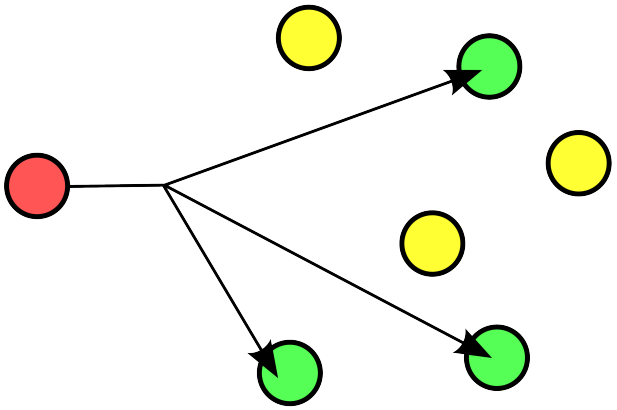

Figura 2.4. Modello di comunicazione Multicast.

Caratteristiche principali:

- **Efficienza della banda:** Indipendentemente dal numero di ricevitori, il traffico generato rimane costante fino ai punti di biforcazione nella rete.
- **Gruppi di destinazione:** I client si iscrivono a gruppi multicast specifici per ricevere determinati flussi di dati.
- **Supporto dell'infrastruttura:** Richiede router compatibili con il protocollo Internet Group Management Protocol (IGMP).
- **Controllo limitato:** Tutti i client ricevono lo stesso flusso contemporaneamente, limitando le opzioni di controllo individuale.

Implicazioni per lo streaming video: Il multicast è teoricamente ideale per eventi live con molti spettatori, poiché riduce drasticamente il carico sui server e sulla rete backbone. Tuttavia, la sua adozione su Internet pubblico è limitata perché molti ISP e router domestici non supportano pienamente il routing multicast. È più comunemente utilizzato in reti aziendali, IPTV e altre reti gestite [50].

2.2.3 Broadcast

Il broadcast consiste nell'invio di dati a tutti i dispositivi all'interno di una rete. A differenza del multicast, che indirizza un gruppo specifico, il broadcast raggiunge indiscriminatamente tutti i nodi in un dominio di broadcast.

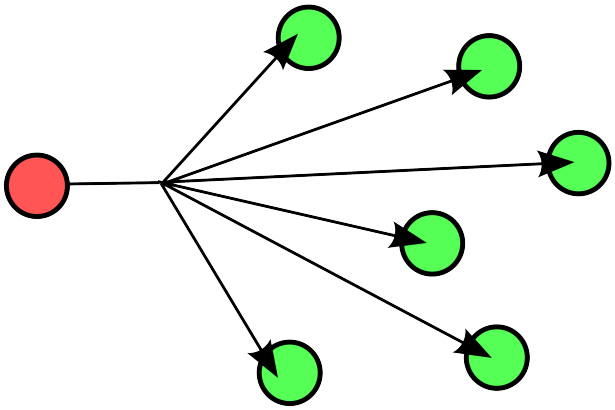

Figura 2.5. Modello di comunicazione Broadcast.

Caratteristiche principali:

- **Copertura totale:** Raggiunge tutti i dispositivi nella rete locale.
- **Nessuna configurazione client:** I destinatari non devono iscriversi o configurare nulla per ricevere i dati.
- **Limitazioni di scala:** Generalmente limitato a singoli segmenti di rete a causa dei problemi di congestione che potrebbe causare.
- **Utilizzo inefficiente:** Molti destinatari potrebbero non essere interessati ai dati trasmessi.

Nelle nostre analisi, andremo a considerare solo soluzioni Unicast in quanto si tratta del modello più diffuso e utilizzato, anche perché le tecnologie di streaming che analizzeremo sono tutte implementate a livello applicativo dello stack TCP/IP e quindi operano naturalmente in modalità unicast.

Implicazioni per lo streaming video: Il broadcast è raramente utilizzato per lo streaming video su Internet, principalmente a causa delle sue limitazioni di scala e dell'inefficienza. È più adatto per scopi come la scoperta di dispositivi

in reti locali o la trasmissione televisiva tradizionale. Nel contesto delle reti IP, il broadcast è generalmente considerato obsoleto per applicazioni di streaming a favore di approcci più mirati come multicast o tecnologie come CDN e P2P [48].

2.2.4 Confronto e applicazioni nello streaming video

Tabella 2.1: Confronto tra modelli di distribuzione dei dati per lo streaming video.

Caratteristica	**Unicast**	**Multicast**	**Broadcast**
Scalabilità	Limitata	Eccellente	Molto limitata
Controllo utente	Completo	Limitato	Assente
Supporto Internet	Universale	Limitato	Non utilizzato
Efficienza larghezza di banda	Bassa	Alta	Molto bassa
Personalizzazione contenuti	Possibile	Difficile	Impossibile
Implementazione CDN	Semplice	Complessa	Non applicabile

Le soluzioni di streaming decentralizzate come PeerTube cercano di combinare i vantaggi del multicast (efficienza della banda) con l'universale supporto dell'unicast, utilizzando connessioni peer-to-peer per distribuire il carico. Questo approccio "multicast simulato" permette di raggiungere un'efficienza simile al multicast nativo senza richiedere supporto specifico da parte dell'infrastruttura di rete.

Le applicazioni CDN (Content Delivery Network) rappresentano un altro approccio per migliorare l'efficienza dell'unicast, posizionando copie dei contenuti più vicino agli utenti finali, ma senza modificare il modello di comunicazione di base [49].

Nell'evoluzione dello streaming video, stiamo assistendo a un crescente interesse verso soluzioni ibride che combinano questi diversi modelli per ottimizzare l'esperienza utente, l'efficienza della rete e i costi operativi.

2.3 Casi d'uso per lo streaming decentralizzato

Vediamo degli esempi di come questa interazione potrebbe essere svolta:

Esempio d'interazione "one to many"	
Use case:	Un utente davanti al proprio computer vorrebbe condividere quello che vede sullo schermo con i propri amici o followers
Soggetti:	Utente principale, viewers, computer, connessione internet
Obiettivi:	Condivisione in live streaming di un contenuto a schermo via internet con TCP/IP

Use case 2.1

Esempio d'interazione "many to many"	
Use case:	Più utenti al proprio computer vorrebbero comunicare e interagire tra di loro contemporaneamente, come una conference-call
Soggetti:	Utenti multipli, computer, connessione internet, microfono, telecamera
Obiettivi:	Live streaming e interazione real-time tra utenti via internet con TCP/IP

Use case 2.2

Esempio d'interazione "one to many"	
Use case:	Un'azienda, per questioni legate alla sicurezza sul lavoro, si ritrova con la necessitò di dover fare dei "workshop" in diretta ai propri dipendenti con diverse locazioni sparse per il nel mondo senza utilizzare però grandi servizi cloud, dato l'elevato costo di banda e di noleggio del servizio per singolo utente finale (in questo caso il singolo dipendente)
Soggetti:	Dipendenti, azienda
Obiettivi:	Video streaming dei "workshop" per i dipendenti sparsi per il mondo

Use case 2.3

Esempio d'interazione "one to many"	
Use case:	Degli amici devono svolgere un progetto universitario assieme e quindi interagire tra di loro facendo pair programming condividendo lo schermo gli uni con gli altri. Per questioni di privacy e sicurezza non vogliono utilizzare un servizio pubblico come Discord in quanto vorrebbero tenere tutto segreto fino al giorno della presentazione
Soggetti:	Studenti
Obiettivi:	Pair programming

Use case 2.4

Esempio d'interazione "one to many"	
Use case:	Una casa produttrice di film emergente vuole condividere i nuovi film in produzione con dei trailer ma a causa di dispute legate a DRM, copyright e content strike di altre aziende più grandi, non vuole utilizzare dei servizi già esistenti con EULA molto restringenti ma vuole avere il controllo dei propri diritti sul contenuto che ha creato
Soggetti:	Filmmakers, appassionati di film
Obiettivi:	Condivisione degli ultimi trailer prodotti dalla casa

Use case 2.5

Esempio d'interazione "one to many"	
Use case:	Un gruppo di amici vuole fare una serata di gioco online e ognuno di loro vuole condividere il proprio schermo con gli altri giocatori per giocare assieme
Soggetti:	Gamer
Obiettivi:	Giocare assieme tramite condivisione dello schermo

Use case 2.6

L'analisi dei casi d'uso evidenzia la necessità di una soluzione di streaming decentralizzata e a licenza libera, capace di operare in modalità uno-a-molti o molti-a-molti, senza affidarsi a servizi cloud proprietari e senza richiedere il deploy di una complessa infrastruttura hardware globale. I requisiti fondamentali, soprattutto in relazione all'elevato fabbisogno di banda tipico dei contenuti video, si declinano nei seguenti scenari applicativi:

Commerciale:

- Streaming di eventi o riunioni interne,
- Assistenza remota,
- Conference calls,
- Possibilità di operare sia come utente che come provider/distributore,
- Ottimizzazione dei costi e del consumo della banda.

Educazionale:

- Trasmissione in diretta di corsi e lezioni.

Sociale:

- Organizzazione di lan party,
- Supporto per streaming di gaming e attività live da parte di streamer,
- Conference calls in piccoli gruppi,
- Impiego per telecamere di sicurezza per questioni di privacy,
- Moderazione dei contenuti online.

Istituzionale:

- Comunicazioni video in situazioni di emergenza.

2.4 Panoramica delle piattaforme di streaming esistenti

2.4.1 Soluzioni centralizzate

Twitch

Twitch è una piattaforma di live streaming di proprietà di Amazon lanciata nel 2011, originariamente specializzata nella trasmissione in diretta di videogiochi, eSports ed eventi riguardanti il mondo videoludico.

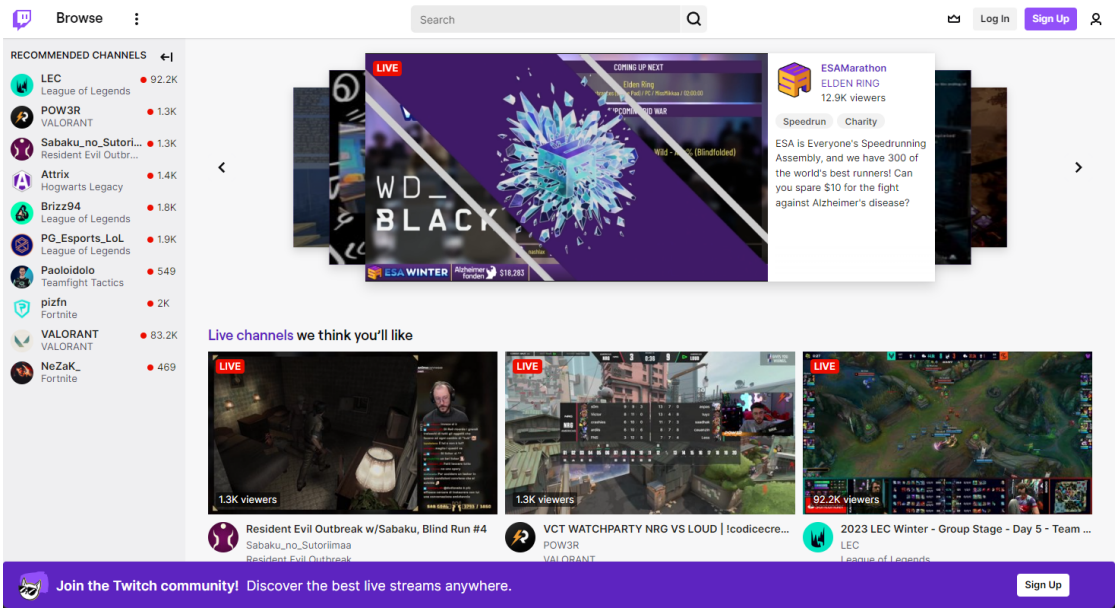

Figure 2.6. Home page di Twitch.

Una delle più grandi e famose, la piattaforma è stata originariamente sviluppata come una controparte di streaming per Justin.tv, una piattaforma di streaming generale. Tuttavia, Twitch si è concentrata esclusivamente sui contenuti di videogiochi e di eSports, anche se continua tutt'ora a essere la scelta "de-facto" per molti utenti non solo per lo streaming di contenuti videoludici ma anche per lo "streaming general purpose" come per esempio lo streaming di musica, arte, cucina e di varie attività creative.

Negli anni successivi, Twitch ha visto una rapida crescita e ha attirato una vasta gamma di creatori di contenuti, dalle grandi organizzazioni di eSports ai singoli streamer indipendenti, fino ad arrivare ad oggi con una base rispettivamente di circa 3 milioni di viewers e 1.5 milioni di broadcaster giornalieri.

Twitch offre molti servizi sotto un unico sito web accessibile con un semplice browser:

Caratteristiche	
Partecipanti:	Chiunque con un account può condividere e guardare
Architettura software:	Centralizzata
Architettura hardware:	Servizio accessibile via browser o applicazione mobile
Tipo di dato condiviso:	Stream video e testo per funzionalità di live chat
Licenza:	Proprietaria con contratto di utilizzo 'EULA'

YouTube

Youtube è una piattaforma di sharing online di video statici, di proprietà di Google, specializzata nella condivisione globale di video generici con funzionalità aggiuntive di social media, monetizzazione e live streaming.

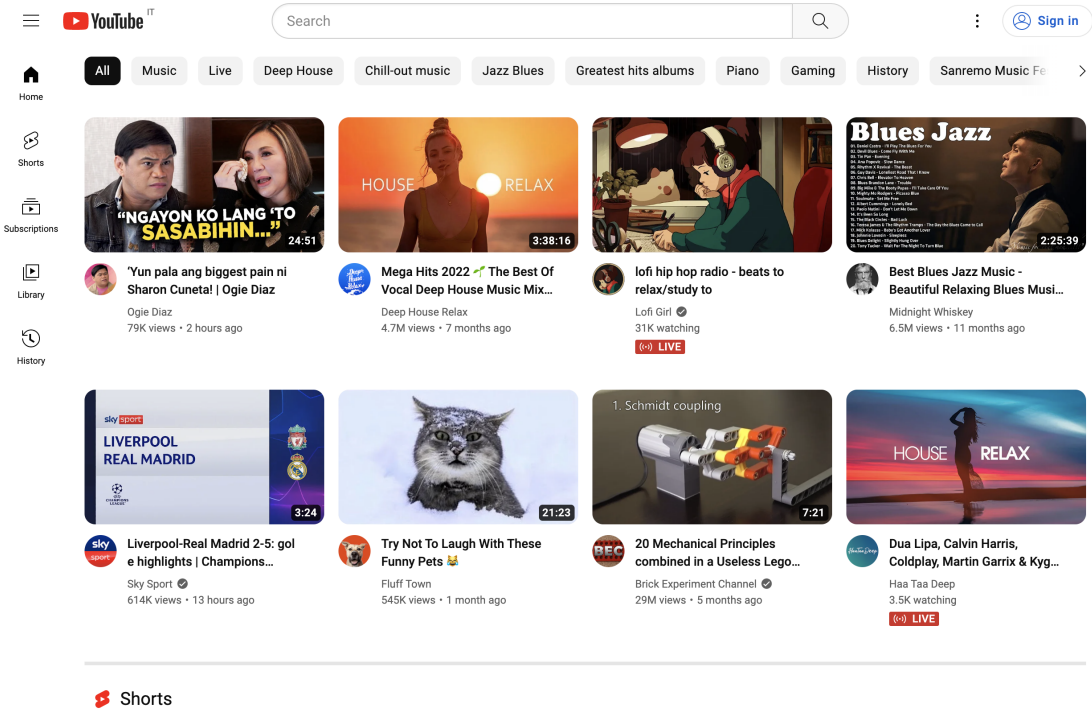

Figure 2.7. Home page di Youtube.

La compagnia è stata lanciata nel febbraio 2005 come indipendente ed è poi stata successivamente acquisita da Google.
Al momento è considerata la piattaforma più grande e longeva di questo segmento, con più di 2.5 miliardi di utenti mensili e milioni di ore di video condivisi ogni giorno. Dall'acquisizione, YouTube ha espanso la propria offerta al di fuori della sola condivisione di video "amatoriali", includendo contenuti come film di produzione professionale, video musicali ufficiali, documentari, news, ecc.
Ha anche integrato la piattaforma pubblicitaria chiamata 'AdSense', sempre di proprietà di Google, permettendo a tutti gli utenti amatoriali o professionali approvati di poter ricevere un ricavo economico dalle pubblicità e dal marketing presenti sul sito.

Caratteristiche	
Partecipanti:	Chiunque sul web
Architettura software:	Servizio accessibile via browser o applicazione mobile
Architettura hardware:	Centralizzata con utilizzo di CDN
Tipo di dato condiviso:	Video statici, live stream
Licenza:	Proprietaria con contratto di utilizzo 'EULA'

2.4.2 Soluzioni decentralizzate e P2P

Tahoe-LAFS

Tahoe-LAFS è un sistema di file distribuito, open source, decentralizzato e sicuro, che permette agli utenti di memorizzare e condividere file in un ambiente di rete distribuita chiamata 'Grid'. Sviluppato da un gruppo di ricercatori dell'Università di Maryland, dal 2007 è stato rilasciato come software open source sotto licenza GPL. Tahoe-LAFS è stato progettato per essere un sistema di file distribuito sicuro, affidabile e scalabile, in modo da poter essere utilizzato in ambienti di rete distribuita, dato che promette di essere un sistema 'provider-independent': i fornitori dei server intermediari non hanno mai la possibilità di accedere o modificare i dati memorizzati dagli utenti finali perché non sono loro a garantire la confidenzialità, l'integrità o l'assoluta disponibilità dei dati, ma sono i client finali a farlo.

Tahoe-LAFS è essenzialmente un sistema di archiviazione chiave-valore. L'archivio utilizza delle corte stringhe, circa 100 byte, chiamate *capabilities* come chiavi e dati arbitrari come valori.

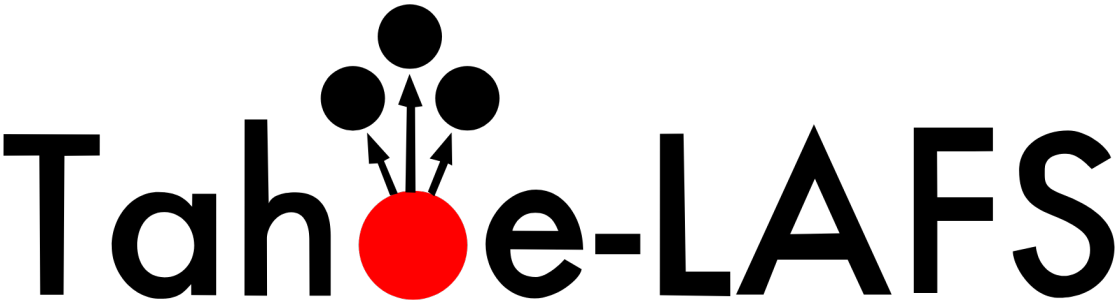

Figure 2.8. Tahoe-LAFS logo.

Queste 'capabilities' posso essere condivise per dare agli altri accesso a determinati valori sulla 'Grid'. Ad esempio, si può dare la possibilità di lettura a un amico e conservare il permesso di scrittura per se stessi. Per eliminare un valore, basta dimenticare (cioè eliminare) la chiave della 'capability': una volta fatto, sarà impossibile recuperare i dati. I server di archiviazione hanno però un modo per fare 'garbage-collection' da condivisioni non referenziate.

In aggiunta al sistema chiave-valore, viene affiancato un livello file-storage classico, che consente di condividere sotto-directory con altri utenti senza, ad esempio, rivelare l'esistenza o il contenuto delle directory principali.

Come indicato poco sopra, sono i clienti a garantire l'integrità e la confidenzialità dei dati. Questo viene realizzato grazie alla crittografia che viene eseguita su ogni 'capability' prima di essere caricata sul 'Grid'. Ogni valore viene prima crittografato con una chiave asimmetrica e poi suddiviso in parti più piccole, e più maneggiabili. Questi segmenti diventano poi effettivamente gli 'share' che verranno memorizzati nei nodi della rete che, ricordiamo, svolgono solo la funzione di memorizzazione dei dati, gli utenti non si affidano a loro per altro [1] [41].

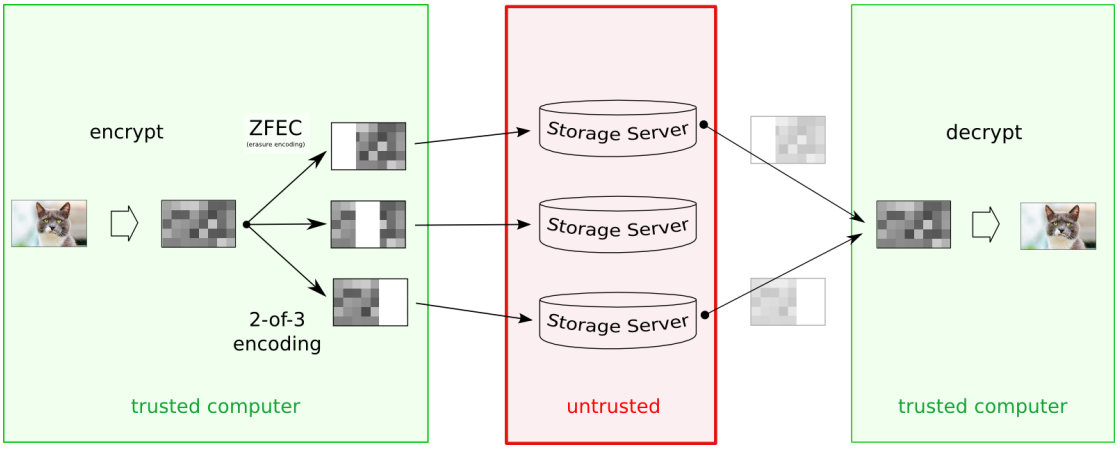

Figure 2.9. Tahoe-LAFS simple data flow [41].

Caratteristiche	
Partecipanti:	Utenti e server di terze parti
Architettura software:	Software client lato utente
Architettura hardware:	Decentralizzata e 'provider-indipendent'
Tipo di dato condiviso:	Qualsiasi tipo di file binario
Licenza:	GPL

IPFS

IPFS o 'InterPlanetary File System' è una suite di protocolli e librerie open source per la condivisione di file in un ambiente di rete distribuita utilizzando meccanismi di 'content-addressing'. Sviluppato nel 2014 da Protocol Labs, un'azienda di ricerca e sviluppo.

IPFS punta a creare un file system condiviso da una rete di nodi decentralizzata che comunica attraverso P2P, dove i singoli file sono organizzati come blocchi indipendentemente individuabili e immutabili (ovvero che non possono essere modificati ma solo aggiunti o eliminati) con degli identificatori chiamati 'CID (Content identifiers)' i quali sono memorizzati in un database distribuito chiamato 'DHT (Distributed Hash Table)' che viene condiviso con ogni singolo nodo della rete al fine di facilitare il routing dei dati attraverso essa stessa.

Uno dei problemi di IPFS è che, per suo stesso design, ogni singolo file è visibile a tutti i nodi della rete e quindi non è possibile creare un sistema di condivisione di file privati, ma solo di file pubblici che, se lo si desidera, possono essere crittografati, impedendo la lettura dei dati da parte di attori terzi [45] [6].

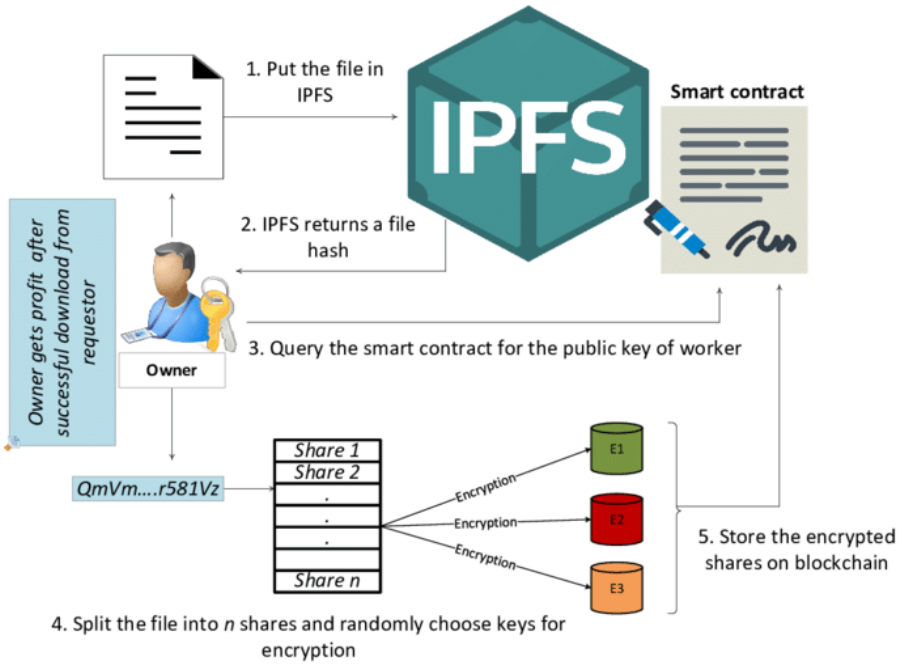

Figure 2.10. IPFS data flow [29].

Caratteristiche	
Partecipanti:	Nodi della rete
Architettura software:	Protocolli e software lato client
Architettura hardware:	Decentralizzata simil Blockchain
Tipo di dato condiviso:	Qualsiasi tipo di file binario
Licenza:	MIT

PeerTube

PeerTube è un servizio di video sharing open source, federalizzato e decentralizzato (lato client), basato su protocolli peer-to-peer come 'WebTorrent', 'WebRTC' e altre tecnologie web standard. Fa parte del cosiddetto 'Fediverse', un insieme

di server interconnessi che comunicano tramite ActivityPub, un protocollo di comunicazione aperto per creare reti federate. Inizialmente creato come una piattaforma di video sharing per i creatori di contenuti indipendenti, è stato progettato per essere estensibile e adattabile a qualsiasi tipo di contenuto video che rispetti i termini delle singole istanze che vengono messe a disposizione pubblicamente.

Il funzionamento è simile ad altre piattaforme video tipo (YouTube, Vimeo, Dailymotion, ecc.) con supporto per video statici e livestream, con la differenza che i video possono essere memorizzati e condivisi non solo dalle singole istanze della federazione attraverso il normale HTTP ma anche tra i client finali usando P2P per alleggerire il carico di banda. PeerTube è iniziato come progetto indipendente da un singolo sviluppatore che poi è stato affiancato dalla no-profit Framasoft [17] [47] [3].

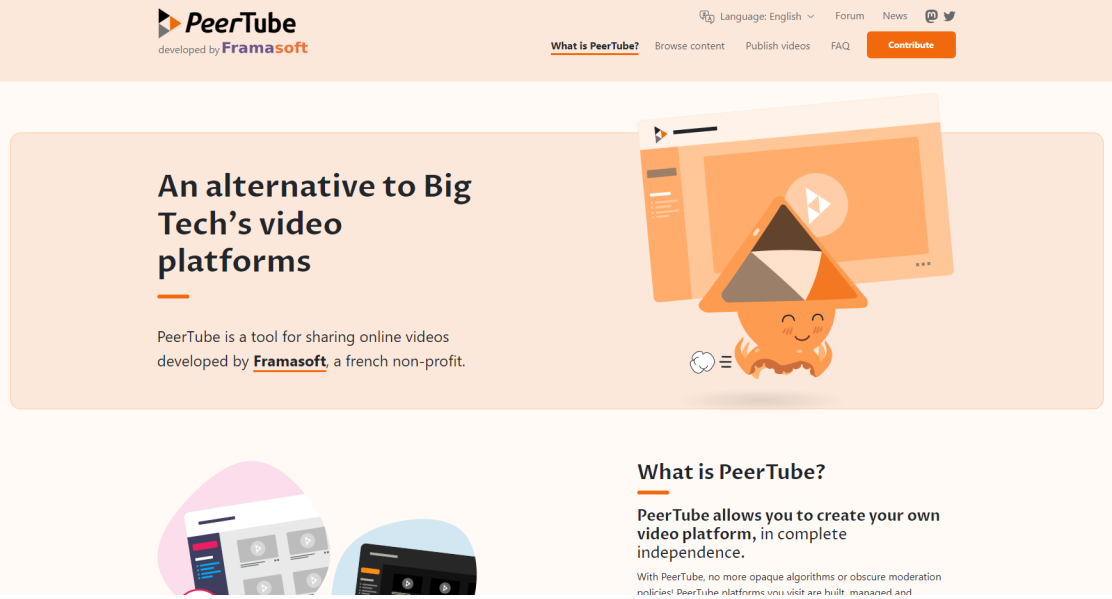

Figure 2.11. PeerTube.

Caratteristiche	
Partecipanti:	Utenti registrati e non delle singole istanze
Architettura software:	Browser web e server indipendenti
Architettura hardware:	Federalizzata e decentralizzata P2P
Tipo di dato condiviso:	Video statici e livestream
Licenza:	AGPLv3+

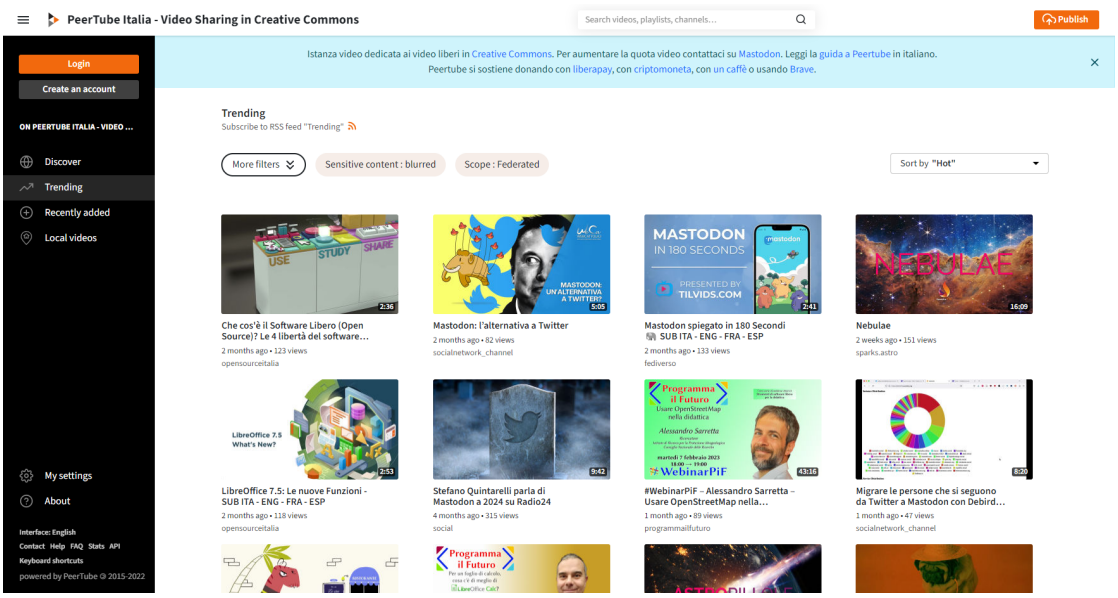

Figure 2.12. Una istanza di PeerTube Italiana.

Ace Stream

Ace Stream è un software proprietario di video streaming basato su P2P, sviluppato da un gruppo di sviluppatori russi, che permette agli utenti di trasmettere in diretta video e audio, riprodurre file multimediali, ricevere e condividere contenuti

in tempo reale. Il software è stato rilasciato per la prima volta nel 2013 e da allora è stato scaricato più di 100 milioni di volte. Ace Stream offre una vasta gamma di funzionalità che lo rendono popolare tra gli utenti che desiderano guardare contenuti in streaming. Utilizzando la tecnologia P2P, il software consente agli utenti di trasmettere video e audio in tempo reale, senza dover attendere il completamento del download. Questo significa che gli utenti possono godere di una riproduzione fluida e senza interruzioni, anche con connessioni internet più lente.

Figure 2.13. PeerTube.

Inoltre, Ace Stream supporta la riproduzione di file multimediali locali, consentendo agli utenti di guardare film, serie TV e altri contenuti salvati sul proprio dispositivo. Il software supporta una vasta gamma di formati video e audio, garantendo un'alta compatibilità.

Un'altra caratteristica interessante di Ace Stream è la possibilità di ricevere e condividere contenuti in tempo reale. Gli utenti possono creare i propri canali di streaming e condividere i propri contenuti con altri utenti. Questo rende il software ideale per esempio per gli appassionati di sport, che possono trasmettere eventi sportivi in diretta e condividerli con altri utenti.

Ace Stream è disponibile per diverse piattaforme, tra cui Windows, Mac, Linux e Android: ciò lo rende accessibile a un vasto pubblico. Inoltre, il software è gratuito da scaricare e utilizzare, rendendolo ancora più attraente per gli utenti.

In conclusione, Ace Stream è un software potente e versatile per lo streaming di video e audio. Grazie alla sua tecnologia P2P, offre una riproduzione fluida e senza interruzioni, consentendo agli utenti di godersi i propri contenuti preferiti in tempo reale. Con la possibilità di condividere contenuti e creare canali di streaming, Ace

Stream offre un'esperienza di streaming interattiva e coinvolgente [5] [2].

Caratteristiche	
Partecipanti:	Utenti registrati e non delle singole istanze
Architettura software:	Browser web e server indipendenti
Architettura hardware:	Federalizzata e decentralizzata P2P
Tipo di dato condiviso:	Video statici e livestream
Licenza:	AGPLv3+

Capitolo 3

PeerTube: architettura e tecnologie

Tra i vari progetti open source che si occupano di decentralizzazione e P2P, PeerTube è uno dei più interessanti e promettenti. In questo capitolo, esamineremo l'architettura e le tecnologie alla base di PeerTube, analizzando come il progetto si è evoluto nel corso degli anni e quali sono le sue caratteristiche principali.

3.1 Storia ed evoluzione del progetto

PeerTube è nato nel 2017 come progetto di un singolo sviluppatore di nome *Chocobozzz* e poi supportato dall'organizzazione non profit francese Framasoft, con l'obiettivo di creare un'alternativa libera e decentralizzata a piattaforme come YouTube. Da allora, ha visto una crescita costante sia in termini di funzionalità che di adozione. Essendo una piattaforma di tipo ActivityPub, PeerTube fa parte del cosiddetto Fediverse, un network federato di piattaforme interconnesse. Questo significa che gli utenti di PeerTube possono interagire con utenti di altre piattaforme compatibili con ActivityPub, come Mastodon, Pleroma e Pixelfed[47].

Ogni istanza di PeerTube fornisce un sito web, del tutto simile ad altre piattaforme come YouTube, per navigare e guardare video ed è, per impostazione predefinita, indipendente dalle altre in termini di aspetto, funzionalità e regole.

Diverse istanze con regole comuni (per esempio, consentire contenuti simili o richiedere la registrazione) possono formare federazioni, in cui seguono i video di altre istanze, pur mantenendo i contenuti archiviati solo sull'istanza che li ha pubblicati.

Le federazioni sono indipendenti tra loro e asimmetriche: un'istanza può seguire un'altra per visualizzarne i video, senza reciprocità necessarie. Gli amministratori

delle istanze possono scegliere di duplicare singoli video o intere istanze amiche, incentivando così la creazione di comunità con larghezza di banda condivisa.

Alcune tappe importanti nella storia di PeerTube includono [52]:

- **2017**: Lancio del progetto da parte di *Chocobozzz*
- **2018**: Supporto da parte di Framasoft e lancio della versione 1.0
- **2021**: Lancio della versione 3.0 con introduzione del supporto per i livestream e miglioramenti all'interfaccia utente
- **2021**: Introduzione di funzionalità di moderazione avanzate e supporto per i sottotitoli
- **2022**: Il Garante Europeo della Protezione dei Dati (EDPS) lancia EU Video, una piattaforma video basata su PeerTube per le istituzioni, gli enti e le agenzie dell'Unione Europea, come progetto pilota su ActivityPub. Viene chiuso ufficialmente nel 2024.

3.2 Stack tecnologico

PeerTube combina diverse tecnologie per fornire una piattaforma di streaming video decentralizzata e federata. La sua architettura si basa su un mix di tecnologie web moderne, tra cui Node.js, PostgreSQL, WebRTC etc.

Interessante caratteristica è che PeerTube è progettato per essere estensibile: consente cioè agli sviluppatori e admin di aggiungere nuove funzionalità e miglioramenti senza dover modificare il codice sorgente principale. Questo approccio facilita la creazione di plugin e moduli personalizzati, che possono essere utilizzati per estendere le funzionalità della piattaforma. Per esempio, di default, PeerTube non ha alcun supporto integrato per le chat room in diretta, ma è possibile installare un plugin molto popolare che consente di integrare un sistema di chat in tempo reale durante le dirette streaming [52].

Riguardo invece la parte di distribuzione dei video, fino alla versione 6.0.0, PeerTube supportava sia HLS su WebRTC-P2P (noto anche come 'HLS con P2P') sia WebTorrent.

WebTorrent è una re-implementazione del protocollo BitTorrent su WebRTC. Ogni server PeerTube funge da tracker torrent, mentre i browser degli utenti che visualizzano un video partecipano attivamente alla sua distribuzione.

HLS, normalmente implementato con hls.js, non include funzionalità P2P. PeerTube ha esteso questa tecnologia con un loader personalizzato, che scarica contemporaneamente i segmenti video sia dal server web sia da altri spettatori

tramite WebRTC, riducendo così il carico sui server e distribuendo la banda attraverso il P2P.

I video sono resi disponibili per il download tramite HTTP, ma la riproduzione privilegia un sistema peer-to-peer utilizzando HLS e WebRTC P2P [52] [35].

3.2.1 Architettura generale

PeerTube utilizza un'architettura ibrida che combina [19]:

- Un backend basato su Node.js e TypeScript
- Un frontend realizzato con Angular
- Un sistema di storage per i video (locale o distribuito)
- Un database PostgreSQL per la persistenza dei dati
- Un sistema di federazione basato su ActivityPub
- Tecnologie P2P per la distribuzione dei contenuti (Novage P2P Media Loader)

3.2.2 Streaming video con HLS

Per lo streaming video, PeerTube utilizza il protocollo HTTP Live Streaming (HLS), sviluppato da Apple. HLS è un protocollo di streaming adattativo che consente di trasmettere video in diretta o on-demand su Internet. Il protocollo HLS funziona dividendo il flusso video in piccoli segmenti e trasmettendoli tramite HTTP. Questo approccio consente di adattare la qualità del video in base alla larghezza di banda disponibile, garantendo una riproduzione fluida e senza interruzioni. Nello specifico il processo funziona come segue [33]:

- Divide il video in piccoli segmenti (tipicamente di 2-10 secondi)
- Crea un file manifest (.m3u8) che elenca i segmenti disponibili
- Permette la selezione automatica della qualità in base alla connessione
- È compatibile con la maggior parte dei browser e dispositivi moderni

Per eseguire streaming video P2P, PeerTube utilizza un loader/player personalizzato. Questo loader consente di scaricare i segmenti video sia dal server web sia da altri spettatori tramite WebRTC, riducendo così il carico sui server e distribuendo la banda attraverso il P2P.

3.2.3 P2P Media Loader e WebRTC

Il cuore del sistema P2P di PeerTube è costituito da:

- **WebRTC**: Una tecnologia web standard per la comunicazione peer-to-peer diretta tra browser
- **P2P Media Loader**: Una libreria JavaScript che integra il P2P nel player video
- **WebTorrent Tracker**: Un tracker[1] BitTorrent per la scoperta dei peer, utilizzato come sistema di signaling

WebRTC

WebRTC [2] è un progetto gratuito e open-source che fornisce ai browser web e alle applicazioni mobili la possibilità di instaurare comunicazioni in tempo reale tramite API standard, tipicamente in JavaScript. Consente inoltre la comunicazione e lo streaming audio-video direttamente all'interno delle pagine web, permettendo la connessione peer-to-peer senza la necessità di installare plugin o scaricare applicazioni native.

I maggiori componenti che costituiscono WebRTC sono:

- **getUserMedia**: API che consente l'accesso alla fotocamera e al microfono del dispositivo.
- **RTCPeerConnection**: API che consente la comunicazione peer-to-peer per lo scambio di audio, video e dati. Gestisce l'elaborazione del segnale, la gestione dei codec, la comunicazione peer-to-peer, la sicurezza e l'ottimizzazione della larghezza di banda.
- **RTCDataChannel**: API che consente comunicazione bidirezionale di dati arbitrari tra peer. Utilizza le stesse API di WebSocket e offre una latenza molto bassa.

Affinché un'app WebRTC possa avviare una sessione, i suoi client devono scambiarsi le seguenti informazioni:

- Messaggi di controllo della sessione, utilizzati per aprire o chiudere la comunicazione.
- Messaggi di errore.

[1] Un server che aiuta a trovare altri peer nella rete BitTorrent mantenendo un elenco dei peer che stanno scaricando un determinato file [22].

[2] Web Real-Time Communication

- Metadati dei media, come codec, impostazioni dei codec, larghezza di banda e tipi di media.
- Chiavi di sicurezza necessarie per stabilire connessioni sicure.
- Dati di rete, come l'indirizzo IP e la porta dell'host visibili dall'esterno.

Questo processo di signaling, ovvero la scoperta dei peer con cui connettersi e la determinazione delle modalità di stabilimento delle connessioni, richiede un meccanismo che permetta ai client di scambiarsi messaggi avanti e indietro.

WebRTC non include tale meccanismo. Questo compito è lasciato all'applicazione, che può utilizzare qualsiasi mezzo di comunicazione per lo scambio di informazioni di segnalazione.

Le applicazioni per fare ciò devono implementare dei metodi e dei protocolli che sono definiti dal JavaScript Session Establishment Protocol (JSEP) (per evitare ridondanze di implementazione e massimizzare la compatibilità con le tecnologie esistenti).

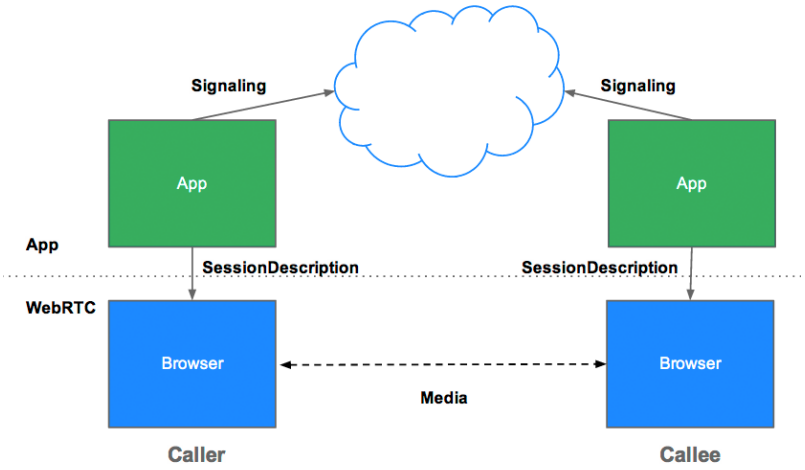

Figure 3.1. Architettura JSEP [8].

JSEP richiede lo scambio tra peer di offerte e risposte, includendo i metadati dei media da voler condividere.

P2P Media Loader

P2P Media Loader è una libreria JavaScript open-source che sfrutta le funzionalità dei browser moderni (HTML5 video e WebRTC) per distribuire contenuti multimediali tramite P2P e riprodurli attraverso l'integrazione con molti lettori video HTML5 popolari. Non richiede plugin o estensioni del browser per funzionare.

Requisiti del browser per P2P Media Loader:

- Media Source Extensions (MSE) o Managed Media Source, necessarie per la riproduzione multimediale tramite i motori Hls.js e Shaka Player.
- Supporto a WebRTC Data Channels per lo scambio di dati tra peer.
- Server STUN, utilizzato da WebRTC per raccogliere *ICE* candidates. Esistono diversi server pubblici disponibili.

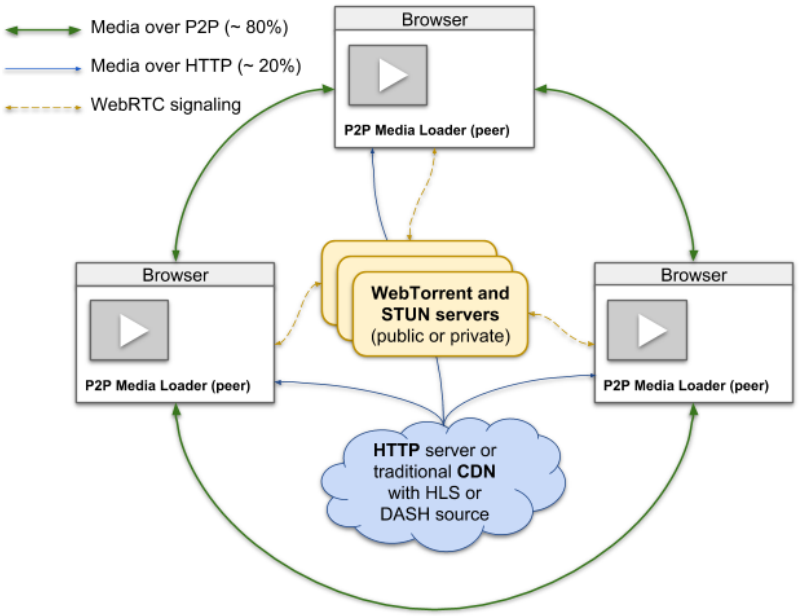

Figure 3.2. Architettura di P2P Media Loader.

Il funzionamento di P2P Media Loader è il seguente: Un browser web esegue un lettore video integrato con la libreria P2P Media Loader. Un'istanza di P2P Media Loader è chiamata peer. Molti peer formano una rete P2P.

P2P Media Loader inizia a scaricare i segmenti iniziali dei media tramite HTTP(S) dal server di origine o CDN. Questo consente di avviare la riproduzione dei video più velocemente. In caso di assenza di peer, il video viene scaricato tradizionalmente con HTTP.

Successivamente, P2P Media Loader invia i dettagli dello stream e i dettagli della sua connessione (candidati ICE) ai tracker di WebTorrent e ottiene da loro l'elenco di altri peer che stanno scaricando lo stesso stream di media.

P2P Media Loader si connette e inizia a scaricare i segmenti dei media dai peer ottenuti, nonché a condividere i segmenti già scaricati con loro.

Di tanto in tanto, peer casuali della rete scaricano nuovi segmenti tramite HTTP(S) e li condividono con gli altri tramite P2P.

3.3 Signaling e NAT Traversal

Come menzionato poco sopra, un aspetto critico del sistema P2P è il processo di signaling e di NAT traversal che permette ai peer di scoprirsi e connettersi tra loro.

NAT è un metodo per mappare uno spazio di indirizzi IP in un altro, modificando le informazioni sugli indirizzi di rete nell'header IP dei pacchetti mentre sono in transito attraverso un dispositivo di routing del traffico.

Questa tecnica è stata inizialmente utilizzata per evitare la necessità di assegnare un nuovo indirizzo a ogni host quando una rete veniva spostata o quando il provider di servizi Internet a monte veniva sostituito ma non poteva instradare lo spazio di indirizzi della rete. Con il tempo, è diventata uno strumento essenziale per la conservazione dello spazio globale degli indirizzi a causa dell'esaurimento degli indirizzi IPv4. Un singolo indirizzo IP instradabile su Internet di un gateway NAT può essere utilizzato per un'intera rete privata.

Poiché la traduzione degli indirizzi di rete modifica le informazioni sugli indirizzi IP nei pacchetti, le implementazioni NAT possono variare nel loro comportamento specifico in diversi casi di indirizzamento e nel loro impatto sul traffico di rete. I produttori di dispositivi che implementano NAT non documentano comunemente i dettagli specifici del comportamento NAT [23].

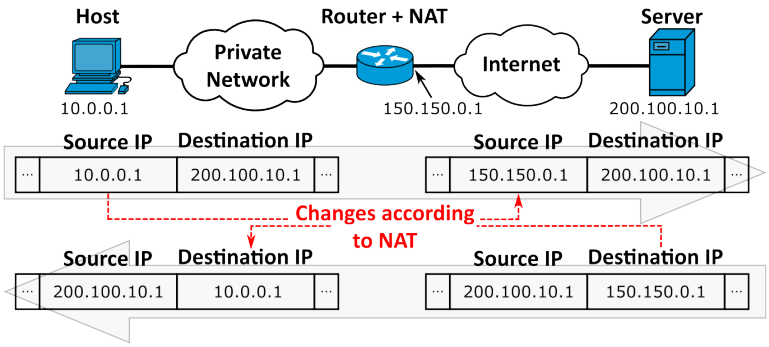

Figure 3.3. Schema NAT [51].

I problemi di attraversamento del NAT (NAT traversal) si verificano quando peer situati dietro NAT diversi tentano di comunicare tra loro direttamente.

Un modo per risolvere questo problema è utilizzare il port forwarding. Un'altra soluzione consiste nell'usare diverse tecniche di attraversamento del NAT.

NAT Traversal è il processo di superamento delle limitazioni imposte dai router NAT (Network Address Translation) che possono impedire la comunicazione diretta tra peer.

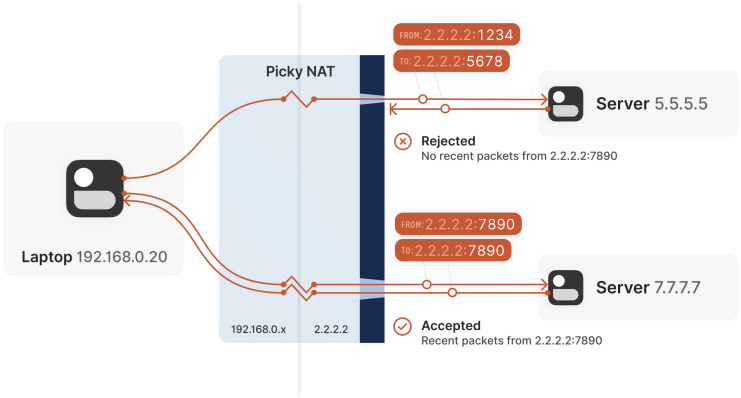

Figure 3.4. Schema NAT Traversal [24].

Esistono diverse tecniche di attraversamento del NAT, tra cui:

- **UPnP** (Universal Plug and Play): Un protocollo di rete di vecchia data, ampiamente implementato che consente ai dispositivi di comunicare tra loro senza la necessità di configurazioni manuali.
- **NAT-PMP** (NAT Port Mapping Protocol): Un protocollo simile a UPnP che consente ai dispositivi di comunicare tra loro attraverso un router NAT creato da Apple.
- **PCP** (Port Control Protocol): Il protocollo successore di NAT-PMP.

Un port mapping riuscito può bypassare efficacemente il NAT per la porta specificata, facilitando le connessioni dirette.

I dispositivi NAT non si comportano tutti in modo uniforme, il che porta a diversi livelli di difficoltà nell'attraversamento:

- **Endpoint-Independent Mapping** (EIM): NAT 'facili' che utilizzano lo stesso IP e porta pubblici per tutte le connessioni in uscita da un determinato IP e porta interni, indipendentemente dalla destinazione.
- **Endpoint-Dependent Mapping** (EDM): NAT 'difficili' o 'simmetrici' che creano mapping diversi di IP e porta pubblici in base all'indirizzo IP di destinazione e, a volte, anche alla porta di destinazione. Questo rende molto più difficile per i peer prevedere il corretto endpoint pubblico. Alcuni dispositivi NAT sono ancora più complessi e generano un mapping NAT completamente diverso per ogni destinazione contattata.

STUN[3] è un insieme standardizzato di metodi, incluso un protocollo di rete, per l'attraversamento dei gateway NAT nelle applicazioni di comunicazione interattiva in tempo reale, come voce, video e messaggistica.

STUN è uno strumento utilizzato da altri protocolli, come Interactive Connectivity Establishment (ICE), Session Initiation Protocol (SIP) e WebRTC. Consente ai dispositivi di rilevare la presenza di un traduttore di indirizzi di rete e di scoprire l'indirizzo IP pubblico mappato e il numero di porta assegnati dal NAT per i flussi UDP dell'applicazione verso host remoti.

Il protocollo richiede l'assistenza di un server di rete di terze parti (un server STUN) situato sul lato opposto (pubblico) del NAT, solitamente su Internet.

STUN è quindi fondamentale per i client dietro un NAT, dato che consente loro di scoprire il proprio indirizzo IP e le porte visibili esternamente, così come vengono viste da un server su Internet.

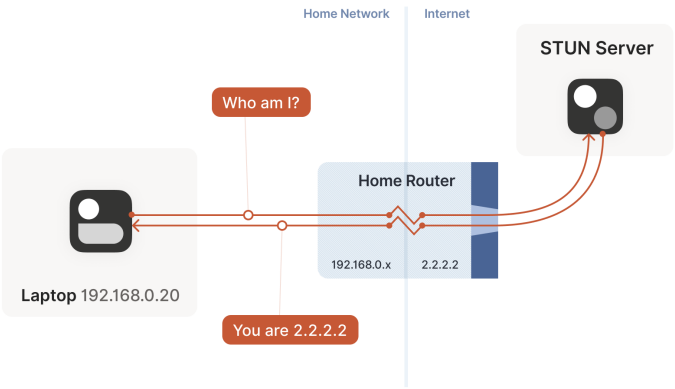

Figure 3.5. Schema STUN [24].

Quando le connessioni peer-to-peer dirette non possono essere stabilite, dei server di inoltro (relay) fungono da intermediari:

- **TURN** (Traversal Using Relays around NAT): un protocollo standard in cui i client si autenticano con un server TURN, che assegna loro un indirizzo IP pubblico e una porta, inoltrando il traffico tra i peer.

WebRTC utilizza il protocollo ICE (Interactive Connectivity Establishment) menzionato prima, per negoziare la connessione tra i peer, utilizzando server STUN e TURN per facilitare il processo [24].

[3]Session Traversal Utilities for NAT; originariamente Simple Traversal of User Datagram Protocol (UDP) through Network Address Translators

3.4 Algoritmo di selezione dei peer

L'efficienza del sistema P2P dipende in modo cruciale dall'algoritmo utilizzato per selezionare i peer da cui scaricare i dati. PeerTube, e di conseguenza Novage P2P Media loader, utilizza come menzionato prima un WebTorrent Tracker come sistema di signaling che utilizza un algoritmo del tutto simile a quello di BitTorrent, dato che il protocollo funziona esattamente come il protocollo BitTorrent ma utilizza WebRTC invece di TCP/uTP come protocollo di trasporto.

La specifica BitTorrent, versione 1.0, definisce vari algoritmi per la selezione dei peer; i peer vengono selezionati attraverso una combinazione di azioni da parte del tracker e dei singoli client. Dalla parte del tracker:

- Quando un client invia una richiesta al tracker (announce), il tracker risponde con un elenco di peer che partecipano allo stesso torrent (o video su una pagina web nel nostro caso).
- Di default il tracker fornisce un elenco fino a 50 peer. Se sono disponibili meno peer, l'elenco sarà più piccolo.
- Il tracker generalmente seleziona i peer in modo casuale per la sua risposta. Tuttavia, può anche utilizzare un algoritmo di selezione dei peer personalizzato.

Una volta che un client riceve un elenco di peer dal tracker e stabilisce le connessioni, gestisce queste connessioni utilizzando un algoritmo di choking e optimistic unchoking. Questo processo determina quali peer il client utilizzerà attivamente per scaricare e caricare i dati [7] [28].

- **Choking**: Un peer può fare choking (rifiutarsi di caricare dati) verso altri peer per diversi motivi, tra cui: Controllo della congestione TCP, per evitare sovraccarichi di rete; Strategia tit-for-tat, per garantire un tasso di download costante, dando priorità ai peer che condividono attivamente. Quando un peer viene 'strangolato', il client non deve inviare richieste per blocchi di dati a quel peer [7].
- **Unchoking**: Per gestire i download e contraccambiare i peer utili, un client solitamente sblocca i quattro peer interessati che hanno il miglior tasso di upload. Questi vengono chiamati 'downloaders'. Se un peer ha un tasso di upload migliore ma non è interessato, potrebbe comunque essere sbloccato, e se diventa interessato, il downloader con il tasso di upload peggiore viene bloccato. I client con un file completo utilizzano il loro tasso di upload per decidere quali peer sbloccare [7].

- **Optimistic unchoking**: Per scoprire potenziali peer migliori, il client sblocca in modo ottimistico un peer per un intervallo di 30 secondi, indipendentemente dal suo tasso di upload. Il peer sbloccato ottimisticamente ruota, il che significa che i peer appena connessi hanno una maggiore probabilità di essere scelti inizialmente.

- **Anti-snubbing**: Se un client smette di ricevere dati da un peer per più di un minuto, suppone di essere stato 'snobbato' e potrebbe temporaneamente smettere di caricare dati verso quel peer. Questo può portare a più di un meccanismo di unchoking ottimistico concorrente per trovare dei peer migliori [7].

3.5 Sistema di monitoraggio integrato con OpenTelemetry

OpenTelemetry è un framework e un toolkit open-source per l'osservabilità. Standardizza la generazione, l'esportazione e la raccolta di dati di telemetria, come tracce, metriche e log, ed è progettato per essere indipendente dai fornitori e dagli strumenti stessi. Affronta la sfida delle pratiche di osservabilità frammentate tra diversi linguaggi, infrastrutture e fornitori, offrendo un unico set di API, specifiche e strumenti per l'istrumentazióne. Questo elimina così il *vendor lock-in* e la necessità di adattarsi a più sistemi proprietari.[46]

PeerTube integra OpenTelemetry, per il monitoraggio delle prestazioni complessive del server, raccogliendo metriche come:

- Percentuale di dati condivisi via P2P e HTTP
- Numero di utenti connessi
- Utilizzo della CPU, della memoria etc
- Utilizzo della banda del server
- Errori e disconnessioni

Queste metriche possono essere poi visualizzate attraverso dashboard Grafana o altri sistemi di monitoraggio compatibili con OpenTelemetry.

Capitolo 4

Verifica empirica delle prestazioni P2P di PeerTube

A dicembre 2023, il team di PeerTube ha pubblicato un articolo [34] in cui va ad analizzarne le prestazioni di PeerTube facendo degli *stress test* per verificare se la tecnologia P2P integrata nel sistema sia effettivamente in grado di ridurre il carico sui server con circa 1000 utenti connessi contemporaneamente, in quanto, secondo i dati raccolti da Twitch nel 2022, coprivano il 99% dei video in diretta della piattaforma.

Per realizzare test veritieri, il team ha simulato 1000 spettatori simultanei utilizzando 1.000 browser Chrome, ciascuno con un indirizzo IP pubblico IPv6 dedicato. Questo è stato realizzato tramite Selenium grid, un software di automazione e testing per i browser, affiancato da Docker su cloud Hetzner e successivamente con un potente server fornito da Octopuce.

La scelta di 1.000 spettatori è significativa poiché copre la stragrande maggioranza delle dirette streaming su piattaforme importanti come Twitch, suggerendo che PeerTube può essere adeguato per un'ampia gamma di casi d'uso. In condizioni ottimali, l'aspetto P2P di PeerTube dovrebbe ridurre la larghezza di banda necessaria per trasmettere un video in diretta di un fattore da 3 a 4, stando quanto detto degli sviluppatori di PeerTube.

Sono stati condotti 4 scenari di test principali:

- Live streaming con impostazione *Normal Latency*
- Live streaming con impostazione *High Latency*
- Live streaming con impostazione *High Latency* e 50% dei peer con P2P disabilitato

- Un normale video *on-demand*

su una macchina virtuale con:

- 4 vCore i7-8700 CPU @ 3.20GHz
- 4 GB di RAM
- 1 Gbps di banda

I dati dei test sono stati raccolti tramite OpenTelemetry e Grafana, con metriche come:

- Percentuale di dati trasferiti via P2P vs. dal server
- Utilizzo di CPU
- Comportamento in condizioni di rete variabili
- Numero di spettatori

con i quali, infine, sono stati in grado di dimostrare che PeerTube è in grado di gestire 1.000 spettatori simultanei con un carico minimo sui server, grazie alla tecnologia P2P integrata, in quanto la quantità di dati trasferiti via P2P è progressivamente aumentata con il tempo, fino a raggiungere un rapporto del 75% dei dati totali trasferiti per i video in diretta e del 98% per i video *on-demand*.

	HTTP peak	HTTP after 5 minutes	P2P after 5 minutes	HTTP/P2P ratio after 5 minutes
Live	150 Mbit/s	90 Mbit/s	350 Mbit/s	25% (P2P saves 75% of bandwidth)
VOD	200 Mbit/s	25 Mbit/s	1150 Mbit/s	2% (P2P saves 98% of bandwidth)

Figure 4.1. PeerTube conclusioni sullo stress test.

Tuttavia, l'articolo non fornisce dettagli sufficienti sulla metodologia utilizzata né rilascia gli strumenti specifici per riprodurre i test in modo indipendente. [34]

Perciò abbiamo deciso di creare un sistema di test automatizzato che possa riprodurre i test descritti nell'articolo originale.

4.1 Metodologia per la verifica empirica

Per verificare le affermazioni degli sviluppatori di PeerTube, abbiamo deciso di:

- Creare un'automazione per riprodurre indipendentemente i test descritti nell'articolo
- Raccogliere metriche più dettagliate rispetto a quelle presentate nell'articolo originale
- Analizzare i dati raccolti per valutare le prestazioni del sistema P2P di PeerTube

4.2 Stack tecnologico per i test

Per i nostri test abbiamo utilizzato una combinazione di tecnologie open source per creare un sistema di test automatizzato che sfrutta le seguenti tecnologie:

- **Docker**: Per creare ambienti isolati e facilmente riproducibili
- **Telegraf**: Per la raccolta di metriche di sistema e di rete
- **MongoDB**: Per l'archiviazione strutturata dei dati raccolti
- **Python**: Come linguaggio principale per l'automazione
- **Selenium**: Per simulare browser reali che guardano lo stream
- **WebRTC Internals Exporter**: Una estensione del browser creata per raccogliere metriche dettagliate sulle connessioni WebRTC
- **WebPack**: Un sistema di compilazione e gestione delle dipendenze per il frontend web
- **Hetzner Cloud e script CLI**: Per distribuire i test su macchine virtuali in diverse regioni geografiche

4.2.1 Docker

Docker è una piattaforma open source che semplifica la creazione, la distribuzione e l'esecuzione di applicazioni in contenitori. I contenitori Docker sono degli ambienti isolati che simulano un sistema operativo completo, come le macchine virtuali, consentendo di eseguire applicazioni in modo consistente su qualsiasi ambiente.

La differenza tra docker e una macchina virtuale è che i container Docker condividono il kernel del sistema operativo host, riducendo l'overhead e migliorando le prestazioni, senza dover ricorrere ad un [1]*Hypervisor* per la virtualizzazione.

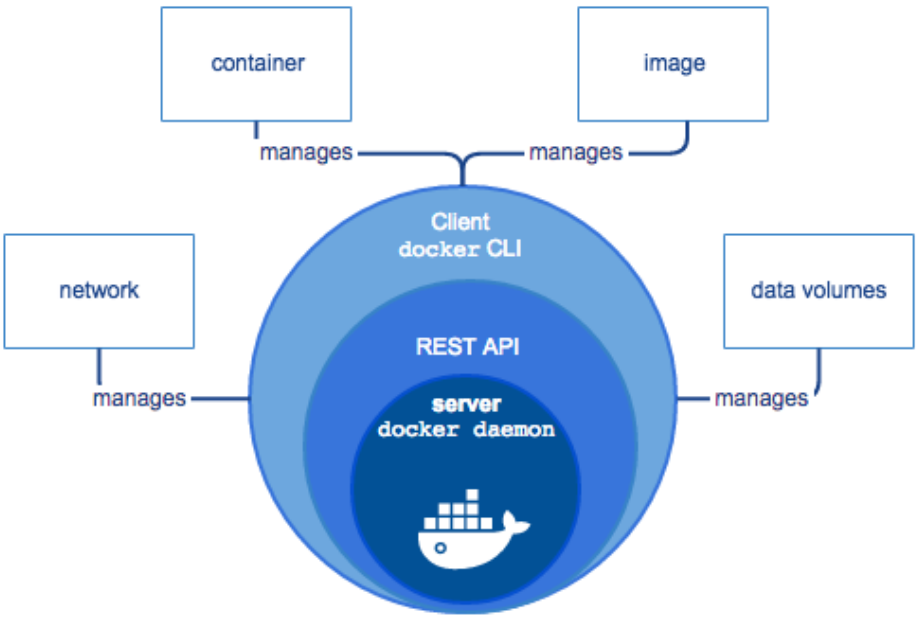

Figure 4.2. Docker Engine [14].

In sé Docker è composto da 4 componenti principali [30] [11]:

- **Docker Engine**: Il core del sistema, responsabile della creazione e gestione dei container nonché il processo demone eseguito sulla macchina host. Fornisce l'accesso a tutte le funzionalità e i servizi messi a disposizione da Docker. Fornisce un insieme di comandi per la gestione dei container, delle immagini e dei volumi.
- **Docker Client**: Interfaccia da riga di comando e *API* [2] per interagire con Docker Engine.
- **Docker Image**: Un *template* di sola lettura che contiene e definisce i parametri di una applicazione da eseguire in un container a runtime. Le immagini vengono create e organizzate per livelli *stateless* e immutabili.

[1]Hypervisor: Software che consente di eseguire più sistemi operativi su singolo hardware.
[2]API: sinonimo di Application Programming Interface.

- **Docker Container**: Un'istanza in esecuzione di un'immagine Docker. Un container è un ambiente isolato che esegue un'applicazione specifica e include tutto il necessario per eseguire l'applicazione, come il codice, le librerie, le variabili d'ambiente e le dipendenze. Il *filesystem* del container è l'ultimo livello che viene aggiunto al quale vi è possibile accedere sia in lettura che in scrittura. I contenitori, inoltre, possono essere associati a dei volumi per la persistenza dei dati fornendo un metodo semplice e immediato per condividere dati tra i container e l'host.
  La comunicazione tra container avviene tramite la creazione di *network* o reti separate che vengono connesse ai singoli container, abilitando così una sorta di LAN interna tra un insieme di container. Per fare comunicare i container con il mondo esterno bisogna invece utilizzare i 'port mapping' tra una porta della macchina host e una porta del container.

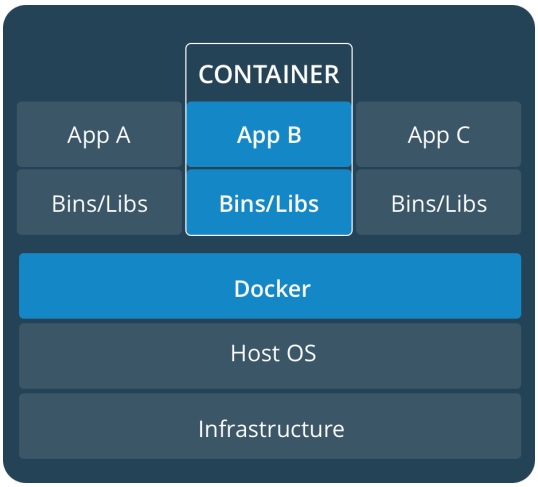

Figure 4.3. Docker Overview [15].

Approfondendo il discorso del networking, Docker mette a disposizione vari tipi di reti per la comunicazione tra container. I 3 principali sono [13]:

- **Bridge**: È la rete di default che viene creata quando si installa Docker. I container connessi a questa rete possono comunicare tra loro e con l'host, ma non possono comunicare con i container in altre reti.
- **Host**: I container connessi a questa rete condividono la rete dell'host, quindi non hanno bisogno di fare *port mapping* per comunicare con il mondo esterno.
- **None**: I container connessi a questa rete non hanno accesso ad alcuna rete, quindi non possono comunicare con altri container o con l'esterno.

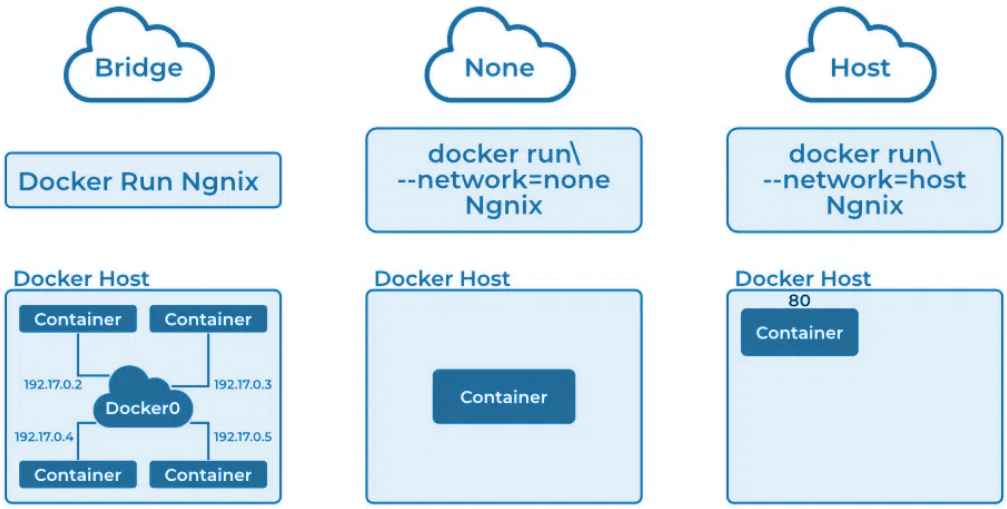

Figure 4.4. Docker Networking [13].

4.2.2 Telegraf

Telegraf è un agente di raccolta di metriche open source sviluppato da InfluxData. È progettato per raccogliere, elaborare e inviare metriche da una varietà di sorgenti, tra cui sistemi operativi, database, applicazioni e dispositivi di rete.

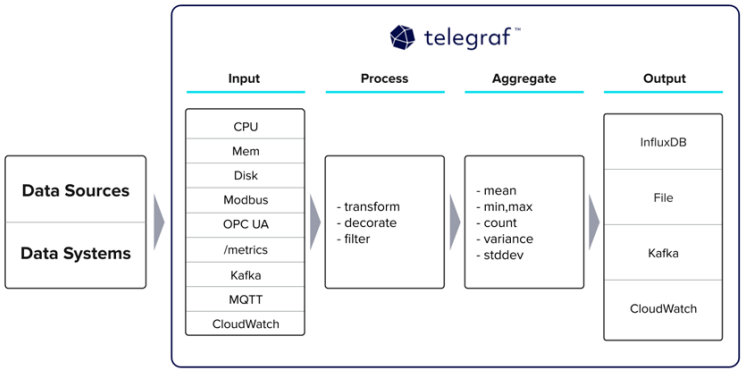

Figure 4.5. Architettura Telegraf [25].

Scritto in *Go*, Telegraf è dotato di oltre 300 plugin di input, trasformazione e output di dati, che consentono di raccogliere metriche da una vasta gamma di sorgenti. Per sua natura, funziona come un pipeline di dati che può essere

instradata attraverso diversi plugin per elaborare e aggregare le informazioni prima di raggiungere l'output finale [38].

Alcuni componenti principali di Telegraf includono [21]:

- **Agent**: Il core del sistema, responsabile della raccolta e dell'elaborazione delle metriche dai plugin di input definiti e le invia ai plugin di output specificati, in base alla configurazione fornita.
- **Input Plugins**: Raccolgono metriche da sorgenti come sistemi operativi, database, applicazioni e dispositivi di rete.
- **Processor Plugins**: Elaborano e trasformano le metriche raccolte prima di inviarle all'output.
- **Output Plugins**: Inoltrano le metriche elaborate a un sistema di monitoraggio o di archiviazione.
- **Aggregator Plugins**: Aggregano le metriche raccolte per ridurre il volume dei dati.

La configurazione di Telegraf è definita da un file di configurazione *TOML* che definisce i plugin di input, processore e output da utilizzare, insieme a eventuali parametri aggiuntivi necessari per la raccolta e l'elaborazione delle metriche.

Viene spesso utilizzato affianco a un database di tipo *time-series* come InfluxDB per l'archiviazione e la visualizzazione delle metriche raccolte, ma può essere integrato con una vasta gamma di sistemi di monitoraggio e analisi dei dati. Una metrica *time-series* è una serie di dati indicizzati in sequenza rispetto al tempo. Un esempio è una sequenza di osservazioni o rilevazioni registrate allo scorrere del tempo. Vi sono due macrocategorie di *time-series* [27]:

- **univariate time series**: le osservazioni sono monodimensionali, ovvero: viene registrato un solo valore numerico allo scorrere del tempo.
- **multivariate time series**: le osservazioni sono multidimensionali, ovvero, si registrano più valori numerici per un singolo istante di tempo.

Tipicamente, vengono rappresentate con una struttura dati che registra un timestamp, che può essere di qualche tipo specifico per date, oppure un intero, contenente uno *Unix timestamp*; oltre a questo contiene dati addizionali, che nella versione più semplice possono essere un unico valore numerico, ovvero l'osservazione registrata allo scorrere del tempo.

4.2.3 MongoDB

MongoDB è un database *NoSQL* open source, orientato ai documenti, sviluppato da MongoDB Inc. È progettato per essere flessibile, scalabile e ad alte prestazioni, consentendo di memorizzare e interrogare dati in modo efficiente. Combina la capacità di scalare orizzontalmente con funzionalità come indici secondari, query per intervallo, ordinamento, aggregazioni e indici geospaziali.

Un database orientato ai documenti sostituisce il concetto di 'riga' con un modello più flessibile, il 'documento'. Grazie alla possibilità di includere documenti incorporati e array, l'approccio orientato ai documenti permette di rappresentare relazioni gerarchiche complesse all'interno di un singolo record. Non ci sono schemi predefiniti: le chiavi e i valori di un documento non hanno tipi o dimensioni fisse. L'assenza di uno schema rigido rende più semplice aggiungere o rimuovere campi secondo necessità. In generale, questo accelera lo sviluppo di nuovi database, permettendo agli sviluppatori di iterare rapidamente e sperimentare con facilità, rendendo possibile testare diversi modelli di dati e scegliere quello più adatto alle proprie esigenze.

MongoDB è progettato per essere un database generico, quindi, oltre alle operazioni di inserimento, lettura, aggiornamento ed eliminazione dei dati (*CRUD*), offre alcune funzionalità uniche in più, tra cui [9]:

- **Indicizzazioni**: Indici di diversi tipi, tra cui indici singoli, composti, geospaziali e testuali. Gli indici possono essere creati per qualsiasi campo all'interno di un documento, consentendo di ottimizzare le query per le prestazioni.
- **Aggregazioni**: Un'ampia gamma di operazioni di aggregazione, come *group*, *match*, *project* e *sort*, che consentono di eseguire query complesse e analisi dei dati direttamente nel database.
- **Collezioni speciali**: Collezioni speciali come le collezioni*time-series* e le collezioni *time-to-live*, che semplificano la gestione di dati temporali e di dati che devono essere eliminati dopo un certo periodo di tempo come ad esempio le sessioni utente.

MongoDB memorizza i record di dati come documenti (nello specifico documenti *BSON*, ovvero una rappresentazione binaria di documenti in formato *JSON*), che sono raggruppati in collezioni. Un database contiene una o più collezioni di documenti [16].

Figure 4.6. Documento JSON [16].

JSON, o JavaScript Object Notation, è un formato di scambio dati testuale, basato su una sintassi di oggetti JavaScript. È comunemente utilizzato per trasmettere dati strutturati su una rete, come ad esempio tra un server e un client web. Un *documento JSON*, invece, è una collezione di campi e valori organizzata in un formato strutturato.

MongoDB Query Language, o *MQL*, è il linguaggio di query utilizzato per interagire con un database MongoDB. MQL è simile a SQL, ma è progettato per lavorare con documenti JSON e collezioni di documenti, piuttosto che con tabelle e righe. Consente agli utenti di recuperare documenti che corrispondono a criteri specifici, eseguire aggregazioni, apportare aggiornamenti ai documenti ed effettuare cancellazioni di documenti. La sintassi di MQL è progettata per essere semplice e intuitiva. Permette agli utenti di specificare condizioni utilizzando operatori come $eq (uguale), $ne (diverso), $gt (maggiore di), $lt (minore di) e molti altri. È possibile anche utilizzare operatori 'bitwise' come $and, $or e $not per creare query complesse [39].

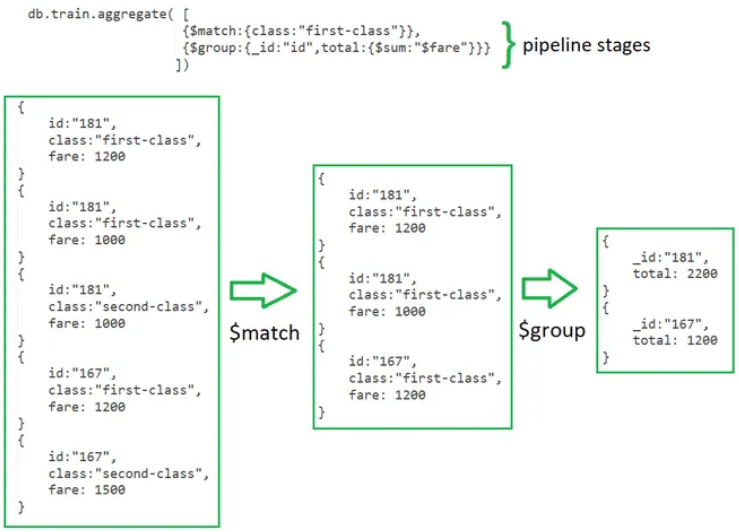

Figure 4.7. MongoDB Aggregation Pipeline [4].

Infine, il MongoDB Query Language Aggregation Framework è un set di processi che trasforma i dati in strutture che possono essere utilizzate per l'analisi. L'aggregazione si basa su un concetto di *pipeline*, in cui i documenti vengono passati attraverso una serie di fasi di trasformazione. Ogni fase della pipeline esegue un'operazione specifica sui documenti, come filtering, proiezione, ordinamento, raggruppamento e altro ancora. Le fasi della pipeline, possono essere quindi concatenate per creare query più complesse [9].

4.2.4 Python

Python è un linguaggio di programmazione ad alto livello, interpretato, orientato agli oggetti e con semantica dinamica. È progettato per essere facile da leggere e scrivere, con una sintassi pulita e chiara che favorisce la leggibilità del codice. È noto per la sua facilità d'uso e la sua versatilità, che lo rendono adatto a una vasta gamma di applicazioni, dallo sviluppo web alla data science, all'automazione dei processi.

Essendo un linguaggio interpretato, Python non richiede una fase di compilazione, ma viene eseguito direttamente da un *interpreter*. Questo lo rende particolarmente adatto per lo sviluppo rapido di applicazioni e per l'iterazione veloce durante lo sviluppo. Python è anche noto per la sua vasta libreria standard, che fornisce un'ampia gamma di moduli e pacchetti pronti all'uso per svolgere una varietà di compiti comuni e per questo viene utilizzato spesso per fare 'data science'.[42]

L'indentazione è parte fondamentale in quanto, a differenza di altri linguaggi di programmazione dove l'indentazione è solo una convenzione, in Python è obbligatoria e definisce la struttura del codice. Questo rende il codice più leggibile e coerente, ma richiede anche attenzione per mantenere una corretta struttura del codice.

```
import numpy as np

def incmatrix(genl1,genl2):
    m = len(genl1)
    n = len(genl2)
    M = None #to become the incidence matrix
    VT = np.zeros((n*m,1), int)  #dummy variable

    #compute the bitwise xor matrix
    M1 = bitxormatrix(genl1)
    M2 = np.triu(bitxormatrix(genl2),1)

    for i in range(m-1):
        for j in range(i+1, m):
            [r,c] = np.where(M2 == M1[i,j])
            for k in range(len(r)):
                VT[(i)*n + r[k]] = 1;
                VT[(i)*n + c[k]] = 1;
                VT[(j)*n + r[k]] = 1;
                VT[(j)*n + c[k]] = 1;

                if M is None:
                    M = np.copy(VT)
                else:
                    M = np.concatenate((M, VT), 1)

                VT = np.zeros((n*m,1), int)

    return M
```

Listing 4.1: Esempio di codice Python

4.2.5 Selenium

Selenium è un framework open-source ampiamente utilizzato per l'automazione dei browser web. Fornisce un insieme di strumenti e librerie che consentono a tester e sviluppatori di interagire con gli elementi di una pagina, simulare azioni dell'utente ed eseguire test automatizzati su applicazioni web.

Uno degli aspetti chiave di Selenium è la sua capacità di funzionare con diversi browser, garantendo la compatibilità cross-browser e permettendo l'esecuzione di test in ambienti differenti.

Supporta diversi browser popolari, tra cui Google Chrome, Mozilla Firefox, Microsoft Edge, Safari e Opera. Ogni browser è gestito tramite implementazioni specifiche di *WebDriver*, che fungono da ponte tra Selenium e il browser.

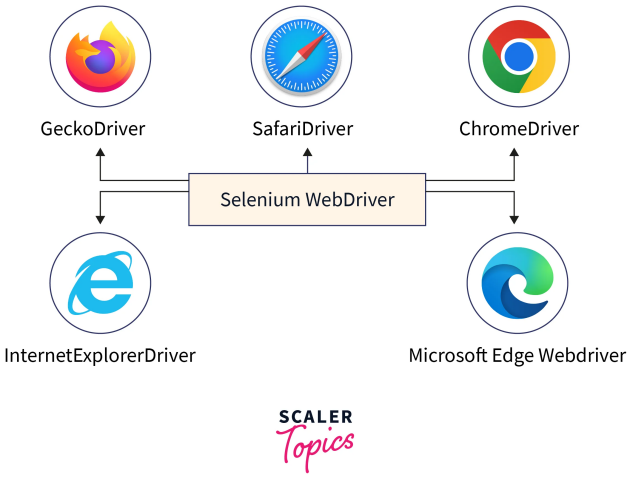

Figure 4.8. Browser supportati da Selenium [26].

Utilizzando i WebDriver, sviluppatori e tester possono scrivere script di automazione nel loro linguaggio di programmazione preferito ed eseguirli su diversi browser. Selenium WebDriver offre un'API unificata che astrae le differenze tra i vari browser, consentendo un'automazione fluida e risultati di test coerenti [26].

Una volta costruiti i casi di test, i dati devono essere comunicati al driver del browser in qualche modo. In Selenium, questo avviene utilizzando il 'JSON Wire Protocol' con HTTP come gestore della comunicazione.

Poiché la comunicazione avviene in un'infrastruttura client-server, nel mondo di Selenium WebDriver questo sistema è noto come *request-response* o *command-response*.

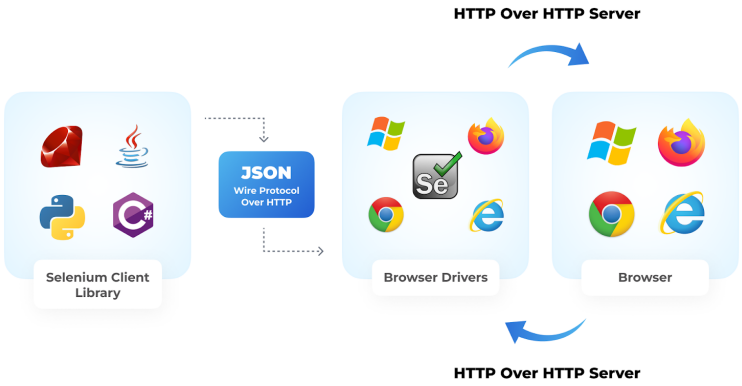

Figure 4.9. JSON Wire Protocol [37].

Il funzionamento dei WebDriver può essere riassunto in alcuni passaggi chiave [37]:

- Un tester scrive uno script di test automatizzato destinato a un driver di browser specifico.
- Quando viene premuto il pulsante 'Esegui', lo script viene convertito in formato API con dati in JSON.
- I dati vengono trasferiti al driver del browser utilizzando il JSON Wire Protocol su una rete HTTP come API RESTful.
- Il driver del browser riceve i dati e, se la validazione ha successo, comunica le azioni al browser tramite HTTP.
- Se la validazione fallisce, gli errori vengono restituiti al client.
- Una volta avviato il browser, il driver esegue le azioni una per una, simulando il comportamento di un tester manuale attraverso l'automazione.
- I comandi vengono inviati tramite HTTP, e le risposte vengono ricevute dallo stesso protocollo nel driver.
- Dopo l'esecuzione di tutte le azioni, il browser si chiude e il driver comunica i risultati al client.

Affinché il driver possa comunicare correttamente con il browser, è fondamentale che il browser stesso sia installato sul sistema e che il sistema e il suo ambiente

sia configurato correttamente, con tutti i prerequisiti necessari per l'esecuzione dei test.

Un altro componente messo a disposizione da Selenium è *Selenium Grid*, che permette di eseguire test su più macchine e browser remoti contemporaneamente, riducendo i tempi di esecuzione e migliorando l'efficienza dei test, instradando i comandi inviati dal client alle istanze remote dei browser. È composto da 2 componenti principali [40]:

- **Hub**: Il server principale che funge da punto di ingresso per i test. Riceve i comandi dai client e li instrada alle istanze dei browser remote.
- **Node**: Le macchine remote che eseguono i test. Si connettono all'*Hub* e ricevono i comandi per eseguire i test sui browser installati.

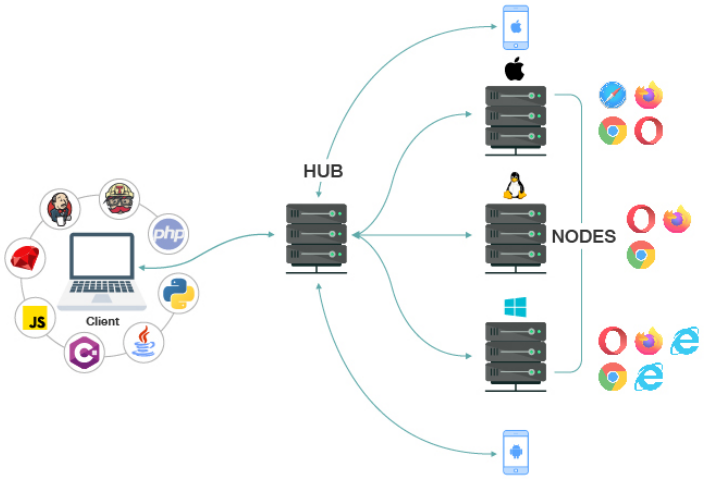

Figure 4.10. Architettura Selenium Grid [36].

Per fare tutto questo, Selenium mette a disposizione varie immagini Docker preconfigurate con tutto il necessario per eseguire i test, come il browser, il driver, l'hub, e le dipendenze necessarie, con anche diverse versioni legacy per testare la compatibilità con versioni obsolete.

In particolare, a noi interessa l'immagine *selenium/standalone-chrome*, che contiene tutto il necessario per eseguire test su Chromium, incluso il browser stesso, il driver e il server Selenium, ed è stata usata come base per **creare una immagine Docker monolitica** per i nostri test, con tutte le ulteriori dipendenze necessarie come:

- **Python**: Per l'automazione.

- **Telegraf**: Per la raccolta ed invidio di tutte le metriche.
- **WebRTC Internals Exporter**: Per la raccolta delle metriche WebRTC.
- **Webpack**: Per la sostituzione delle variabili d'ambiente nel codice JavaScript.

4.2.6 WebRTC Internals Exporter

WebRTC Internals Exporter è un'estensione per Chromium che consente di esportare le metriche WebRTC da un browser in un formato leggibile e analizzabile. Creato da un ricercatore Italiano, è stato progettato per semplificare il processo di raccolta e analisi delle metriche WebRTC, che altrimenti sarebbero difficili da ottenere in quanto Chromium mette a disposizione uno strumento chiamato WebRTC-internals, che consente di visualizzare le statistiche WebRTC in tempo reale, ma non offre un modo semplice per esportarle, dato che di base offre solo la creazione di file 'dumps' non automatizzabile.

L'uso di webrtc-internals con il processo di creazione/caricamento dei dump presenta alcune grosse limitazioni:

- È necessario aprire la pagina *chrome://webrtc-internals* prima di avviare una sessione. Se la sessione è già in corso e si verificano problemi, non è possibile raccogliere metriche retroattivamente.
- La visualizzazione delle metriche è limitata nel tempo e non è possibile personalizzarle, filtrarle o creare query aggiuntive sui dati raccolti.
- L'uso di file dump può rendere difficile l'archiviazione e l'organizzazione dei dati dei test, oltre alla condivisione dei risultati. Inoltre, il caricamento di file dump di grandi dimensioni in altri strumenti di analisi può richiedere molto tempo e memoria.

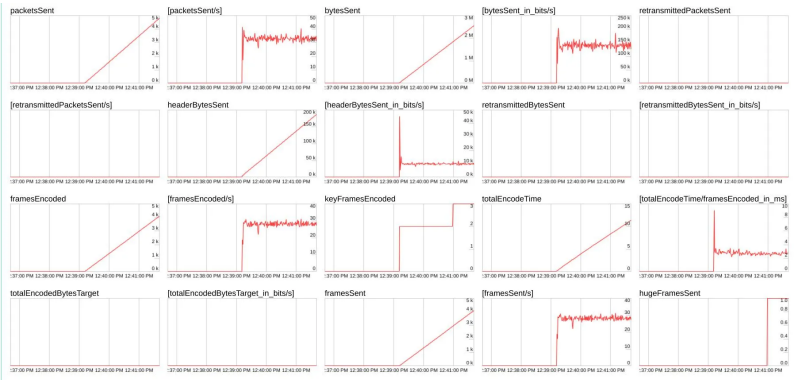

Figure 4.11. Grafici WebRTC Internals.

WebRTC Internals Exporter risolve questi problemi abilitando l'esportazione delle metriche verso un servizio chiamato *Prometheus PushGateway*. Prometheus PushGateway è un servizio progettato per permettere a lavori effimeri e batch di esporre le proprie metriche a Prometheus. Poiché questi processi potrebbero non esistere abbastanza a lungo da essere interrogati direttamente, possono inviare le loro metriche al PushGateway, garantendo così la raccolta dei dati.

Tecnicamente l'estensione funziona in questo modo: viene pre-caricato un codice JavaScript che sovrascrive la classe globale *RTCPeerConnection*.

La nuova classe assegna un ID casuale a tutti gli oggetti peer connection creati e li memorizza all'interno dell'oggetto *webrtcInternalExporter*. L'utente può abilitare l'estensione per la scheda attualmente aperta facendo clic sull'icona dell'estensione e attivando la relativa checkbox.

Dopo l'attivazione, lo script di override inizierà a chiamare periodicamente il metodo *getStats* sugli oggetti creati. Tutte le metriche selezionate verranno inviate allo script in background, che le inoltrerà al servizio PushGateway configurato.

Il metodo *getStats* è un metodo standardizzato definito nell'API WebRTC che consente di ottenere le statistiche relative a una connessione peer-to-peer.[31]

Questo progetto fa uso di una **versione altamente modificata**[3] di WebRTC Internals Exporter, per adattarsi alle nostre esigenze di test e automazione. In particolare, abbiamo modificato le funzionalità:

- Abilitare l'esportazione delle metriche WebRTC verso un *endpoint HTTP RESTful* al posto di Prometheus PushGateway, con una formattazione dei dati personalizzata, ma sempre basata su JSON.
- Aggiunta di metriche esportate aggiuntive rispetto a quelle di base fornite da *getStats*.
- Abilitazione automatica dell'estensione all'avvio del browser, senza intervento manuale.
- Supporto alla configurazione tramite variabili d'ambiente per semplificare l'integrazione con il nostro sistema di test.

[3]Disponibile qui: https://gitlab.di.unimi.it/mirko.milovanovic/peertube-collector/-/tree/main/webrtc-internals-exporter

```
{
    "_id": "67cf2a286904313f4e5ad32c",
    "player": {
        "Codecs": {
            "Audio": "mp4a.40.2",
            "Video": "avc1.64002a"
        },
        "Connection Speed": {
            "Granularity": "s",
            "Speed": 16777216
        },
        "Download Breakdown": {
            "Peers": 0,
            "Server": 11534336
        },
        "Live Latency": {
            "Edge": 0,
            "Latency": 21
        },
        "Network Activity": {
            "Down": 7340032,
            "Up": 0
        },
        "P2P": "enabled",
        "Player mode": "p2p-media-loader",
        "Resolution": "1080p",
        "Total Transfered": {
            "Down": 11534336,
            "Up": 0
        },
        "Video UUID": "70663bee-df65-4080-84bb-eefc5a11ab75",
        ...
    },
    "peers": [
        {
            "connectionState": "new",
            "iceCandidateErrors": [
                {
                    "errorCode": 701,
                    "errorText": "STUN host lookup received error.",
                    "timestamp": 1741629977060,
                    "url": "stun:stunserver2024.stunprotocol.org:3478"
                },
                ...
            ],
            "iceCandidates": [
                {
                    "candidate": "candidate: 3130924284 1 udp 2113937151
                      172.17.0.2 43668...",
                    "component": "rtp",
                    "foundation": "3130924284",
                    "port": 43668,
                    "priority": 2113937151,
                    "protocol": "udp",
                    "relatedAddress": null,
                    "relatedPort": null,
                    "sdpMLineIndex": 0,
                    "sdpMid": "0",
                    "tcpType": null,
                    "timestamp": 1741629977042,
                    "type": "host",
```

```
                        "usernameFragment": "6V/z"
                    },
                    ...
                ],
                "iceConnectionState": "new",
                "iceGatheringState": "complete",
                "id": "0dfd00af-518b-46c3-a2de-bc3330da9312",
                "signalingState": "have-local-offer",
                "url": "https://tube.kobim.cloud/w/eT1NZibmwMy6bx6N2YGLwr",
                "values": [
                    {
                        "bytesReceived": 0,
                        "bytesSent": 0,
                        "id": "D1",
                        "label": "01e722a46bc1440994d511d27e86171d976d105a",
                        "messagesReceived": 0,
                        "messagesSent": 0,
                        "protocol": "",
                        "state": "connecting",
                        "timestamp": 1741629979445.746,
                        "type": "data-channel"
                    },
                    ...
                ]
            }
        ],
        "tags": {
            "host": "selenium-nbg1-node-5-instance-1",
            "session": "64001538ac0eb86fafc5ed531b96ff7c",
            "url": "https://tube.kobim.cloud/w/eT1NZibmwMy6bx6N2YGLwr"
        },
        "timestamp": "2025-03-10T18:06:19.496Z"
    }
```


Listing 4.2: Esempio di dati JSON con le metriche esportate da WebRTC Internals Exporter

4.2.7 Webpack

Webpack è un bundler di moduli per applicazioni JavaScript. Prende moduli con varie dipendenze e genera dei file statici che rappresentano quei moduli. È ampiamente utilizzato per applicazioni web moderne, in particolare quelle basate su framework come React, Angular e Vue.js.

Quando Webpack elabora un'applicazione, costruisce internamente un grafo delle dipendenze a partire da uno o più entry points. Successivamente, combina tutti i moduli necessari al progetto in uno o più bundle, che sono asset statici utilizzati per servire i contenuti dell'applicazione.

Di default, Webpack 'comprende' solo file JavaScript e JSON.

Per gestire altri tipi di file, utilizza i *Loaders*, che permettono di elaborare e convertire vari formati (come CSS, immagini o TypeScript) in altro moduli validi. Questi moduli possono essere quindi usati dall'applicazione e aggiunti al grafo delle dipendenze.

Altro componente essenziale sono i *Plugin*. Mentre i *Loaders* trasformano i file in moduli validi, i *Plugin* servono per eseguire operazioni più avanzate, come ottimizzazione, gestione degli asset, variabili d'ambiente e molto altro.[10]

4.2.8 Hetzner Cloud e script CLI

Hetzner Cloud è un servizio di cloud computing offerto da Hetzner Online GmbH, un provider di servizi di hosting e data center con sede in Germania. Hetzner Cloud offre una vasta gamma di servizi di cloud computing, tra cui server virtuali, storage, reti e servizi di sicurezza.

Sulla base di quello che è stato fatto da PeerTube, abbiamo deciso di utilizzare Hetzner Cloud per distribuire i nostri test su macchine virtuali in diverse regioni geografiche, in quanto, gli script per l'automazione dei test sono l'unica parte fornitaci da PeerTube nell'articolo originale. Anche qui abbiamo utilizzato delle versioni altamente modificate per adattarle alle nostre esigenze.

Gli script fanno utilizzo della *CLI*[4], di Hetzner Cloud, che consente di gestire le risorse cloud direttamente dalla riga di comando, e sono formati da 2 parti principali: uno script per la creazione delle macchine virtuali e uno script per l'avvio dei test [20] [18].

Rispetto a quelli originali, li abbiamo modificati per far utilizzo delle variabili d'ambiente per la configurazione del nostro sistema di test e per sostituire Selenium Grid, con l'immagine Docker monolitica standalone che abbiamo creato.

[4]CLI: sinonimo di Command-line interface

4.3 Architettura del sistema di test

Finora abbiamo descritto le tecnologie utilizzate alla base per creare il nostro sistema di test, senza però spiegare effettivamente come queste vengono integrate e utilizzate insieme per creare un sistema di test automatizzato.

Introduciamo quindi l'ultimo pezzo necessario per far funzionare il tutto, ovvero il ***collector***, un'applicazione Python che si occupa di fare scraping[5] delle metriche di PeerTube, di raccogliere le metriche WebRTC tramite l'estensione Chromium e di inviare il tutto a Telegraf per l'elaborazione e l'invio al database.

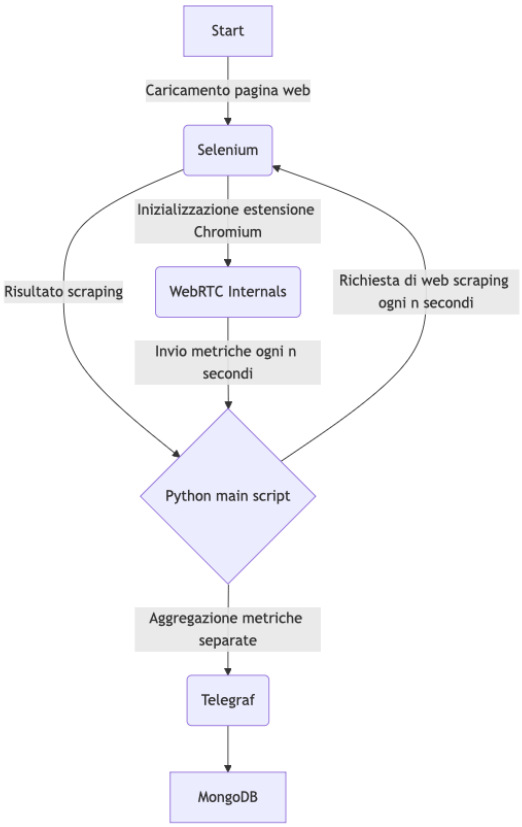

Figure 4.12. Architettura del collector.

[5]Il data scraping è una tecnica in cui un programma informatico estrae dati da un output leggibile dall'uomo generato da un altro programma.

Nella figura 4.12 viene mostrato il funzionamento e il flusso dei dati all'interno del *collector* e come esso si interfacci con le altre tecnologie utilizzate.

Nello specifico il *collector* è composto da 3 parti principali:

- Un Python script principale che coordina il tutto
- Selenium
- WebRTC Internals Exporter

Lo script Python è il cuore del sistema, che si occupa di coordinare le altre componenti, di avviare e fermare i test, di raccogliere i dati, aggregarli e di inviarli a Telegraf per l'invio al database.

L'intero progetto è disponibile qui:

- https://gitlab.di.unimi.it/mirko.milovanovic/peertube-collector

4.3.1 Difficoltà incontrate e soluzioni

Durante la creazione del sistema, abbiamo incontrato alcune difficoltà, per lo più sistemistiche, che hanno richiesto soluzioni creative per superarle. Alcune delle principali sfide sono state:

- **Accesso agli indirizzi IP forniti da WebRTC**: Dalle ultime versioni, come misura di sicurezza e privacy, non permette più l'accesso e la visualizzazione degli indirizzi IP locali delle macchine coinvolte nella connessione grazie a tecniche di offuscazione, rendendo difficile la raccolta di queste informazioni. Per ovviare a questo problema ci siamo avvalsi di 2 soluzioni: la prima consiste nell'utilizzare una versione vecchia di Chromium, nello specifico una versione anteriore o uguale alla 129.0, in quanto in queste versioni l'offuscazione è presente ma è possibile disabilitarla tramite un flag da interfaccia grafica; la seconda soluzione invece consiste proprio nel disabilitare questa funzionalità ma da riga di comando tramite un altro flag, all'avvio del browser.
- **Limitazioni delle API di PeerTube**: PeerTube fornisce delle API per la gestione della piattaforma, moderazione dei video, e altro ancora, ma non fornisce API per la raccolta delle metriche dei video. Alcune metriche generiche sono esposte via OpenTelemetry ma non sono specifiche per un singolo utente o video. Per ovviare a questo problema, abbiamo utilizzato il data scraping per estrarre le metriche direttamente dalla pagina web.

- **Gestione delle variabili d'ambiente nell'estensione Chrome**: WebRTC Internals Exporter è un'estensione per Chromium che consente di esportare le metriche WebRTC da un browser in un formato leggibile e analizzabile. Tuttavia, l'estensione non supporta la sostituzione delle variabili d'ambiente nel codice JavaScript, in quanto viene caricata ed eseguita runtime in un ambiente isolato, rendendo difficile la configurazione dinamica dell'estensione. L'utilizzo di WebPack ci consente di configurare l'estensione in fase di compilazione, sostituendo le variabili d'ambiente con i valori corretti prima di caricarla nel browser.

- **Raccolta dei dati dei test su macchine virtuali distribuite**: Data la distribuzione geografica delle nostre macchine virtuali di test, abbiamo dovuto affrontare la sfida di raccogliere e aggregare i dati in modo efficiente e coordinato. Questa sfida l'abbiamo risolta utilizzando Telegraf come agente di raccolta dati su ogni VM, configurandolo per inviare le metriche a un'istanza centrale di MongoDB. MongoDB è stato scelto per il suo supporto nativo a documenti JSON e per la capacità di gestire grandi volumi di dati time-series provenienti da fonti eterogenee.Le API di Hetzner Cloud ci hanno permesso di automatizzare il deployment e la configurazione di tutte queste componenti su macchine distribuite in diverse regioni geografiche.

4.3.2 Python script

Il funzionamento dello script è relativamente semplice; vengono eseguiti dei passaggi in successione per arrivare in una situazione in cui i dati sono costantemente raccolti e inviati al database, creando quindi un ciclo infinito che si ripete ogni 2 secondi circa, fin quando il test non viene fermato.

Importante particolarità dello script è la possibilità di creare e caricare plugin personalizzati che permettono di estrarre metriche di vario tipo da potenzialmente qualsiasi pagina web e non solo da PeerTube. Ciò potenzialmente permette di estendere il sistema di test a qualsiasi altra piattaforma di video streaming, o di qualsiasi altro tipo, semplicemente creando un plugin personalizzato. Questa funzionalità è stata implementata utilizzando le *Abstract Base Classes* di Python, che permettono di definire delle classi base che devono essere implementate dalle classi derivate, e che permettono di definire delle interfacce comuni.

```python
# Abstract Base Classes for Plugins
class StatsSetupPlugin(abc.ABC):
    @abc.abstractmethod
    def setup_stats(
        self,
        driver: webdriver.Remote | webdriver.Chrome,
        url: str,
        retries: int = 5
    ) -> webdriver.Remote | webdriver.Chrome:
        pass

class StatsDownloadPlugin(abc.ABC):
    @abc.abstractmethod
    def download_stats(
        self,
        driver: webdriver.Remote | webdriver.Chrome,
        peersDict: dict,
        socket_url: str,
        socket_port: int
    ):
        pass

    @staticmethod
    def saveStats(stats: list, socket_url: str, socket_port: int):
        try:
            sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
            logger.debug(f'Saving stats: {json.dumps(stats, indent=4)}')
            sock.sendto(json.dumps(stats).encode(), (socket_url, socket_port))
            sock.close()
            logger.debug('Sent stats to socket.')
        except socket.error as e:
            logger.error(f'Got socket error: {e}')
```

Listing 4.3: Abstract Base Classes

Per tutto questo, alla creazione, abbiamo fatto uso di alcune librerie Python tra cui: *selenium* e *beautifulsoup4*.

BeautifulSoup è una libreria Python che permette di estrarre dati da file HTML e XML. È in grado di navigare nella struttura del documento, estrarre i dati, modificarli e salvare le modifiche. È spesso utilizzata per fare *web scraping*. Assieme, quste due librerie ci permettono di interagire con il browser e di estrarre i dati necessari dalla pagina web di PeerTube [32].

I passaggi presenti in figura 4.12 possono essere riassunti come:

- **Inizializzazione**: Viene impostata da riga di comando o tramite variabili d'ambiente la configurazione del test.

  Queste possono includere: I plugin personalizzati BeautifulSoup, l'URL del video da analizzare, l'URL del server Telegraf, l'URL dell'hub di Selenium Grid, il percorso dell'estensione di WebRTC Internals Exporter, e altro ancora.

- **Avvio del browser**: Viene avviato il browser e aperta la pagina del video specificato tramite Selenium, con l'estensione di WebRTC Internals Exporter installata ed abilitata. Viene anche avviato in ascolto un server HTTP in background per ricevere le metriche WebRTC tramite un endpoint locale POST dedicato.

- **Avvio del test**: Viene caricata la pagina del video e tramite Selenium viene controllato che il video sia in riproduzione tramite il controllo dell'esistenza del bottone play.

- **Raccolta delle metriche WebRTC**: L'estensione per WebRTC inizia a raccogliere le metriche e ogni 2 secondi circa le invia allo script principale. Viene quindi di fatto instaurato un ciclo che viene usato dallo script come sorgente di clock per la raccolta delle altre metriche.

- **Raccolta delle metriche video di PeerTube**: Utilizzando BeautifulSoup, vengono trasformate e raccolte le metriche del player video di PeerTube, come la qualità del video, il buffering, quantità di dati trasferiti, la latenza e altro ancora. Il tutto viene convertito in un formato JSON.

- **Invio delle metriche a Telegraf**: Le metriche raccolte, vengono inviate a Telegraf tramite una socket di sistema, per l'elaborazione e l'invio al database finale, nel nostro caso MongoDB. C'è un passaggio fondamentale però che Telegraf esegue, ed è quello di fare *string-join* dei dati JSON, ovvero di 'appiattire' e trasformare i dati in una stringa per poi inviarli al database. Questo viene fatto in quanto Telegraf, utilizzando il plugin di input per il formato JSON, converte gli array **in singole metriche separate**, e non come un unico documento, perdendo quindi la struttura originale dei dati. In sé questo non è un problema, ma per la nostra analisi è fondamentale mantenere la struttura originale dei dati per facilitarci il lavoro. Infine, per ricavare i documenti interi, un ulteriore passaggio finale va effettuato nel database, in differita, per convertirli da stringhe a oggetti JSON validi.

Va inoltre evidenziata la scelta di utilizzare un solo script Python per coordinare il tutto, sia l'inizializzazione sia raccolta delle metriche dalle pagine web e dalla estensione esterna, in quanto questo ha semplificato la gestione e soprattutto la creazione del sistema e permette come bonus di avere un unico punto di controllo per l'intero processo.

```python
# Default Plugin Implementation
class DefaultStatsSetupPlugin(StatsSetupPlugin):
    def setup_stats(
        self, driver: webdriver.Remote,
        url: str,
        retries: int = 5
    ) -> webdriver.Remote:
        logger.log(logging.INFO, 'Setting up stats.')
        actions = ActionChains(driver)
        wait = WebDriverWait(driver, 30, poll_frequency=0.2)

        sleep(2)

        for attempt in range(retries):
            driver.get(url)
            try:
                wait.until(
                    ec.presence_of_element_located(
                        (By.CLASS_NAME, 'vjs-big-play-button')
                    )
                )
                break
            except Exception:
                logger.error(
                    f'Timeout. Attempt {attempt + 1} of {retries}'
                    )
                if attempt == retries - 1:
                    logger.error('Timeout limit reached. Exiting.')
                    driver.quit()
                    raise SystemExit(1)

        actions.click(
            driver.find_element(By.CLASS_NAME ,'video-js')
        ).perform()
        wait.until(
            ec.visibility_of_element_located((By.CLASS_NAME, 'vjs-control-bar'))
        )
        actions.context_click(
            driver.find_element(By.CLASS_NAME ,'video-js')
            .perform()
        )
        statsForNerds = driver.find_elements(
            By.CLASS_NAME ,'vjs-menu-item'

        actions.click(statsForNerds[-1]).perform()
        wait.until(
            ec.presence_of_element_located(
                (By.CSS_SELECTOR, 'div.vjs-stats-content[style="display: block;"]'
                    )
            )
        )
        actions.move_to_element(
            driver.find_element(By.CLASS_NAME ,'vjs-control-bar')
        ).perform()

        logger.log(logging.INFO, 'Stats setup complete.')

        return driver
```

Listing 4.4: Codice del plugin per la raccolta delle metriche di PeerTube

Il diagramma finale dell'architettura del nostro sistema è il seguente:

- Un server centrale che esegue un'istanza di PeerTube e che raccoglie le metriche fornite da OpenTelemetry.
- Molteplici macchine virtuali distribuite geograficamente sul globo che simulano gli spettatori.
- Una applicazione Python in esecuzione su ogni singola macchina, che coordina Selenium e Telegraf per raccogliere le statistiche del video in riproduzione e le metriche WebRTC.
- Un database centralizzato per la raccolta dei dati.

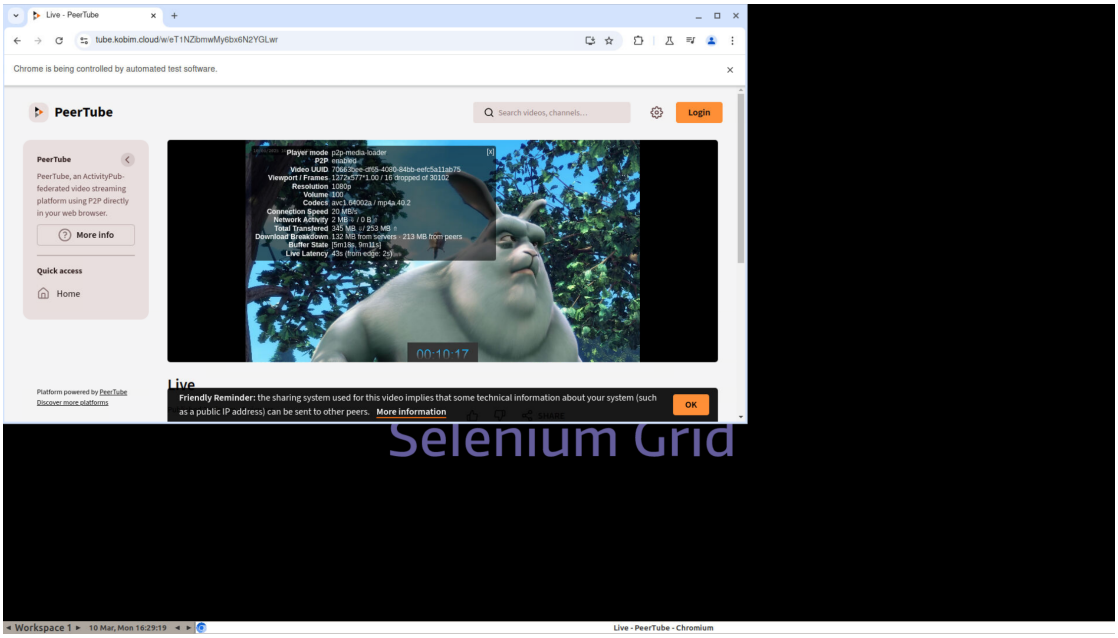

Figure 4.13. Istanza di Selenium Standalone con il collector in esecuzione.

4.4 Casi d'uso e scenari di test riprodotti

Come descritto in precedenza, abbiamo riprodotto due scenari di test principali dei 4 casi d'uso descritti nell'articolo di PeerTube per motivi di tempo e risorse. Questi sono:

- Live streaming con impostazione *Normal Latency*: Il setup standard di PeerTube

- Live streaming con impostazione *High Latency*: La configurazione che privilegia il P2P a scapito della latenza

Per ciascun scenario, abbiamo impostato il numero di client a 5 (limite di Hetzner Cloud), distribuiti per il globo, e raccolto le seguenti statistiche:

- Statistiche del player web (risoluzione, bitrate, buffering, latenza, ecc.)
- Statistiche WebRTC (peer connection, ICE candidates, ICE connection state, ecc.)
- Informazioni identificative dei singoli peer (url del video, ID sessione, ID peer)
- Timestamp di raccolta

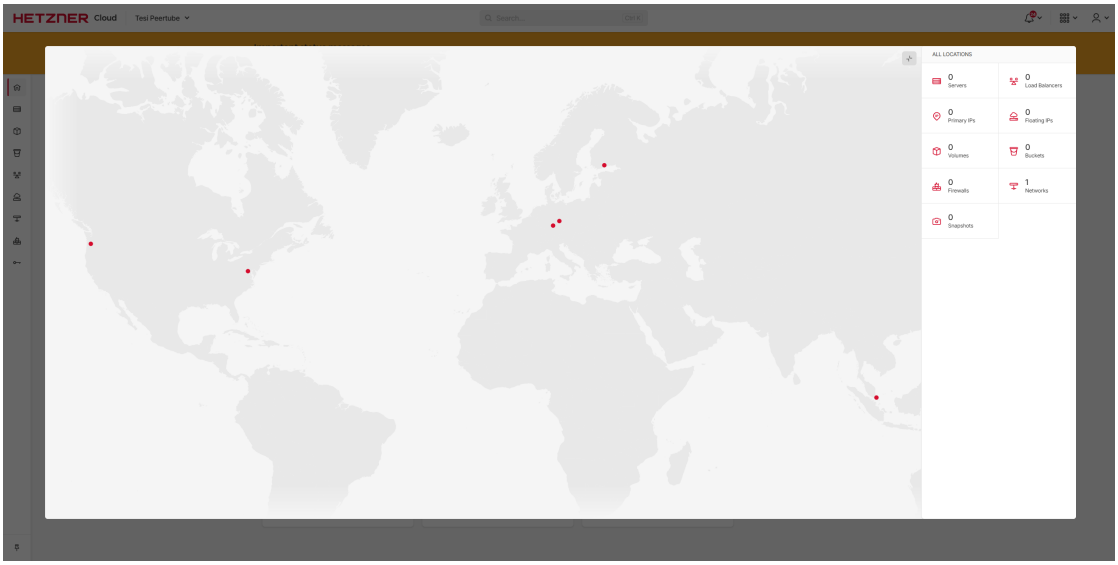

Figure 4.14. Distribuzione geografica delle macchine virtuali Hetzner Cloud.

Capitolo 5

Conclusioni

5.1 Risultati e analisi

Attraverso il nostro sistema di test automatizzato, siamo riusciti a raccogliere i dati che ci hanno permesso di analizzare le prestazioni del sistema P2P di PeerTube. I risultati principali confermano molte delle affermazioni fatte dagli sviluppatori nella loro pubblicazione originale.

I nostri dati raccolti arrivano da più fonti differenti; per verificare le affermazioni originali, ci siamo avvalsi nell'utilizzare sia i dati raccolti dal sistema di monitoraggio interno di PeerTube (ovver OpenTelemetry), sia i dati raccolti tramite il nostro sistema di test automatizzato esterno ed indipendente. Questo ci ha permesso di avere una visione più completa e dettagliata delle prestazioni del sistema P2P e soprattuto permette la verifica di soluzioni diverse che non includono un sistema di monitoraggio interno.

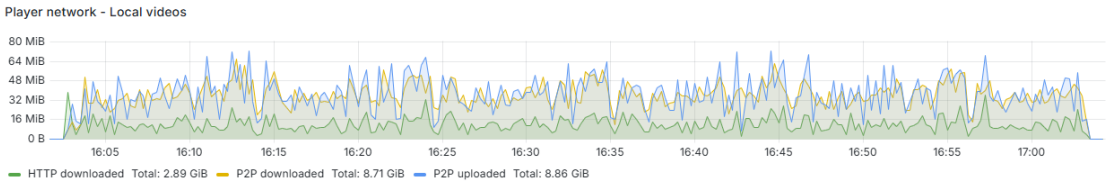

Figure 5.1. Grafico nel tempo del traffico HTTP e P2P per la configurazione Normal Latency raccolto da Grafana/OpenTelemetry.

Nella figura 5.1 sono mostrati i grafici del traffico HTTP e P2P nel tempo raccolti da automaticamente da OpenTelemetry per la configurazione Normal Latency. Come si evidenzia, il traffico P2P totale è nettamente superiore al traffico HTTP, con un rapporto di circa 3:1.

Questo viene ulteriormente confermato dai nostri dati raccolti esternamente da Selenium sulle statistiche messe a disposizione dal player video, che mostrano un traffico P2P che raggiunge il 75% del traffico totale in download. Questo è un risultato molto positivo, in quanto dimostra che il sistema P2P di PeerTube è in grado di ridurre significativamente il carico sui server centrali, permettendo una distribuzione più efficiente dei contenuti video.

Tabella 5.1: Traffico totale in download per la configurazione Normal Latency.

Metrica	**Valore**	**Percentuale**
Download totale da peer	8.10 GiB	75%
Download totale da server	2.75 GiB	25%
Download totale	10.86 GiB	100%
Rapporto P2P/Server	3:1	

I valori non sono strettamente identici a quelli mostrati in figura 5.1 in quanto sono stati raccolti in con metodologie diverse, ma sono comunque molto vicini e confermano i risultati ottenuti.

Ulteriore conferma arriva dai valori raccolti da WebRTC e il campo *datachannel*, che mostrano un traffico P2P in download totale di 8.80 GiB, leggermente maggiore rispetto a quelli forniti dal player/OpenTelemetry, ma pur sempre con un rapporto di 3:1 rispetto al traffico HTTP totale.

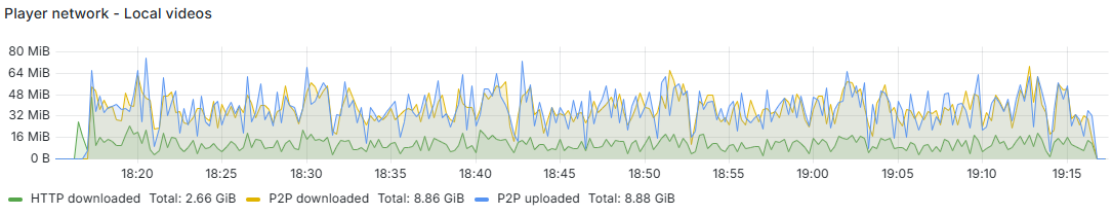

Figure 5.2. Grafico nel tempo del traffico HTTP e P2P per la configurazione High Latency raccolto da Grafana/OpenTelemetry.

Stesso discorso vale per la configurazione High Latency, mostrata in figura 5.2, dove il traffico P2P è nettamente superiore al traffico HTTP, con un rapporto di circa 3:1. In questo caso, i dati raccolti dal player video mostrano un traffico P2P che raggiunge il 77% del traffico totale in download, leggermente superiore rispetto alla configurazione Normal Latency ma non sorprendente, in quanto i nostri test sono stati eseguiti in un ambiente di rete ideale e con pochi client simultanei rispetto a quelli mostrati nell'articolo originale.

Tabella 5.2: Traffico totale in download per la configurazione High Latency.

Metrica	**Valore**	**Percentuale**
Download totale da peer	8.28 GiB	77%
Download totale da server	2.53 GiB	23%
Download totale	10.82 GiB	100%
Rapporto P2P/Server	3:1	

Il traffico misurato da WebRTC e il campo *data-channel* mostrano un traffico P2P in download totale di 8.84 GiB, con un rapporto di 3:1 rispetto al traffico HTTP totale, anche in questo caso leggermente superiore rispetto alla configurazione Normal Latency.

$$\text{P2P/Server Ratio} = \frac{\text{P2P Traffic}}{\text{HTTP Traffic}} \approx 3 \tag{5.1}$$

$$\text{P2P Percentage} = \frac{\text{P2P Traffic}}{\text{Total Traffic}} \times 100\% \approx 75\% \tag{5.2}$$

Figure 5.3. Formule matematiche per il calcolo del rapporto P2P/Server e della percentuale di traffico P2P.

Infine, i dati raccolti dai nostri test mostrano che:

- Il traffico P2P aumenta progressivamente nel tempo, confermando il comportamento descritto nell'articolo originale
- Nei test con High Latency, il traffico P2P ha raggiunto fino al 77% del traffico totale dopo qualche minuto di riproduzione.
- Con Normal Latency, abbiamo osservato un comportamento simile ma con una curva di adozione P2P più graduale, raggiungendo circa il 75% di traffico P2P in download.

```
[
    {
        $group: {
            _id: "$tags.session",
            maxDownPeers: {
                $max: "$player.Download Breakdown.Peers"
            },
            maxDownServer: {
                $max: "$player.Download Breakdown.Server"
            },
        }
    },
    {
        $group: {
            _id: null,
            totalDownPeers: {
                $sum: "$maxDownPeers"
            },
            totalDownServer: {
                $sum: "$maxDownServer"
            }
        }
    },
    {
        $project: {
            _id: 0,
            totalDownPeers: 1,
            totalDownServer: 1,
            totalComputedDown: {
                $sum: [
                    "$totalDownPeers",
                    "$totalDownServer"
                ]
            }
        }
    },
    {
        $project: {
            totalDownPeers: 1,
            totalDownServer: 1,
            totalComputedDown: 1,
            percentageOfTotalPeers: {
                $round: {
                    $multiply: [
                        {
                            $divide: [
                                "$totalDownPeers",
                                "$totalComputedDown"
                            ]
                        },
                        100
                    ]
                }
            },
            percentageOfTotalServer: {
                $round: {
                    $multiply: [
                        {
                            $divide: [
                                "$totalDownServer",
                                "$totalComputedDown"
```

```
                                ]
                            },
                            100
                        ]
                    }
                },
                ratio: {
                    $round: {
                        $divide: [
                            "$totalDownPeers",
                            "$totalDownServer"
                        ]
                    }
                }
            }
        },
        {
            $sort: {
                ratio: -1
            }
        },
        {
            $limit: 10
        }
    ]
```

Listing 5.1: Query MongoDB usata per calcolare il traffico totale in download

Tutti i dataset e le query MongoDB usate per l'analisi dei dati sono disponibili pubblicamente qui:

- https://gitlab.di.unimi.it/mirko.milovanovic/peertube-collector/-/tree/main/server/peertube%20data
- https://gitlab.di.unimi.it/mirko.milovanovic/Tesi/-/tree/main/peertube/datavis/CRUD

5.2 Limitazioni dello studio

È importante evidenziare alcune limitazioni del nostro approccio:

- A differenza dei test di PeerTube che utilizzavano 1000 client simultanei, abbiamo potuto testare con un numero limitato di client a causa di vincoli di risorse.
- I nostri client erano distribuiti su data center pubblici con connessioni di qualità superiore rispetto a quelle tipiche degli utenti finali
- Non abbiamo potuto testare tutte le combinazioni possibili di configurazioni di rete (diversi tipi di NAT, firewall, ecc.).
- La nostra analisi si è concentrata principalmente sulle metriche di rete e non sugli aspetti soggettivi dell'esperienza utente.

5.3 Prospettive future

Questo lavoro apre diverse possibilità per ricerche future:

- Estendere i test con un numero maggiore di client distribuiti in ambienti di rete più realistici.
- Confrontare PeerTube con altre soluzioni P2P per lo streaming video.
- Sviluppare metriche standardizzate per valutare l'efficienza dei sistemi di streaming decentralizzato.
- Studiare l'impatto di implementazioni WebRTC alternative.
- Analizzare i dati aggiuntivi raccolti da WebRTC Internals Exporter per ottenere informazioni più dettagliate sulle prestazioni del sistema P2P come ad esempio: il grafo delle connessioni, la latenza, e altro ancora.

5.4 Considerazioni finali

Il nostro lavoro ha confermato che PeerTube rappresenta una soluzione tecnica efficace per la distribuzione decentralizzata di contenuti video. L'implementazione P2P basata su WebRTC offre vantaggi tangibili in termini di riduzione del carico sui server centrali, mantenendo una qualità di streaming adeguata.

La configurazione High Latency si è dimostrata particolarmente efficace nel massimizzare il traffico P2P, rappresentando un ottimo compromesso per contenuti dove la latenza non è critica. Il sistema ha dimostrato una buona resilienza e capacità di adattamento a diverse condizioni di rete.

In un'epoca in cui la centralizzazione dei servizi di streaming solleva preoccupazioni riguardo privacy, censura e sostenibilità economica, PeerTube offre un'alternativa concreta e funzionante. Il nostro studio conferma che, nonostante alcune limitazioni tecniche, l'approccio decentralizzato allo streaming video non solo è tecnicamente possibile ma anche efficiente in termini di utilizzo delle risorse.

Ma soprattutto la metodologia e gli strumenti sviluppati per questo studio potranno essere utilizzati per valutare future implementazioni di tecnologie di streaming decentralizzato basate su WebRTC, contribuendo all'evoluzione di Internet verso un ecosistema più equo e distribuito.

Bibliografia

[9] Kristina Chodorow. *MongoDB: The Definitive Guide*. O'Reilly Media, Inc., 2013. ISBN: 1449344682.

[12] The 360 Degree. *Own work*. Licensed under CC BY-SA 4.0. 2022. URL: https://commons.wikimedia.org/w/index.php?curid=116056511.

[29] Muqaddas Naz, Nadeem Javaid e Sohail Iqbal. «Research Based Data Rights Management Using Blockchain Over Ethereum Network». Tesi di dott. Set. 2019.

[32] Sakshi Pant et al. «Web Scraping Using Beautiful Soup». In: *2024 International Conference on Knowledge Engineering and Communication Systems (ICKECS)*. Vol. 1. 2024, pp. 1–6. DOI: 10.1109/ICKECS61492.2024.10617017.

[42] Guido Van Rossum e Fred L. Drake. *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009. ISBN: 1441412697.

[43] David Vignoni e Calimo. *By Gnome-fs-client.svg: David Vignoni, Gnome-fs-server.svg: David Vignoni, derivative work: Calimo*. Licensed under LGPL. 2009. URL: https://commons.wikimedia.org/w/index.php?curid=15782858.

Sitografia

[1] *About-tahoe; Tahoe-LAFS — tahoe-lafs.org*. https://www.tahoe-lafs.org/trac/tahoe-lafs/browser/trunk/docs/about-tahoe.rst. [Accessed 02-Apr-2023].

[2] *Ace Stream — web.archive.org*. [Accessed 28-09-2023]. URL: https://web.archive.org/web/20180618052904/http://info.acestream.org/#/about/acestream.

[3] *ActivityPub Rocks! — activitypub.rocks*. https://activitypub.rocks/. [Accessed 04-Apr-2023].

[4] *Aggregation in MongoDB - GeeksforGeeks — geeksforgeeks.org*. https://www.geeksforgeeks.org/aggregation-in-mongodb/. [Accessed 01-04-2025].

[5] *Announcement! ACE Stream; New era of TV and Internet broadcasting — oldforum.acestream.media*. [Accessed 28-09-2023]. URL: http://oldforum.acestream.media/index.php?topic=1479.0.

[6] Juan Benet. *IPFS - Content Addressed, Versioned, P2P File System*. 2014. DOI: 10.48550/ARXIV.1407.3561. URL: https://arxiv.org/abs/1407.3561.

[7] *BitTorrentSpecification - TheoryOrg — wiki.theory.org*. https://wiki.theory.org/BitTorrentSpecification. [Accessed 18-03-2025].

[8] *Build the backend services needed for a WebRTC app — Articles — web.dev — web.dev*. https://web.dev/articles/webrtc-infrastructure#what-is-signaling. [Accessed 18-03-2025].

[10] *Concepts — webpack — webpack.js.org*. https://webpack.js.org/concepts/. [Accessed 22-03-2025].

[11] *Containerization using Docker - GeeksforGeeks — geeksforgeeks.org*. https://www.geeksforgeeks.org/containerization-using-docker/. [Accessed 18-03-2025].

[13] *Docker Networking - GeeksforGeeks — geeksforgeeks.org*. https://www.geeksforgeeks.org/basics-of-docker-networking/. [Accessed 18-03-2025].

[14] *Docker overview — docker-docs.uclv.cu*. https://docker-docs.uclv.cu/get-started/overview/. [Accessed 01-04-2025].

[15] *Docker vs Docker Compose vs Docker Swarm s docker engine — innovationyourself.com*. https://innovationyourself.com/docker-vs-docker-compose-vs-docker-swarm/. [Accessed 01-04-2025].

[16] *Documents - MongoDB Manual v8.0 - MongoDB Docs — mongodb.com*. https://www.mongodb.com/docs/manual/core/document/. [Accessed 19-03-2025].

[17] *FAQ — JoinPeerTube — joinpeertube.org*. [Accessed 04-Apr-2023]. URL: https://joinpeertube.org/faq#what-are-the-main-advantages-of-peertube.

[18] *Framasoft / PeerTube / Selenium stack · GitLab — framagit.org*. https://framagit.org/framasoft/peertube/selenium-stack. [Accessed 20-03-2025].

[19] *GitHub - Chocobozzz/PeerTube: ActivityPub-federated video streaming platform using P2P directly in your web browser — github.com*. https://github.com/Chocobozzz/PeerTube. [Accessed 24-03-2025].

[20] *GitHub - hetznercloud/cli: A command-line interface for Hetzner Cloud — github.com*. https://github.com/hetznercloud/cli. [Accessed 20-03-2025].

[21] *GitHub - influxdata/telegraf: Agent for collecting, processing, aggregating, and writing metrics, logs, and other arbitrary data. — github.com*. https://github.com/influxdata/telegraf. [Accessed 19-03-2025].

[22] *GitHub - webtorrent/bittorrent-tracker: Simple, robust, BitTorrent tracker (client & server) implementation — github.com*. https://github.com/webtorrent/bittorrent-tracker?tab=readme-ov-file. [Accessed 18-03-2025].

[23] Matt Holdrege e Pyda Srisuresh. *IP Network Address Translator (NAT) Terminology and Considerations*. RFC 2663. Ago. 1999. DOI: 10.17487/RFC2663. URL: https://www.rfc-editor.org/info/rfc2663.

[24] *How NAT traversal works — tailscale.com*. https://tailscale.com/blog/how-nat-traversal-works. [Accessed 18-03-2025].

[25] *Infrastructure Monitoring Basics with Telegraf, InfluxDB, and Grafana — influxdata.com*. https://www.influxdata.com/blog/infrastructure-monitoring-basics-telegraf-influxdb-grafana/. [Accessed 01-04-2025].

[26] Simran Kumari. *Which Browser Supports Selenium ? - Scaler Topics — scaler.com*. https://www.scaler.com/topics/selenium-tutorial/which-browser-supports-selenium/. [Accessed 20-03-2025].

[27] Angelo Lazzari. *Time series: una piccola, ma dettagliata, introduzione - AIKnow — aiknow.io*. https://www.aiknow.io/cosa-sono-le-time-series/?doing_wp_cron=1742414242.9129240512847900390625. [Accessed 19-03-2025].

[28] Philip Matthews et al. *Session Traversal Utilities for NAT (STUN)*. RFC 5389. Ott. 2008. DOI: 10.17487/RFC5389. URL: https://www.rfc-editor.org/info/rfc5389.

[30] Franco Palermo. *Azure Functions with Docker — franco.palermo812*. https://medium.com/@franco.palermo812/azure-functions-with-docker-47e22866330. [Accessed 18-03-2025].

[31] Vittorio Palmisano. *WebRTC debugging with Prometheus/Grafana — vpalmisano*. https://medium.com/@vpalmisano/webrtc-debugging-with-prometheus-grafana-254b6ac71063. [Accessed 20-03-2025].

[33] Roger Pantos e William May. *HTTP Live Streaming*. RFC 8216. Ago. 2017. DOI: 10.17487/RFC8216. URL: https://www.rfc-editor.org/info/rfc8216.

[34] *PeerTube stress tests: resilience lies in your peers! — JoinPeerTube — joinpeertube.org*. https://joinpeertube.org/news/stress-test-2023. [Accessed 18-03-2025].

[35] *PeerTube v6 is out, and powered by your ideas ! — framablog.org*. https://framablog.org/2023/11/28/peertube-v6-is-out-and-powered-by-your-ideas/#-and-there-s-always-more-. [Accessed 18-03-2025].

[36] *Selenium Grid 4 - Architecture and Communication — qabrains.com*. https://qabrains.com/selenium-grid-4-architecture-and-communication. [Accessed 01-04-2025].

[37] *Selenium WebDriver – Step-by-Step Tutorial*. Set. 2024. URL: https://testgrid.io/blog/selenium-webdriver/.

[38] *Telegraf documentation*. URL: https://docs.influxdata.com/telegraf/v1/ (visitato il giorno 19/03/2025).

[39] *The Best Introduction to MongoDB Query Language (MQL) — knowi.com*. https://www.knowi.com/blog/the-best-introduction-to-mongodb-query-language-mql/. [Accessed 19-03-2025].

[40] *The Selenium Browser Automation Project — selenium.dev*. https://www.selenium.dev/documentation/. [Accessed 20-03-2025].

[41] *Tor at the Heart: Tahoe-LAFS — Tor Project — blog.torproject.org*. https://blog.torproject.org/tor-heart-tahoe-lafs/. [Accessed 02-Apr-2023].

[44] W3C. *What is the difference between the Web and the Internet?* https://www.w3.org/Help/. [Accessed 31-Aug-2022]. 2021.

[45] *What is IPFS? — IPFS Docs — docs.ipfs.tech*. https://docs.ipfs.tech/concepts/what-is-ipfs/. [Accessed 04-Apr-2023].

[46] *What is OpenTelemetry? — opentelemetry.io*. https://opentelemetry.io/docs/what-is-opentelemetry/. [Accessed 25-03-2025].

[47] Wikipedia. *ActivityPub — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=ActivityPub&oldid=1134432238. [Online; accessed 05-April-2023]. 2023.

[48] Wikipedia. *Broadcasting (networking) — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Broadcasting%20(networking)&oldid=1238402634. [Online; accessed 31-March-2025]. 2025.

[49] Wikipedia. *Content delivery network — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Content%20delivery%20network&oldid=1279155231. [Online; accessed 31-March-2025]. 2025.

[50] Wikipedia. *Multicast — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Multicast&oldid=1270123732. [Online; accessed 31-March-2025]. 2025.

[51] Wikipedia. *Network address translation — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Network%20address%20translation&oldid=1283227709. [Online; accessed 01-April-2025]. 2025.

[52] Wikipedia. *PeerTube — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=PeerTube&oldid=1281511347. [Online; accessed 24-March-2025]. 2025.

[53] Wikipedia. *Unicast — Wikipedia, The Free Encyclopedia*. http://en.wikipedia.org/w/index.php?title=Unicast&oldid=1222709440. [Online; accessed 31-March-2025]. 2025.