Alma Mater Studiorum · Università di Bologna

Software Architectures Course

TWITCH

Architecture Overview

Professore:
Paolo Ciancarini

Simone Bonfante
Matricola: 819606

Anno Accademico 2018-2019

Indice

Capitolo 1

Introduzione

1.1 Cos'è Twitch

Twitch è un popolare servizio online per la visione e lo streaming di trasmissioni video digitali. Quando è stato fondato nel 2011, Twitch inizialmente si è concentrato quasi interamente sui videogiochi, ma da allora si è espanso per includere stream dedicati alla creazione di opere d'arte, musica, talk show e occasionali serie TV.

Figura 1.1: Dati Twitch

Twitch è la piattaforma video e community leader a livello mondiale per i giocatori. Oltre 140 milioni di giocatori si riuniscono ogni mese su Twitch per trasmettere, guardare e chattare sui giochi. La piattaforma video di Twitch è la spina dorsale della distribuzione live e on-demand per l'intero ecosistema dei videogiochi. Ciò include editori di giochi, sviluppatori,

media, convenzioni di settore e conferenze stampa, giocatori occasionali e giochi per eventi di beneficenza.

Il servizio di streaming vanta oltre 2 milioni di streamers unici ogni mese e oltre 17mila di questi utenti guadagnano denaro tramite il programma Twitch Partner su un totale di 2,2 milioni di broadcasters mensili unici; un servizio che fornisce agli streamer funzionalità aggiuntive come abbonamenti a pagamento e posizionamenti di annunci. Twitch è stata acquistata da Amazon nel 2014 e rimane una delle più alte fonti di traffico Internet del Nord America, addirittura quarta!

Twitch è più di uno streamer di game

Twitch potrebbe aver iniziato come servizio di streaming di videogiochi, ma da allora è stato ampliato e ora offre una varietà di flussi live diversi, destinati ad attrarre un pubblico più ampio. La categoria non di gioco più popolare è IRL (In Real Life) che offre agli streamer semplicemente un servizio di chatting con i propri spettatori in tempo reale. Talk Shows è un'altra popolare opzione non di gioco che contiene un mix di dibattiti dal vivo, podcast e persino sfilate di varietà prodotte professionalmente.

Gli spettatori che cercano qualcosa di un po più artistico dovrebbero consultare la categoria Creative. E' qui che artisti, programmatori, animatori, cosplayer e designer condividono il loro processo creativo con il mondo e questi flussi di solito attraggono un pubblico molto diverso rispetto a quelli che guardano altre categorie.

1.2 Overview architetturale

Come accennato precedente Twitch è la piattaforma leader a livello mondiale per lo streaming live, soprattutto per il gaming. In questa sezione si introduce un'analisi sull'architettura del sistema, la quale poi verrà approfondita nei capitoli seguenti.

- Uno dei più grandi sistemi di distribuzione di live video al mondo

- Un sistema di chat real time
- Servizi Web che forniscono accesso a funzionalità e dati
- Sistemi di archiviazione dati
- Applicazioni client web-based e su una moltitudine di piattaforme, come console (PS4, XBOX ONE) e dispositivi mobili
- Infrastruttura di data science

Ognuno di questi diversi componenti di Twitch presenta diverse sfide, tra cui l'ottimizzazione a basso livello della codifica video, la scalabilità del sistema su larga scala e l'implementazione di prodotti su più piattaforme.

La velocità esponenziale della crescita di Twitch ha portato all'adozione di nuove tecnologie. La startup a soli due anni dalla sua nascita, nel 2011, ha raggiunto numeri impressionanti come 45 milioni di attivi al mese, 12 milioni di minuti mensili di streaming visualizzati, 900k streamers mensili.

Ad oggi i numeri di twitch sono spaventosi: 560 miliardi di minuit guardati (+58percento rispetto al 2017), 3.4 milioni di broadcaster unici ogni mese (+70percento), 1,007,000 è la media dei visitatori nel 2018 e addirittura 41,000 i canali live concorrenti.
Solo a novembre 2018 sono state spese quasi 900 milioni di ore a guardare Twitch [10].

Video System

Sicuramente il componente principale della piattaforma è il sistema video. Sostanzialmente ha il compito di trasmettere il video dal broadcaster ai suoi viewers.
Twitch utilizza lo streaming RTMP (Real Time Messaging Protocol su HTTP) coprendo 14 regioni, per compensare i punti deboli delle fonti come, per esempio, fluttuazione e basse prestazioni della rete. Lo streaming originale è trasferito attraverso HTTP Live Streaming da server di streaming

agli spettatori.
Nel capitolo due verranno analizzati i meccanismi di distribuzione e scalabilità del sistema.
I componenti principali che compongono l'architettura video:

- **Video ingest:** Vengono raccolti video RTMP per poi essere trasportati al sistema di transcodifica.
- **Transcode System:** Viene preso il flusso RTMP (real time messaging Protocol) in arrivo dall'emittente per poi essere transcodificato in più streams HLS (HTTP Live Streaming). Questo è implementato tramite una combinazione di C / C ++ e Go.
- **Distribution ed Edge:** distribuiscono i flussi HLS ai POP (point of presence) geograficamente disparati, in modo da avere un'esperienza di streaming video di altissima qualità. Di nuovo, per lo più scritto in Go.
- **VOD (Video on Demand):** tutti i sistemi video in arrivo e archiviati per il sistema VOD.

Chat

La comunicazione interattiva è una caratteristica unica in Twitch. Un insieme di server di messaggistica interattivo riceve i messaggi in tempo reale dello spettatore, quindi invia i messaggi all'emittente corrispondente e ad altri spettatori, migliorando l'esperienza dei partecipanti per gli eventi dal vivo verso un ambiente di concorrenza realistico. Detto questo, gli spettatori non sono più passivi, ma possono influenzare anche il progresso della trasmissione. In particolare, per la trasmissione di giochi dal vivo, il servizio interattivo consente agli spettatori di interagire con i giocatori e i commentatori in tempo reale.

Il sistema di chatting è un sistema distribuito in tempo reale altamente scalabile scritto in Go. Offre centinaia di miliardi di messaggi al giorno agli utenti che guardano i video tramite i protocolli Twitch, oltre a supportare l'IRC

(internet real chat) come protocollo di comunicazione, rendendo facile agli sviluppatori la creazione di bot IRC per aggiungere le proprie funzionalità di chat personalizzate. I componenti principali della chat includono:

- **Edge:** riceve e distribuisce messaggi tra client e servizi di back-end. Edge parla il protocollo IRC sia su TCP che WebSocket.
- **Pubsub:** distribuisce i messaggi internamente attraverso i nodi Edge. Pubsub e Edge si combinano per formare un sistema gerarchico di distribuzione dei messaggi.
- **Clue:** esegue analisi sulle azioni degli utenti. L'utente è bannato nel canale in cui sta parlando? Sono abbonati? Sono vietati o esibiscono comportamenti abusivi? Per informare queste decisioni, Clue aggrega i dati attraverso molte fonti, inclusi database, API interne e cache.
- **Room:** responsabile per l'elenco degli spettatori. Room aggrega, archivia e invia dati delle iscrizioni su tutti gli edge per recuperare gli elenchi di visualizzatori per la chat room di ciascun canale.

Web Api

Oltre ai servizi in tempo reale (video e chat) twitch offre anche un numero considerevole di servizi tra cui, ma non solo:

- Web APIs che consentono agli utenti di gestire e personalizzare i loro profili e abbonamenti
- Search e Discovery services che aiutano l'utente a trovare gli streams che sta cercando.
- Revenue system: sono sistemi che consentono di gestire pubblicità e abbonamenti e assicurare che i partner ottengano i loro guadagni

Tutto questo è basato su un mix di Rails, Go e varie applicazioni open source utilizzate per il routing, la memorizzazione nella cache e l'archiviazione dei dati. Sono state anche effettuate numerose partizioni verticali e orizzontali

delle API di dati e dati al fine di migliorare i sistemi, sia a livello tecnico che organizzativo.

A livello fisico, attualmente viene gestito un numero abbastanza elevato di POP "bare metal" (punti di presenza) in tutto il mondo: questo consente di offrire video di qualità superiore a causa delle insolite esigenze di consegna dei video (molta larghezza di banda!).

Ora tutti questi servizi sono stati spostati su Amazon Web Services, il che aiuta a ridurre la quantità di costi operativi, oltre a sfruttare la convenienza e la scalabilità di molti dei loro servizi.

Capitolo 2

Architettura di Twitch

2.1 Contesto

In questa sezione si analizzano i casi d'uso di Twitch individuati a partire dalle features. Si costruisce inoltre un diagramma di contesto che evidenzia le entità che hanno in qualche modo a che fare col servizio.
Twitch può essere usato dagli utenti finali con lo scopo di trasmettere/guardare video in diretta, utilizzare la chat nei canali di streaming e fare donazioni agli streamer preferiti.
Twitch offre un'interfaccia Internet Relay Chat (IRC) per la funzionalità di chat. I chatbot ti consentono di interagire a livello di codice con Twitch utilizzando gli standard IRC; il bot si collega alla rete Twitch IRC come client per eseguire le diverse azioni citate.
Esistono due stati speciali per i bot che migliorano l'esperienza utente di Twitch: conosciuti e verificati. Entrambi gli stati offrono privilegi elevati. I bot verificati hanno un throughput più elevato rispetto agli altri. Lo stato verificato è concesso solo raramente.
Grazie alle API pubbliche fornite da Twitch gli sviluppatori terzi possono interfacciarsi con il sistema creando Bot ed estensioni utilizzabili dagli utenti finali.
Twitch è disponibile come già accennato per diversi sistemi operativi per device desktop o mobile, anche se il client che va per la maggiore è la versione web di Twitch anch'essa ottimizzata per diversi browser. Inoltre è possibile

utilizzare Twitch nelle migliori console di gioco come Play Station, Xbox ma anche dalla propria Smart TV o grazie al collegamento Chromecast dal proprio laptop alla TV.
Infine, nel diagramma di contesto, si tengono in considerazione anche i principali competitor verso cui Twitch guarda per assicurarsi di fornire un servizio competitivo e con features all'avanguardia.

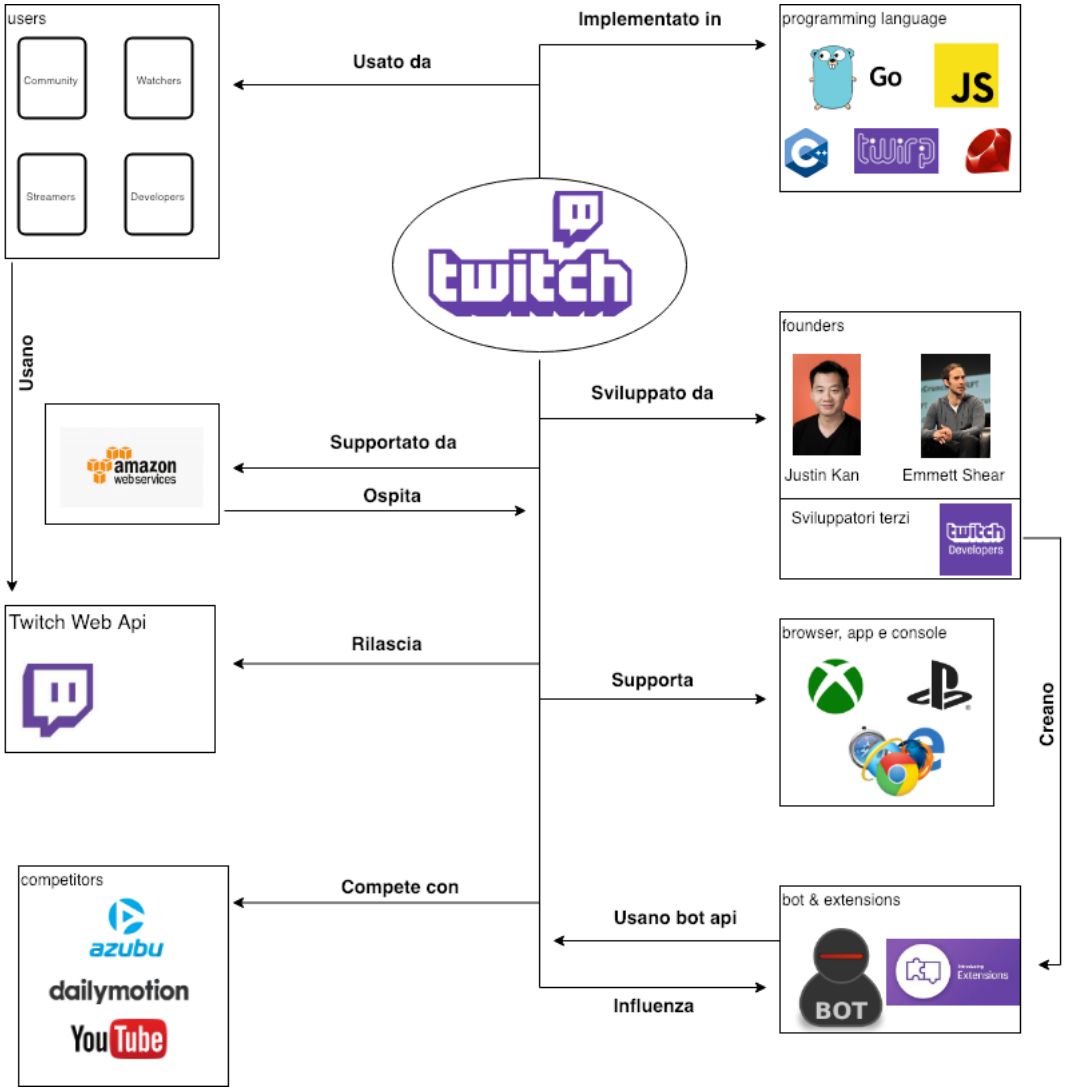

Figura 2.1: Context Diagram

2.2 Proprietà rilevanti: requisiti non funzionali

L'analisi seguente, talvolta, tiene in considerazione aspetti non pubblicamente noti che vengono dedotti sulla base di fatti di cronaca o dallo svolgimento di esperimenti che possono rilevare alcuni aspetti del sistema. Quella del benchmarking è una scienza. Le deduzioni seguenti non hanno la pretesa di essere considerate come assolutamente vere. Lo scopo è quello di fornire indicazioni verosimili allo scopo di valutare il soddisfacimento di alcuni requisiti non funzionali.

2.2.1 Scalabilità e Distribuzione

Uno dei grandi punti deboli di Twitch è l'impossibilità di shiftare nel tempo il contenuto del video, e spesso gli streamers non sono geograficamente vicini o ben connessi con chi guarda.
A differenza di YouTube o Netflix che distribuiscono migliaia di cache nelle reti edge, Twitch serve direttamente milioni di utenti da, relativamente, pochi server in Nord America (NA), Europa (UE) e Asia (AS).
Vi sono molteplici fattori che influenzano la politica di reindirizzamento e scalabilità, inclusa ovviamente la popolarità del canale ma anche, e soprattutto, la configurazione di rete del client (peering). Per esempio il 50% dei client in Asia sono serviti da server in America.

I dati che verranno presi in considerazione ora sono frutto di molteplici esperimenti che, per quanto ben accurati, rimangono tali e non garantiscono la veridicità al 100%.
Gli obiettivi dell'esperimento sono quelli di trovare il numero e la locazione dei server dell'infrastruttura di Twitch, le sue politiche di reindirizzamento e scalabilità e come gli utenti vengono mappati ai server [7].

In totale si possono contare 806 proxy su 50 stati diversi (Europa (154),

Asia (372), Africa (24), Australia (4), Nord America (138) e Sud America (114)).
Inoltre è stato osservato che Twitch reindirizza spesso un client a server differenti quando si richiede lo stesso canale più volte, evidenziando così un meccanismo di bilanciamento del carico.
Sono state fatte più di un milione di richieste sui 30 canali più popolari in modo da emulare un client tipico.
I log mostrano come tutti i flussi video sono serviti da sotto domini di: ***hls.ttvnw.net***.
Ogni dominio viene associato a un airport code che accenna una posizione geografica: ***video11.fra01.hls.ttvnw.net*** indica che il server si trova a Francoforte.

A differenza di Netflix e Google, Twitch ha un sistema autonomo di hosting dove si trovano tutti i server. Purtroppo sapere dove sono i server twitch non serve a capire come distribuisce il carico di lavoro.
Un canale è popolare in base a quanti viewers ha. Ci si aspetta che più un canale sia popolare più esso venga distribuito su più server. In Twitch no, o almeno non solo.

La scala non si basa solo sul numero di spettatori. Sono stati scelti 30 Streamers tra i più popolari in diverse regioni nel mondo e sono state fatte diverse richieste, una ogni 5 min. La capacità viene regolata in modo indipendente per ogni regione, e dimostra che Twitch bilancia dinamicamente il numero di server assegnati a un canale, a seconda del numero di visualizzazioni. Inoltre, indica che ogni regione è scalata in modo indipendente in base al numero di telespettatori di tale regione. Quindi: il numero di server che ospitano il canale è correlato con il numero di spettatori che guardano il canale **per regione**.

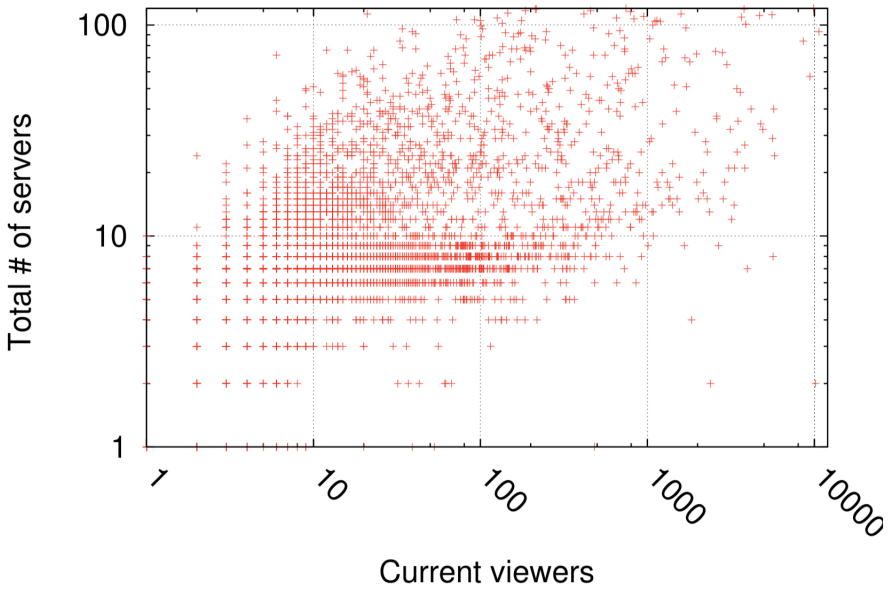

Figura 2.2: Distribuzione server

I canali con pochi viewers tendono a essere serviti solo dall'America mentre i canali con più di 50 spettatori vengono serviti da tutti e 3 i continenti: America, Europa, Asia.

Come avviene il reindirizzamento? Ovviamente uno dei fattori principale è la vicinanza dal server, ma non solo: il 99,4% delle richieste in Nord America e il 96% delle richieste in Sud America sono gestiti dal server di NA; 82% delle richieste in Europa e il 78,2% delle richieste in Africa sono serviti dai server europei.
La maggior parte di client in Asia viene gestito da Server in America. Perchè? Ispezionando il 15% dei client asiatici che si basano esclusivamente su server asiatici, vediamo che tutti condividono gli stessi servizi di peering privati con Twitch (sulla base di PeeringDB). Pertanto, è probabile che l'Asia non riesca a localizzare le sue richieste a causa di questi scarsi accordi di peering esistenti. Anche se i server in Asia sono geograficamente vicini, la loro distanza di rete potrebbe essere più alta. La topologia e il peering sono

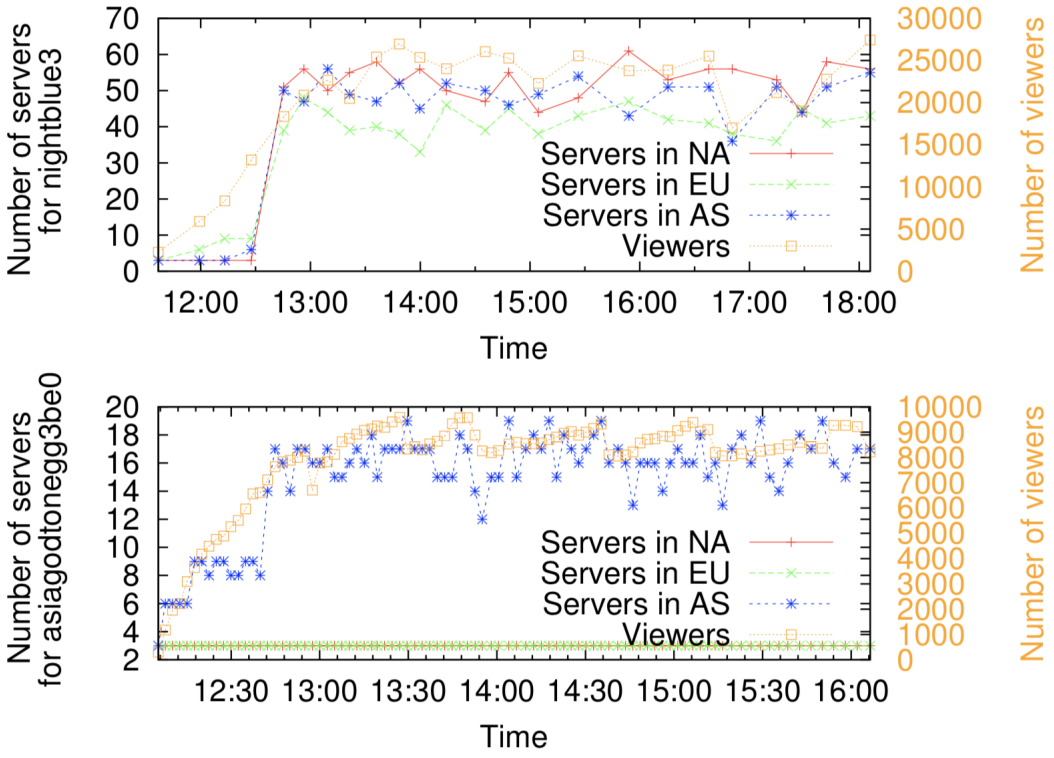

Figura 2.3: Analisi scalabilità

molto più importanti della distanza geografica.
Questa è una sostanziale differenza rispetto a YouTube, il quale adotta una strategia statica.[6]

2.2.2 Performance

Twitch è un software proprietario e in quanto tale è difficile da testare e valutarne le performance. Per questo motivo viene condotto un duplice esperimento atto a valutare alcune metriche sulle performance della piattaforma. Gli esperimenti sono stati effettuati grazie a due tool, utilizzando una rete in fibra ottica da 1Gps in download e 200mbps in uploads: TwitchTest e Twitch Inspector.

TwitchTest è un programma open source gratuito che consente di misurare facilmente la velocità di upload su ciascun server Twitch. La valutazione delle performance di ciascun server viene misurata attraverso 3 metriche:

SERVER	BANDWIDTH	RTT	QUALITY
EU: Amsterdam	3187 kbps	32 ms	90
EU: Frankfurt	2475 kbps	40 ms	85
EU: London	3676 kbps	31 ms	93
EU: Paris	2363 kbps	47 ms	91
EU: Milan	2760 kbps	13 ms	97
US Central: Dallas	3246 kbps	119 ms	71
US East: Miami	3168 kbps	103 ms	75
US West: Los Angeles	2567 kbps	143 ms	65

Figura 2.4: Twitch Test Perfomance

Bandwidth, RTT, Quality.

- **Bandwidth:** indica la larghezza di banda. Da tener presente che TwitchTest non supererà mai i 6mbps durante i test e viene consigliato di non superare i 3500kbps per lo streaming.
- **RTT:** tempo necessario per stabilire la connessione (Round Trip Time).
- **Quality:** questa è una metrica che cerca di misurare la stabilità / qualità complessiva della connessione in base alla velocità con cui i dati sono stati inviati e il numero di pacchetti ritrasmessi. È misurato su 100 e per un flusso stabile è consigliata una valutazione di 80 o superiore. Se la connessione Internet è inferiore a 10 Mbps, i numeri di qualità potrebbero essere inferiori.

Vengono testati diversi server in Europa e in America in modo da valutarne le diverse prestazioni. I risultati (vedi Fig. 2.4) mostrano come i server locati nello stesso continente abbiano un RTT inferiore ai 50ms, mentre per quelli in America sia superiore ai 100ms.

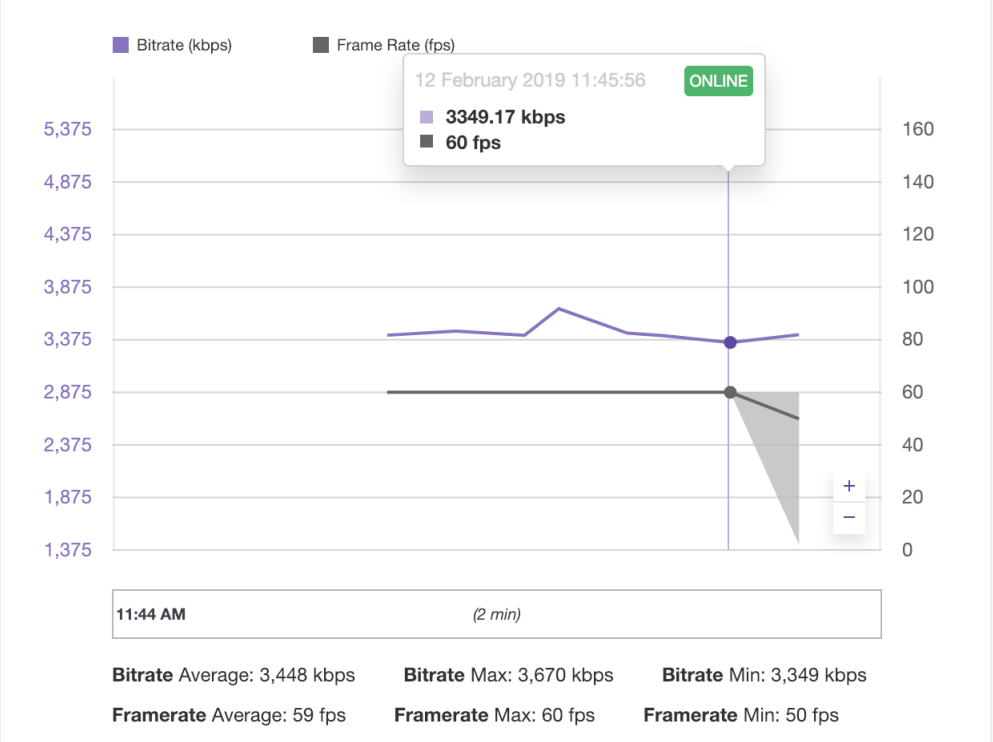

Figura 2.5: Twitch test bitrate

Twitch Inspector invece è un tool proprietario di Twitch che ti permette di verificare le performance di trasmissione su qualunque server del sistema. Viene scelto OBS come software per la trasmissione. Un broadcaster invia un flusso video a Twitch: ogni broadcast usa un URL RTMP in questo formato: *rtmp://<twitch-ingest-server>/app/<stream-key>*
Twitch specifica un bitrate massimo (bit trasferiti al secondo del video) di 3.500 kbps, ma la maggior parte dei flussi di Twitch usa meno. Mentre un bitrate più elevato può portare a video di qualità superiore, riduce il numero di potenziali spettatori, poiché alcuni computer o connessioni Internet non sono in grado di gestire un bitrate più alto. Inoltre, un bitrate più alto non sempre porta a una migliore qualità dell'immagine. Un bitrate più basso spesso significa una connessione più stabile, che si traduce in un minor numero di problemi per gli spettatori.
I test vengono fatti su 2 server in Europa (Milano e Marsiglia) e uno in America (Dallas). In tutti i test vengono settati i seguenti parametri: 60

fps, 1680x1050 risoluzione, 3500 e 6000 bitrate. I risultati sono mostrati in figura 2.5; non viene riscontrata alcuna anomalia in nessuno dei 3 casi. Come detto precedentemente Twitch consiglia di non superare 3500kbs di bitrate per la trasmissione seppur non crea problemi fino 6000kbs.

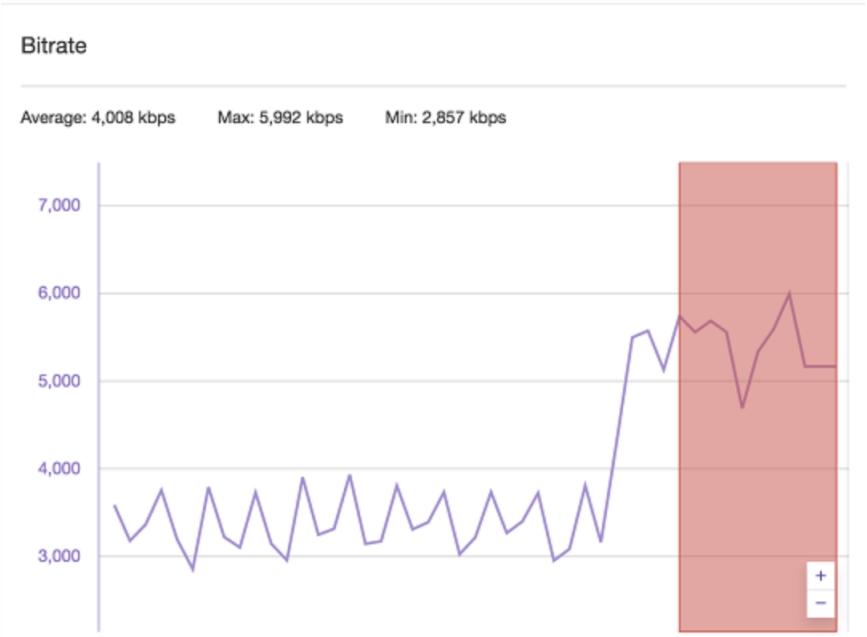

Figura 2.6: Twitch test bitrate 14k

Qui l'emittente trasmette in streaming a 14.000 kbps (vedi Fig. 2.6). Usare semplicemente un bitrate più alto non significa necessariamente una qualità migliore; in questo caso un bitrate così elevato causa instabilità. In molti casi, l'elevata larghezza di banda causa instabilità del flusso durante una trasmissione[1].

2.2.3 Modificabilità

Non ci si può esprimere con sicurezza per quanto riguarda il cloud. Si può denotare la grandezza dell'ecosistema Twitch e anche di ogni suo componente il che non rende facile e immediata l'integrazione di nuove features. A maggior ragione nelle prime versioni di Twitch il quale era organizzato in

un'unica enorme app divisa in componenti e sottocomponenti comunicanti tra loro attraverso APIs. Immaginiamo un ecosistema enorme gestito in un'unica codebase; ogni API di ogni servizio come accetterà le richieste? E cosa dovrebbe restituire l'endpoint? E come dovrebbe comunicare errori? E ci sono intestazioni particolari che dovrebbero essere impostate? Che dire del controllo delle versioni? L'elenco delle preoccupazioni può durare a lungo.
Abbiamo parlato di un singolo servizio che di certo non ha solo un'"azione". Immaginiamo tutto moltiplicato per ogni azione di ogni servizio per ogni servizio che abbiamo. E siamo solo alla creazione. Immaginiamo poi questo lavoro per ogni nuova api che dobbiamo progettare.
Per questo motivo nel 2015 gli sviluppatori hanno creato un RPC proprietario chiamato Twirp, il quale permette di concentrarsi solo sulla logica del servizio e semplifica molto la comunicazione lato backend. Viene creato un micro servizio per ogni features e ciò rende molto più facile la modificabilità del sistema. Verrà poi approfondito nel capitolo 3.3

2.2.4 Disponibilità

La capacità del sistema di gestire eventuali guasti. Per un servizio di live streaming essere sempre online e funzionante è importante. Qualora un server dovesse andare in down sarebbe impensabile che tutti i broadcaster comunicanti con esso non trasmettessero più, o se si perdessero tutti i messaggi di una chat room. Gli ingest server sono la parte più critica da gestire poiché contengono tutte le informazioni riguardanti lo stream video, compresi i messaggi.

Come esplicitato precedentemente Twitch presenta diversi server in giro per il mondo, Europa, Asia e America. Questo, oltre che essere determinante per la scalabilità, permette di tollerare i guasti sostituendo automaticamente le istanze corrotte e reindirizzando il broadcast. Il sistema crea continuamente copie ridondanti dello status del broadcast salvando le istanze dei messaggi in chat e dello stream. In caso di guasto l'istanza viene in pochissimo tempo

reindirizzata ad un altro server e lo status di quest'ultima viene ripristinato. Per la distribuzione del carico e la fault-tolerance Twitch utilizza una tecnica chiamata Round-Robin DNS: Round-robin DNS funziona rispondendo alle richieste DNS non solo con un unico potenziale indirizzo IP, ma con un elenco di potenziali indirizzi IP corrispondenti a diversi server che ospitano servizi identici. In caso di diversi tentativi di connessione, i client ricavano il servizio da diversi provider, distribuendo così il carico complessivo tra i server.

2.2.5 Usabilità

Si può accedere al portale attraverso diversi client su diversi dispositivi: console, tablet, mobile, desktop.
Sulla base dell'esperienza dell'autore del documento sono stati fatti esperimenti sulle features principali del sistema per valutarne l'usabilità: streaming video, servizio di chatting, web apis, bot.

Streaming Video

L'interfaccia di Twitch è semplice e intuitiva. Compaiono in home le live più popolari o quelle degli streamer che segui. E' un task molto semplice entrare in un canale di una diretta e interagire con gli altri utenti e lo streamer stesso.
E' possibile fare donazioni a uno streamer durante una diretta o fare un abbonamento usufruendo di diverse features, come avere a disposizione più emoticon nella chat, o privilegi nei canali di interesse.
Meno basilare è invece creare una diretta su Twitch.
Si possono usare diversi client: tra i più popolari OBS e XSplit Broadcaster. Lo streamer riceve la chiave segreta per lo streaming e il client gli chiederà le informazioni base come la risoluzione, gli fps e il bitrate.

Servizio di Chatting

In ogni canale è possibile interagire con gli altri utenti e lo streamer attraverso il servizio di chatting. E' un servizio altamente scalabile che invia miliardi di messaggi ogni giorno attraverso protocolli come IRC. Tutto ciò verrà approfondito nel capitolo 3.1. Venendo all'usabilità è un servizio performante e intuitivo. Nella chat sono presenti semplici utenti che guardano la live, utenti abbonati, bot, utenti dello staff, gli amministratori e i moderatori del canale. Ognuno di essi ha un badge, un'icona che caratterizza l'utente. Twitch offre una libreria dedicata alle emoticon utilizzabili. Gli amministratori si occupano di monitorare i messaggi e ogni messaggio che non rispetta le normative del sistema verrà cancellato con segnalazione per l'utente e possibile espulsione dalla chat room e cancellazione del profilo.
I moderatori hanno a disposizione un set di comandi e funzionalità che consentono di monitorare e moderare la chat con estrema precisione. Le opzioni variano dalle brevi sospensioni di un utente al filtro anti-spam incorporato.

Web APIs and Data

Le API Web consentono agli utenti di gestire e personalizzare i propri profili e abbonamenti, di cercare e individuare i canali streaming desiderati, gestire servizi di abbonamento e altro. Vi sono diverse versioni di APIs sulla documentazione di Twitch e viene permesso agli utenti di utilizzare qualsiasi versione. Per effettuare chiamate API, è necessario un ID client. Nella dashboard del sistema è possibile registrare un app e autenticarsi in OAuth.
Tutti i dati sono inviati e ricevuti in formato JSON e tutti gli endpoint API supportano JSONP fornendo un parametro di callback con la richiesta.
Le nuove API Twitch includono un sistema webhook che consente di iscriversi a un argomento e di essere avvisato quando arrivano nuovi dati per quell'argomento. Ciò aiuta ad alleviare la necessità di eseguire il polling dell'API per ottenere nuovi dati.

Bot

Twitch offre un'interfaccia Internet Relay Chat (IRC) per la funzionalità di chat. I chatbot ti consentono di interagire a livello di codice con la chat in un canale utilizzando gli standard IRC; il bot si collega alla rete Twitch IRC come client per eseguire determinate azioni.
Per la creazione di un bot Twitch richiede un username per il bot, il nome del canale dove eseguire il bot e il token OAuth per autenticare il chatbot.
Si implementano utilizzando node e qualsiasi programmatore con un po di dimestichezza riuscirà a crearne uno senza troppe difficoltà. La documentazione ufficiale esplica con precisione come utilizzare i molteplici endpoint contattabili.
Vi sono Bot di diverso tipo con differenti funzionalità come i Bot moderatori, o per la donazioni o per le analisi. Tra i più famosi cito *Nightbot* e *Moobot* come moderatori, *Streamtip* per le donazioni, *Muxy* sempre per ricevere donazioni e raccogliere e analizzare dati sul proprio canale.

2.2.6 Sicurezza

Per Twitch la sicurezza è un aspetto importante. "E' fondamentale garantire la sicurezza ai nostri utenti", dice Emmett Shear, founder di Twitch. Twitch garantisce sicurezza sia nell'autenticazione sia nello streaming.
Gli utenti possono interfacciarsi con le applicazioni solo dopo essersi autenticati, i dati dell'utente sono conservati in un oggetto di archiviazione locale che scade dopo alcune ore. L'autenticazione avviene tramite OAuth 2.0 ed è formata da un token di accesso e un token di aggiornamento. Il token di autenticazione contiene le informazioni sull'utente mentre quello di aggiornamento viene utilizzato quando il token di sessione scade ed è necessario ottenerne uno nuovo.

Per quanto riguarda lo streaming, Twitch fornisce a ciascun utente per ogni stream una chiave, la quale si raccomanda di tenere segreta. E' una chiave di autenticazione. E' consigliato inoltre l'utilizzo di una VPN sia se l'utente si trova in un paese dove il governo prevede delle censure, sia per una que-

stione di sicurezza e protezione di attacchi comuni a tutti gli streamer come DDOS e perdite di indirizzi IP.

Capitolo 3

Struttura

In questa sezione analaizziamo i componenti di Twitch dal punto di vista architetturale. Qual è la pipeline dei componenti core, come sono strutturati i server per il servizio di chatting e di video streaming.

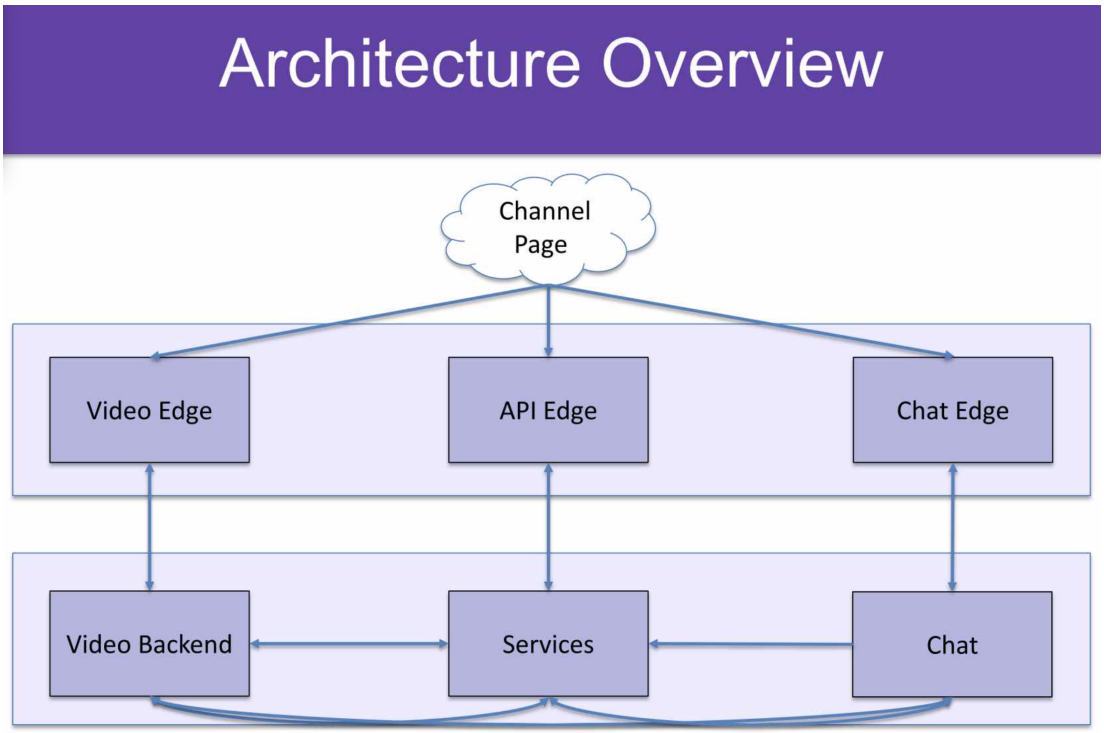

Figura 3.1: Architecture Overview

3.1 Chat Servers Organization

Twitch vanta milioni di utenti attivi che scambiano centiania di miliardi di messaggi ogni giorno. Come accennato nella sezione di introduzione il servizio di chatting è costituito da un sistema real-time distribuito in decine di server, altamente scalabile, scritto in Go. Lo scambio di messaggi degli utenti avviene tramite il protocollo IRC (Internet Relay Chat).

Data la crescita esponenziale dei messaggi scambiati sui canali Twitch l'idea migliore è stata quella di seprare i server in diversi cluster al fine di rendere il sistema più robusto (se un cluster dovesse andare in down il sistema non crolla) e performante.

- Main: contiene al massimo 10/12k di viewers in un canale
- Event: 100k di viewers degli eventi più popolari, come Twitch Plays Pokemon O League Of Legend
- Test: Serve per testare nuove porzioni di codice. 0.2% del traffico dei messaggi

Il server edge si trova al "margine" della rete pubblica e della rete interna di Twitch e ha il compito ricevere e distribuire messaggi tra client e servizi di back-end. Edge parla il protocollo IRC sia su TCP che WebSocket. Lo user crea una comunicazione attravero un *flash socket* con il server periferico il quale da in pasto il messaggio a una pipline composta da message *Ingestion* e message *Distribution* (vedi Fig. 3.2)

Un edge sever sostanzialmente è costituito da una struttura schematica di questa forma:

user:room:edge

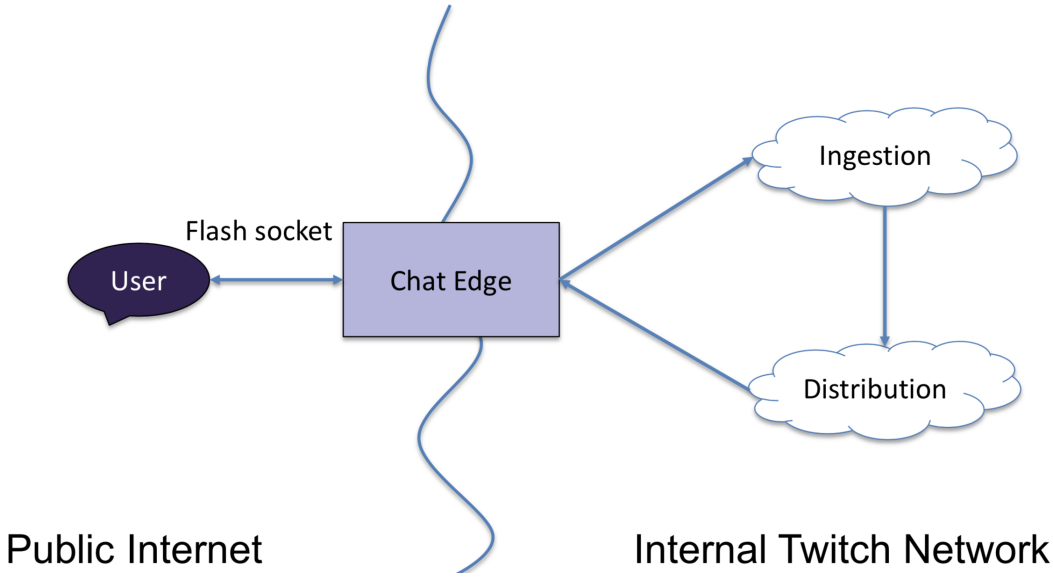

Figura 3.2: Chat Architecture

Il server deve gestire la coppia *User-Room*. Non c'è nessun clustering specifico per raggruppare le "Room". Un utente può connettersi alle chat-room attraverso uno o più edge server (vedi Fig 3.3)

Il componente più importante dell'architettura di Ingestion è *Clue*. Clue è il message business logic server e ha il compito di eseguire analisi sulle azioni degli utenti. L'utente è bannato nel canale in cui sta parlando? Sono abbonati? Sono vietati o esibiscono comportamenti abusivi?

La comunicazione tra il server edge e la pipeline Ingestion avviene attraverso chiamate HTTP, questo perchè è il modo più semplice di comunicare e anche più facile da debuggare.
Tutti i log e i messaggi elaborati da Clue vengono poi memorizzati tramite delle API su database Redis distribuiti grazie all'assegnamento di una chiave all'utente, questo sempre per evitare che il sovraccarico faccia crollare il sistema.

La decisione di come e quando reindirizzare i messaggi viene presa dal mes-

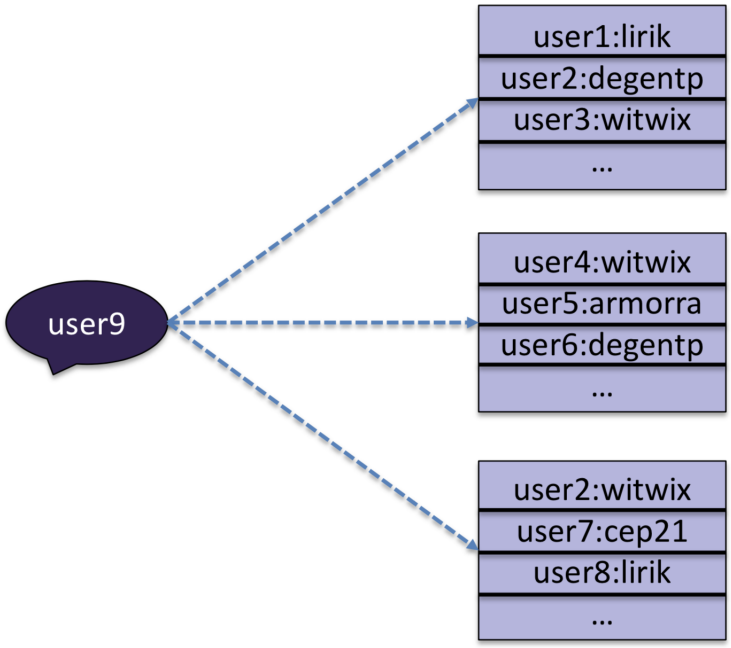

Figura 3.3: User Room

sage *Distribution.* Si occupa di indirizzare i messaggi provenienti dal servizio Clue alla chat-room giusta e fare il broadcasting a tutti gli utenti viewrs di quella chat.

3.2 Video Service

In linea generale e ad alto livello i componenti core del video service non si differenziano concettualmnte da quelli del chat service. Vi sono sempre i componenti Ingest e Distribution i quali raccolgo flussi dagli streamers e li distribuiscono ai viewers.

Entriamo più nel dettaglio. Il servizio inizia raccogliendo la richiesta di broadcasting dallo streamer e come visto nel capitolo precedente sceglie un PoP (point of presence) secondo determinate metriche al quale trasmettere il flusso video RTMP.

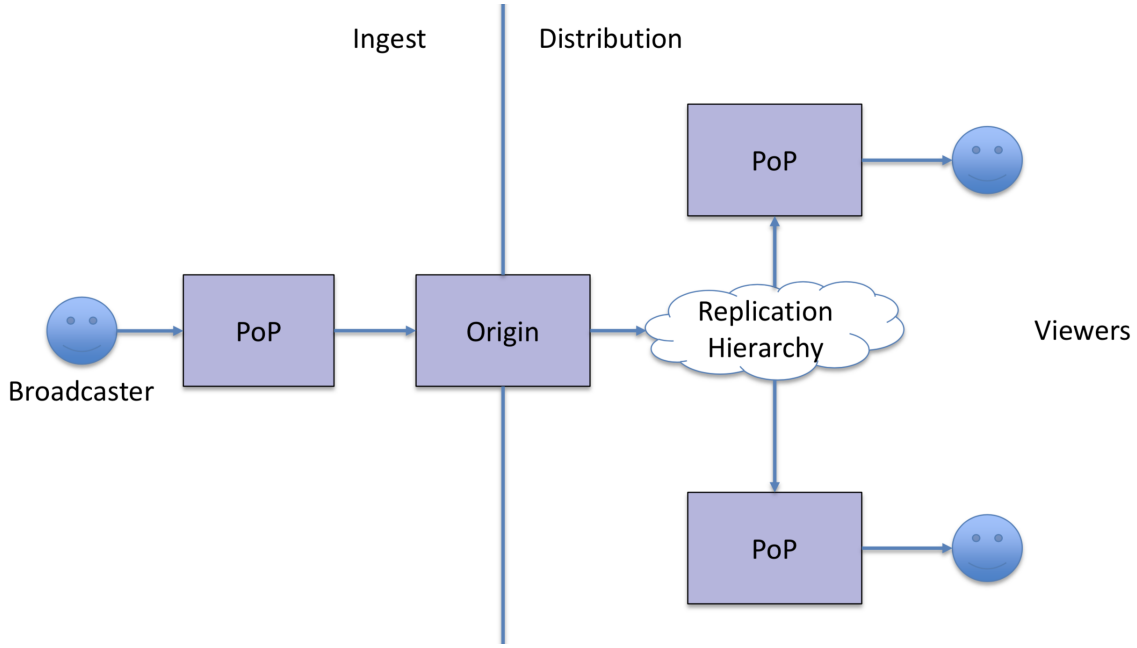

Figura 3.4: Video Architecture

RTMP è un protocollo basato su TCP che mantiene connessioni persistenti e consente comunicazioni a bassa latenza. Per distribuire i flussi in modo fluido e trasmettere quante più informazioni possibili, divide i flussi in frammenti e le loro dimensioni sono negoziate dinamicamente tra client e server. Il flusso viene poi passato al sistema di transcodifica/trasmissione e su questo punto ha senso soffermarsi e capire che strategie vengono adottate.

3.2.1 FFmpeg vs TwitchTranscoder

Come accennato precedentemente Twitch, come molti altri servizi di streaming live, riceve i caricamenti dal live stream in Real-Time Messaging Protocol dai suoi emittenti. RTMP è un protocollo progettato per lo streaming di video e audio su Internet ed è principalmente utilizzato per la comunicazione punto a punto.
Per adattare il contenuto del live streaming a innumerevoli spettatori, Twitch

utilizza **HTTP Live Streaming**, un protocollo di comunicazione di streaming multimediale basato su HTTP utilizzato anche dalla maggior parte dei siti Web di video.
Transcoder è incaricato di convertire un flusso RTMP in entrata nel formato HLS con più varianti (ad es. 1080p, 720p, ecc.). Questo per ogni tipo di utente con ogni tipo di banda.

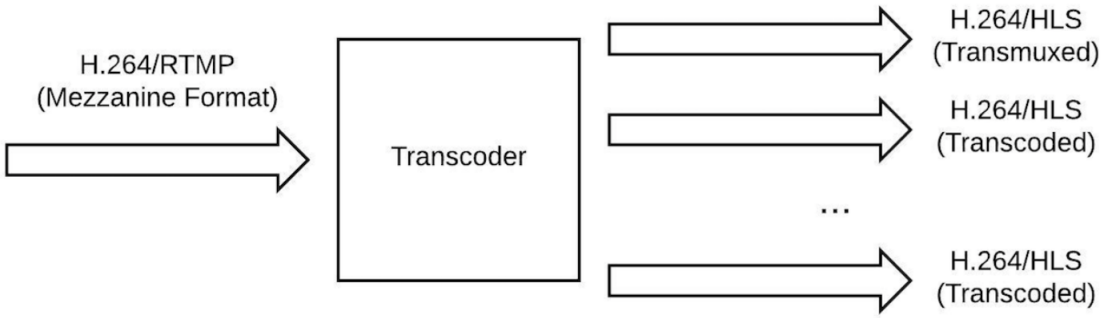

Figura 3.5: Transcoder

FFmpeg è software open source popolare progettato per registrare processare e trasmettere video e audio. Utilizzato per la trasmissione e la transcodifica di streaming live.
Supponiamo di ricevere lo standard di compressione video più utilizzato di H.264 in formato RTMP a 6mbps e 1080p60 (risoluzione di 1920 per 1080 con un frame rate di 60 frame al secondo). Si vogliono generare 4 varianti HLS di:

- 1080p60 HLS / H.264
- 720p60 HLS / H.264
- 720p30 HLS / H.264
- 480p30 HLS / H.264

Una soluzione consiste nell'eseguire 4 istanze indipendenti di FFmpeg consentendo che i segmenti HLS di uscita di tutte le varianti siano perfettamente allineati nel tempo, come richiesto dallo standard HLS.

La transcodifica abbasserà notevolmente la qualità del video.
A causa di questi due vincoli (video ad alta risoluzione e frame rate elevato), si preferisce trasmutare piuttosto che transcodificare la variante più alta dalla sorgente RTMP per salvare la potenza di calcolo e preservare la qualità del video.
La trasmissione del flusso di bit di origine è una tecnica efficace, ma potrebbe far sì che l'output HLS perda la sua conformità alle specifiche, causando la sua non riproducibilità su alcuni dispositivi.

Probliemi tecnici usando FFmpeg

La sezione precedente ha dimostrato come FFmpeg può essere utilizzato per generare HLS per i live streaming. Sebbene utili, alcuni problemi tecnici rendono FFmpeg una soluzione tutt'altro che ideale.
In HLS, una variante è composta da una serie di segmenti, ognuno dei quali inizia con un frame IDR. La specifica HLS richiede che i fotogrammi IDR dei segmenti corrispondenti di una variante siano allineati, in modo tale che abbiano lo stesso Timestamp di presentazione (PTS). Solo in questo modo un lettore HLS Adaptive Bitrate (ABR) può passare senza problemi tra le varianti quando cambia la condizione della rete dell'utente.

Se si transcodifica sia la fonte che il resto delle varianti, si otterranno segmenti HLS perfettamente allineati nel tempo poiché viene forzato FFmpeg a codificare gli IDR con precisione ogni 2 secondi. Tuttavia, non si ha il controllo sugli intervalli IDR nel flusso di bit sorgente RTMP, che è completamente determinato dalla configurazione del software di trasmissione. Se si transmuta la sorgente, i segmenti delle varianti transmeded e transcoded

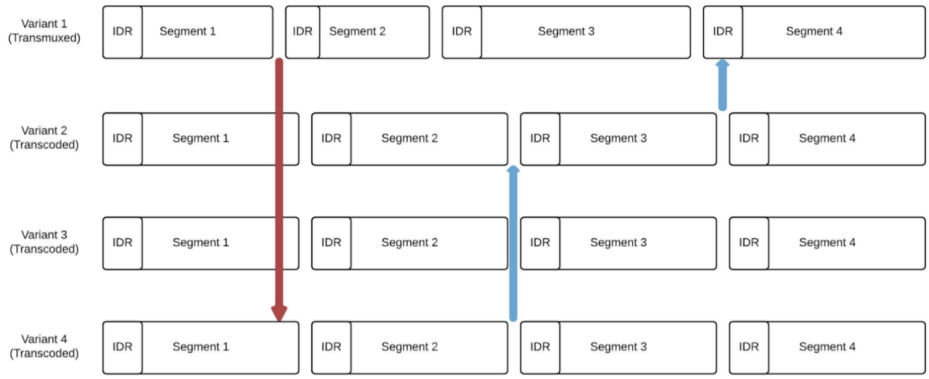

Figura 3.6: Transcoder non sincronizzato

non sono garantiti per l'allineamento (vedi Fig. 3.6). Questo disallineamento può causare problemi di riproduzione. Ad esempio, si nota che Chromecast mostra costantemente pause di riproduzione quando riceve flussi HLS con segmenti disallineati.

Per il flusso RTMP di origine con intervalli IDR variabili, idealmente si vuole che l'output HLS sia allineato come in Figura 3.7

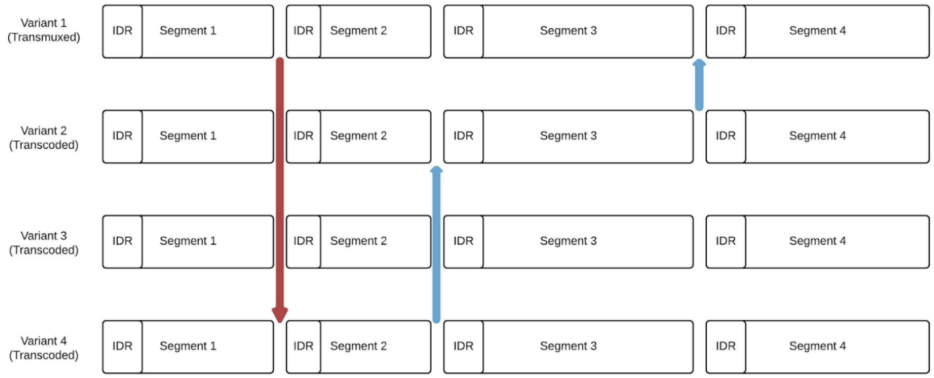

Figura 3.7: Transcoder sincronizzato

Come discusso in precedentemente, il transcoder RTMP-HLS prende un in-

put di 1 stream e produce un output di N flussi (N = il numero di varianti HLS, ad es., N = 4 in Fig. 3.5). Il modo più semplice per ottenere questo risultato è creare N transcodificatori 1-in-1-out indipendenti, ciascuno dei quali genera 1 flusso di output. La soluzione FFmpeg descritta sopra utilizza questo modello e ha istanze N FFmpeg.
Ci sono 3 componenti all'interno di un transcoder 1-in-1-out, ovvero decoder, scaler e encoder (vedi Fig. 3.7). Pertanto, per le istanze N FFmpeg, avremo complessivamente N decoder, N scaler e N encoder.
Poiché gli N decodificatori sono identici, il transcodificatore dovrebbe idealmente eliminare i decodificatori N-1 ridondanti e alimentare le immagini decodificate dall'unico decodificatore agli scaler e agli N encoder downstream.

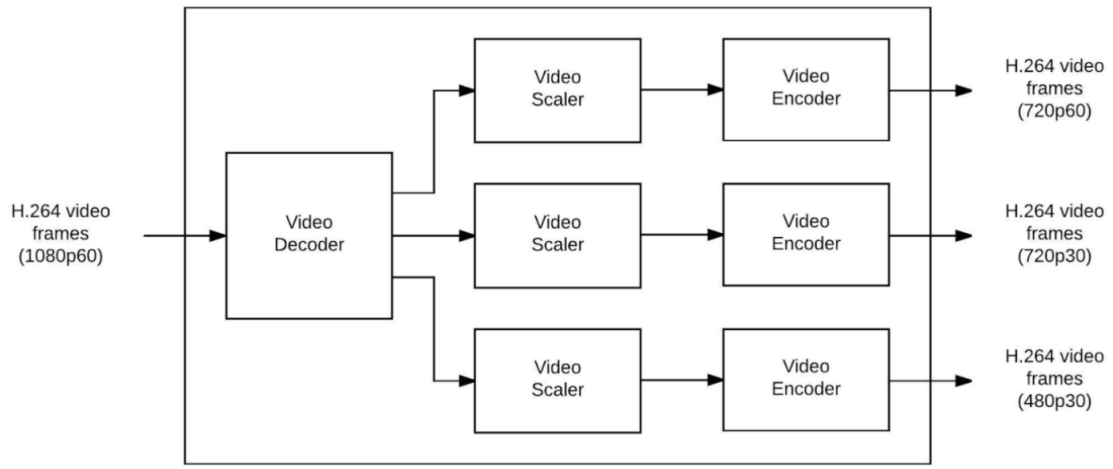

Figura 3.8: Componenti transcoder

Come rilevato dagli esperimenti, il ridimensionamento è un passo molto costoso (computazionalmente) nella pipeline di transcodifica. Evitare i processi di ridimensionamento ridondanti e non necessari può ottimizzare in modo significativo le prestazioni del transcoder.
Oltre alla condivisione di decoder e scaler, una funzione più importante è il multithreading. Poiché sia l'encoder che lo scaler sono processi molto costosi dal punto di vista computazionale, è fondamentale utilizzare la moderna

architettura CPU multi-core e consentire l'elaborazione simultanea di più varianti.

FFmpeg è un versatile software di elaborazione video che supporta vari formati video / audio per il flusso di lavoro di transcodifica ABR standard. Tuttavia, non può gestire una serie di requisiti tecnici specifici per l'operazione di Twitch. Per esempio *Frame rate downsampler* poichè dai paesi PAL che trasmettono 50fps la variante di bit rate inferiore deve essere ridotta a 25 fps invece di 30 fps e il downsampler deve comportarsi in modo diverso per i due diversi tipi di bitstream in entrata. Oppure *Inserimento di metadati* che grazie al transcoder di Twitch possono essere inglobati.[9]

Performance

TwitchTranscoder è il software interno sviluppato per risolvere i problemi tecnici discussi in precedenza. È stato utilizzato nella nostra produzione per elaborare decine di migliaia di live streaming simultanei 24/7.
Per determinare se TwitchTranscoder funzionasse meglio di FFmpeg nelle attività di transcodifica giornaliere, sono stati eseguiti una serie di test di benchmark di base. Per i test, sono stati alimentati entrambi gli strumenti con un live stream di Twitch e un file video 1080p60 utilizzando gli stessi preset, profili, bitrate e altri flag. Ogni sorgente è stata transcodificata nel nostro tipico stack di 720p60, 720p30, 480p30, 360p30 e 160p30.

I risultati delle figure 3.9 confrontano il tempo di esecuzione di TwitchTranscoder vs FFmpeg. Mostrano che il nostro transcoder è effettivamente più veloce per la transcodifica.
FFmpeg è leggermente più veloce per l'output a singola variante 720p60 perché TwitchTranscoder gestisce più attività. Quando il numero di varianti aumenta, il modello multithreading di TwitchTranscoder ha un vantaggio maggiore che lo aiuta a superare FFmpeg. Per la scaletta ABR completa di Twitch, TwitchTranscoder risparmia il 65% del tempo di esecuzione rispetto

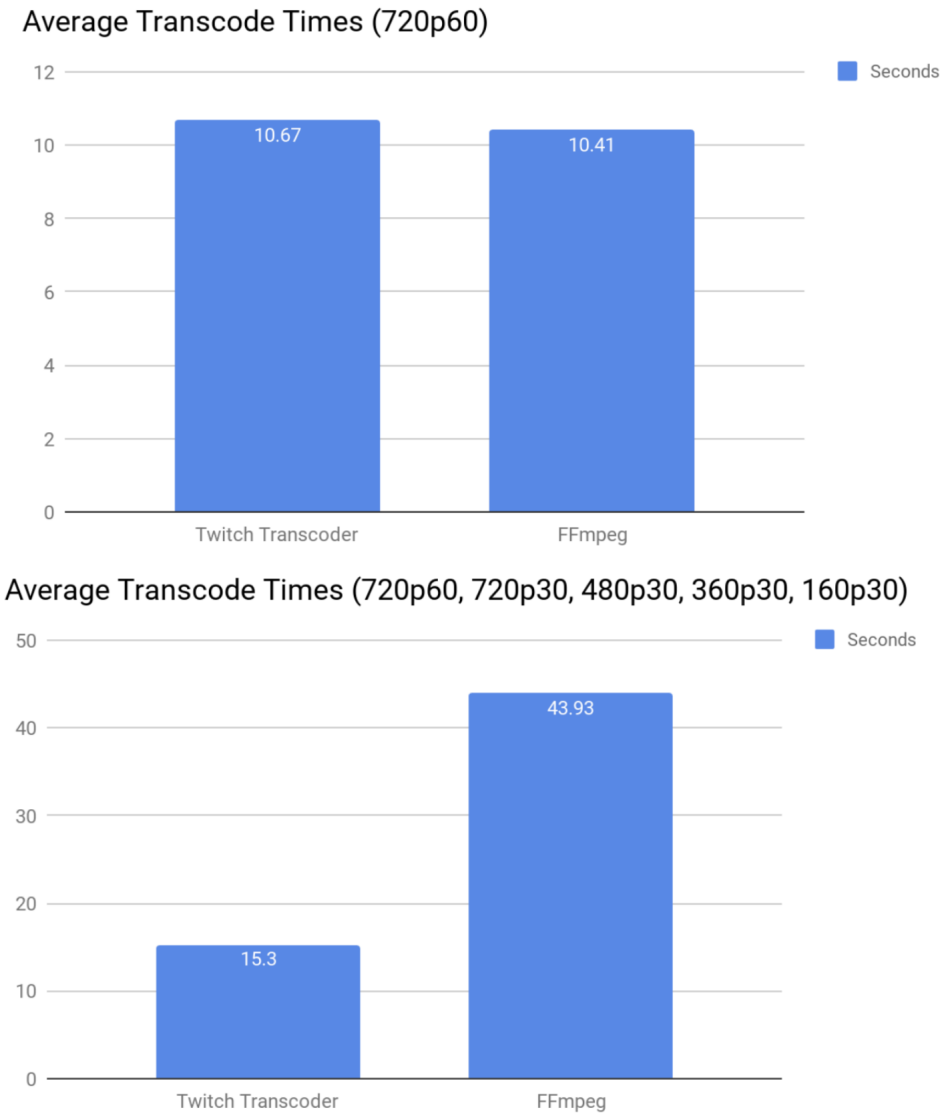

Figura 3.9: Risultati test Twitch Transcoder

a FFmpeg.

3.3 Twirp

Oggi Twitch sta rilasciando un framework RPC che viene usata per la comunicazione tra i server di back-end scritti in Go. Si chiama Twirp, ed è

Figura 3.10: Twirp

ora disponibile sotto una licenza open source Apache 2.
Twirp ha avuto un enorme successo presso Twitch, è cresciuto in termini di utilizzo esponenziale, all'incirca triplicato ogni tre mesi, mentre sempre più team interni continuano ad adottarlo per via dei suoi vantaggi rispetto alle API "REST" o gRPC.

3.3.1 Twirp vs APIs

Gli RPC strutturati sono molto più facili da progettare e gestire rispetto alle API REST orientate agli URL, in quanto consentono di concentrarsi sulla logica aziendale anziché sugli schemi di routing. Cambiare le API per aggiungere nuovi campi o metodi è molto più semplice e le peculiarità della serializzazione (come, ad esempio, la mancanza di JSON dei numeri a 64 bit) possono essere nascoste.

Nel 2015 Twitch ha fatto una grande spinta per uscire da una singola enorme app e tirare fuori la logica in servizi separati. Da una codebase monolitica a microservizi. Quando lo fai, devi quasi subito iniziare a progettare le API per questi nuovi servizi. In che modo gli altri componenti di Twitch entre-

ranno in contatto con il nuovo sottocomponente?
Immaginiamo un mini servizio molto semplice che aggiorna solo l'indirizzo email dell'utente. Come accetterà le richieste? Qualcuna di queste API HTTP sembra plausibile:

- POST */users/:id/email*, body è il nuovo indirizzo email da utilizzare
- PUT */users/:id/email*, body è il nuovo indirizzo email da utilizzare
- POST o PUT */users/:username*, body è Json-econded come "email": <nuovo valore>
- PATCH */users/:username/info*, body è una struttura JSON che descrive le modifiche (per i veri puristi di REST)

E cosa dovrebbe restituire l'endpoint? E come dovrebbe comunicare errori? E ci sono intestazioni particolari che dovrebbero essere impostate? Che dire del controllo delle versioni? L'elenco delle preoccupazioni può durare a lungo.

Abbiamo parlato di un singolo servizio che di certo non ha solo un'"azione". Immaginiamo tutto moltiplicato per ogni azione di ogni servizio per ogni servizio che abbiamo. E siamo solo alla creazione. Immaginiamo poi questo lavoro per ogni nuova api che dobbiamo progettare. Diventa infinita la cosa.

- Si avranno schemi URL diversi per servizi diversi, il che rende i log più difficili da comprendere e metriche molto difficili da organizzare in modo chiaro.
- Ogni client dovrà essere scritto a mano e si avrà bisogno della documentazione scritta a mano di ogni API, che dovrà essere aggiornata per aiutare gli altri a scrivere quei client.
- Quando arriva il momento di aggiornare il servizio per supportare nuovi prodotti - che saranno prima di quanto pensiate - si dovranno apportare

modifiche, che saranno difficili da propagare ai client, assumendo che si abbia persino un elenco completo dei vostri client.

3.3.2 Twirp

Twirp risolve il problema di comunicazione del backend attraverso la *code generation*.

Viene scritto un piccolo documento protobuf (protocol buffer) che descrive l'API. Si passa attraverso il generatore di codice Twirp e si emette un file Go contenente il codice per un router HTTP e un client HTTP corrispondente.

Quando Twirp genera il codice Go da questo file protobuf, genererà un'interfaccia che descrive il servizio.
Sul lato client, tutto quello che devi sapere è l'host e la porta del servizio, richiamando poi il metodo che si vuole invocare.

Come funzionano client e server generati?
È tutto semplicemente HTTP 1.1. Ogni richiesta è una POST e gli URL sono stringhe statiche che identificano il metodo giusto nel servizio giusto. Il corpo è o protobuf codificato in JSON o in binario, e così è la risposta. Un'intestazione identifica la codifica del corpo. E questo è tutto.

3.3.3 Twirp vs gRPC

Questo approccio alla generazione del codice non è affatto un'idea originale. Google fornisce un framework, **gRPC**, che fa una cosa molto simile, e gRPC è diventato piuttosto prominente. In realtà Twitch inizialmente utilizzava gRPC. Tuttavia, non ha ottenuto molta trazione, ma gli sviluppatori hanno avuto diversi problemi i quali li hanno portati alla creazione di Twirp. Ci sono stati quattro problemi principali in gRPC che Twirp ha risolto:

1. **Mancanza di supporto HTTP 1.1:** gRPC supporta solo http / 2 Twirp supporta HTTP 1.1 e http / 2. Questo è importante perché molti load balancer (sia hardware che software) supportano solo HTTP 1.1, inclusi gli ELB di AWS

2. **Ampio tempo di esecuzione con cambi di rottura:** gRPC è molto complesso, e il codice generato da Go è relativamente sottile e chiama in runtime grpc-go. Il codice generato e la libreria di runtime sono strettamente collegati e devono corrispondere. Sfortunatamente, quel runtime ha visto cambiamenti improvvisi, a volte senza avvertimenti o spiegazioni. Ciò sarebbe semplicemente fastidioso, ma diventa un vero e proprio dolore quando si dispone di una vasta rete di servizi che comunicano tra loro, importando i reciproci client.
   Questo significa che i client devono utilizzare la stessa versione di runtime gRPC come quella dei servizi da cui dipendono

3. **Bug a causa del runtime complesso:** grpc-go include un'implementazione http / 2 completa, indipendente dalla libreria standard, e introduce i propri meccanismi di controllo del flusso. Questa cosa è molto difficile da capire rapidamente e può portare a comportamenti confusi, controintuitivi e persino totalmente interrotti (l'ultimo bug ha causato diverse interruzioni a Twitch).
   Twirp, al contrario, può usare semplicemente il vecchio HTTP 1.1, che potrebbe non essere incredibilmente efficiente, ma almeno è semplice e sappiamo come lavorarci, e l'implementazione HTTP 1.1 della libreria standard è solida. E se hai bisogno degli extra boost di http / 2, anche Twirp può usarlo - **non elimina l'efficienza, ma solo la complessità.**

4. **Difficoltà a lavorare con il protobuf binario:** gRPC supporta solo payload binari protobuf. Twirp supporta la codifica binaria, ma supporta anche la codifica JSON di protobuf per i carichi utili, utilizzando le specifiche di mappatura JSON ufficiali.

Consentire JSON presenta due vantaggi: primo, rende più semplice la scrittura di client cross-language e di terze parti dei server Twirp
Secondo semplifica la scrittura rapida delle richieste CURL da riga di comando per eseguire il debug di un server in esecuzione.

Il vantaggio di gRPC è che supporta gli RPC di streaming bidirezionali, inviando flussi di dati ininterrotti avanti e indietro tra client e server. Twirp non ha nulla di simile, solo una semplice richiesta e risposta.

Capitolo 4

Aspetti analitici

In questo capitolo si presenta un *Utility Tree* per Twitch. Nei livelli più a sinistra si trovano alcuni dei requisti non funzionali analizzati nel capitolo 2.

4.1 Utility Tree

- **Disponibilità:**
  - Errori Hardware: considerato che Twitch utilizza diversi datacenter, è importante sfruttare questo vantaggio ed in caso di mancata connettività dell'ISP o mancata corrente, è opportuno inoltrare il traffico verso un altro datacenter. Questo non è mai facile. Twitch fornisce un servizio che salva le informazioni di ogni account, compresi gli stream trasmessi e lo status degli stream live. I dati persistenti devono essere sempre disponibili, anche in caso di fallimento di connessione al database
  - Errori Software: l'organizzazione in microservices può favorire il soddisfacimento di questo scenario. Tuttavia ciò dipende anche dalle tecnologie utilizzate; garantire un downtime vicino allo zero in caso di errori software richiede ricerca e sviluppo.

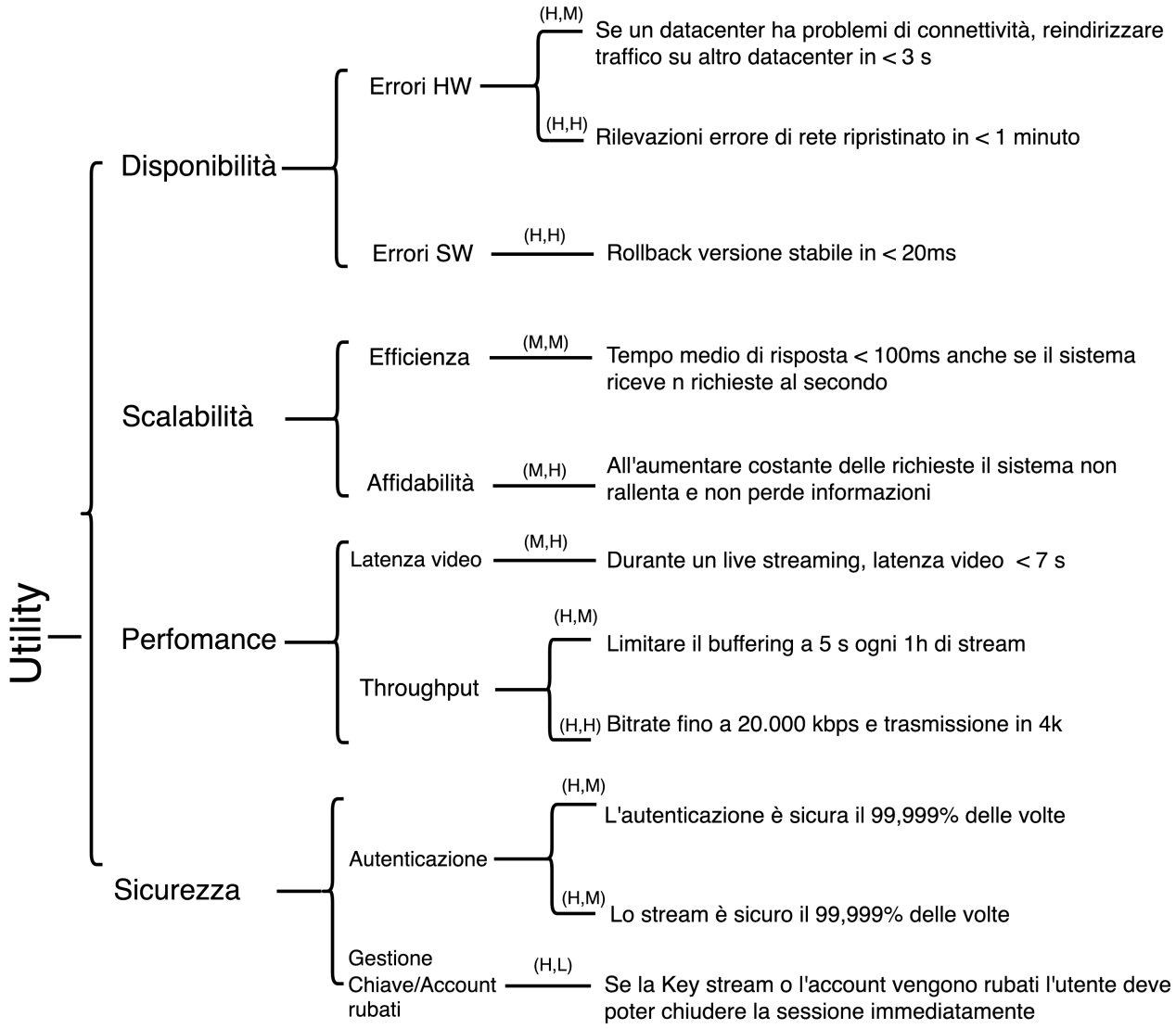

Figura 4.1: Utility Tree

- **Scalabilità:**
  - Efficienza: Twitch serve direttamente milioni di utenti da relativamente pochi server in Nord America, Europa e Asia. Twicth presenta molteplici fattori che influenzano la politica di reindirizzamento e scalabilità, per garantire sempre efficienza nei tempi di risposta. Questi fattori non si basano solo sulla vicinanza o sul numero di spettatori sui canali, ma anche e soprattutto sulla

configurazione di rete del client (peering). [6]

  - Affidabilità: all'aumentare del numero di richieste Twitch non perde informazioni nè rallenta. Secondo le metriche spiegate nel capitolo 2 Twitch riesce a scalare in maniera autonoma. Inoltre oltre i 50 spettatori in un canale Twitch serve da tutti e 3 i continenti, ovunque sia la posizione del boradcaster: America, Europa, Asia.

- **Performance:**

  - Latenza: i Test condotti nel capitoli precedenti dimostrano che Twitch è capace in buone condizioni di garantire le metriche considerate. Non è facile mantenere le latenze basse soprattutto quando il client ha una scarsa disponibilità di banda.
  - Throughput: come descritto nel capitolo 2 gli streamer di gioco hanno bisogno di una larghezza di banda fino, al massimo, a 6000 kbps per trasmettere un video full HD 1980px x 1080px con 60 fps. Grazie anche alla modalità *Low Latency* Twitch limita di molto il buffering durante lo streaming live. Ciò però genera un trade-off con la qualità di trasmissione. Per garantire un quasi annullamento della bufferizzazione si perde molto in qualità di trasmissione, mantenendo dei bitrate relativamente bassi.[11]

- **Sicurezza:**

  - Autenticazione: il fatto che Twitch non sia un software open source e quindi anche le relative Api, sicuramente rende il sistema più sicuro rispetto ad altri. Il fatto poi che tutti i componenti siano eseguiti sui servizi di Amazon Web Service ciò garantisce un ottimo grado di affidabilità.
  - Gestione chiave/account rubati: se un account o una chiave privata di streaming vengono compromessi o rubati, la sessione deve essere immediatamente chiusa. Questo scenario è facile da soddisfare: Twitch implementa la funzionalità e può fare affidamento al

suo sistema di accoppiamento account / chiave per terminare una sessione.

Capitolo 5

Twitch vs Youtube Live: implicazioni di caratterizzazione e prestazioni

Twitch ha diversi competitor, come play me, azubu ecc., ma il principale e più simile è Youtube Live. Youtube non ha di certo bisogno di presentazioni, è la piattaforma leader per quanto riguarda Video on Demand e acquistata da Google nel 2006 per 1,65 miliardi di dollari. Nel 2011 è nato un nuovo progetto di Youtube atto a permettere agli utenti di generare flussi di video in real time. Esattamente come Twitch.
Si analizzano in questo capitolo le principali differenze tra Twitch e Youtube Live in termini di features, usabilità e performance anche se risulta molto difficile testare la latenza di trasmissione di Youtube, tanto quanto lo è stato per Twitch, in quanto non sono software open source e non si conosce molto sull'architettura di Youtube Live [3].

5.0.1 Features e Usabilità

In questa sezione si mettono a confronto Twitch e Youtube Live dal punto di vista delle features offerte e dell'usabilità. I due servizi sono leader per quanto riguarda lo streaming video live e offrono un set di feature molto simile, ma esistono alcune differenze importanti.

Youtube, esattamente come Twitch, viene supportato ed eseguito su diversi dispositivi: Web App, mobile, console e smart TV. Per entrambi i sistemi è

possibile trasmettere flussi in diretta sia da smartphone, tablet console o da browser.
Youtube ha sicuramente un bacino di utenza maggiore rispetto a Twitch in quanto nasce molto prima ma con un core business diverso. Twitch rimane leader come piattaforma di live streaming video con più di 15 milioni di utenti attivi al giorno, 250 mila utenti che guardando simultaneamente e circa 3 milioni di broadcaster al mese. D'altro canto secondo la maggior parte degli utenti, e anche secondo il mio punto di vista, Twitch risulta essere meno intuitivo dal punto di vista dell'usabilità e più difficile da utilizzare. Youtube Live presenta tutte le principali features, sia di trasmissione video, sia di analisi in un'unica pagina, mentre su Twitch la curva di apprendimento è più ampia. [5]
E' anche vero però che gli utenti su Twitch riscontrano un guadagno maggiore. Confrontiamo alcuni dati:
Twitch

- **Abbonamenti:** ogni abbonato porta in $ 5 al mese. Twitch prende il 50%, quindi lo streamer ottiene $ 2,50 per iscritto. Questo è solo il contratto standard e può variare per gli streamer più grandi. I grandi streamer sulla piattaforma possono avere migliaia, persino decine di migliaia di abbonati.

- **Donazioni:** fino al donatore per quanto pagano, ma generalmente intorno a 1− 10 per la maggior parte delle donazioni. Il 100% del denaro della donazione va alla streamer. I grandi streamer possono guadagnare oltre $ 1.000 al giorno dalle donazioni.

- **Bits:** Bits sono il sistema di donazione integrato di Twitch. Generalmente sono meno utilizzati e pagano in quantità minori. Twitch subisce un taglio del 29%.

- **Annunci:** funzionano allo stesso modo di YouTube, visualizzati all'inizio dello streaming. Questi non finiscono per pagare tanto ma rappresentano ancora una buona fetta [4].

Youtube Live

- **Super chat:** questo è essenzialmente il sistema di donazione integrato di YouTube. YouTube prende il 30%, rispetto allo 0% di Twitch per la maggior parte delle donazioni.
- **Membri:** i membri di YouTube sono gli abbonati a Twitch. YouTube prende di nuovo il 30%, che è inferiore a Twitch, ma il sistema dei membri non viene utilizzato da nessuna parte vicino tanto quanto gli abbonamenti Twitch.
- **Annunci pubblicitari:** per alcune persone, questi sono peggio su YouTube di quanto non lo siano su Twitch. Gli annunci sono anche inesistenti per i canali che YouTube ha ritenuto "demonetizzati".

Nonostante questi numeri viene indicata Youtube come piattaforma per acquisire notorietà più rapidamente. Su Twitch, gli spettatori trovano i canali principalmente cercando i giochi che vogliono vedere. Se stai giocando a quel gioco, potrebbero vederti. Quando vanno a vedere un flusso di un particolare gioco, Twitch visualizza i canali in ordine discendente del numero di spettatori che guardano quel flusso. Ciò crea un pregiudizio che favorisce pesantemente gli streamer stabiliti, poiché è probabile che le persone scelgano un flusso ben popolato prima di vedere anche quelli meno popolati. L'algoritmo di YouTube può funzionare a favore dello streamer suggerendo il suo stream ad altre persone.
Twitch è in gran parte focalizzato sui giochi. Hanno recentemente aperto la loro sezione "IRL", che è diventata popolare nell'ultimo anno, ma per la maggior parte sono ancora un servizio di streaming di giochi. YouTube è molto più flessibile, con contenuti praticamente su qualsiasi cosa tu possa pensare di vivere in streaming.
Un altro punto a favore di Twitch è la miglior usabilità della chat. Youtube dedica meno importanza al servizio di chatting: presenta molte più restrizioni e meno funzionalità. Nella chat di Twitch esistono diversi ruoli, contornati da tanti badge diversi i quali vengono molto apprezzati dagli utenti. Su Youtube c'è la figura del moderatore, dell'admin ovvero lo streamer e basta. Le limitazioni stanno nello scarso filtraggio di messaggi di spam, nel

limite di caratteri imposto agli utenti per messaggio e nel delay tra l'invio di un messaggio e l'altro. Youtube impone un massimo di 3 messaggi in 30 secondi. I bot sono molto meno, vengo usati solo alcuni tra i "moderatori" più famosi, i quali sono ovviamente implementati anche per Twitch.
Una features, a mio avviso molto importante, che ha YouTube e non Twitch e la funzionalità **DVR** (pause, play, rewind). Durante una diretta è possibile mettere in pausa, ricominciare a guardarla e tornare indietro nella Live. Inoltre se si è perso l'inizio di una live è possibile tornare indietro fino all'inizio.
Nella tabella seguente si riassumono le principali differenze in termini di features tra i due servizi [2].

Features	Twitch	Youtube Live
Client	Mobile, Tablet, Web, Console, Chromecast	Mobile, Tablet, Web, Console, Chromecast
Revenue	V	X
Usability	X	V
Chat Service	V, No limits	X, 3 msg in 30s
Chat Bot	Moderatori, Donatori, Analisti	Moderatori
Content	Game	Game, IRL etc.
DVR (pause, play, rewind)	X	V
Apis	V	X

Figura 5.1:

5.0.2 Performance

Come già accennato precedentemente risulta particolarmente difficile testare la latenza di trasmissione di Youtube, dato che i client non sono OpenSource e non possono essere modificati per calcolare metriche precise. Tutta via ho

condotto un esperimento similare a quello svolto su Twitch. Viene sempre sfruttata una rete in fibra ottica a 1Gbps in download e 200 Mbps in upload. I risultati mostrano come Twitch presentava alcuni problemi di buffering, Youtube funzioni molto bene anche come bitrate notevolmente maggiori.

Qualità
Secondo le linee guida di Twitch Broadcasting gli streamer di gioco hanno bisogno di una larghezza di banda fino a 6000 kbps per trasmettere un video full HD 1980px x 1080 px con 60 fps, come abbiamo sottolineato nel capitolo 2.
una risoluzione di 4k (3840 x 2160 px) con una velocità di 60 fps e un bitrate massimo di 51000 kbps (!). Ciò significa maggior qualità e minor latenza rispetto a Twitch. Dal lato degli spettatori, questo si traduce al massimo con la stessa larghezza di banda necessaria per guardare il flusso di Twitch. L'effettiva larghezza di banda necessaria dipende dal fatto che il flusso possa essere guardato in più qualità e bitrate - un'opzione sempre disponibile su YouTube, e solo a volte disponibile su Twitch. Con una connessione mediocre pare chiaro che sia consigliato generare flussi in live su Youtube e non su Twitch.

Latenza
Twitch ha anche recentemente aperto una versione beta della loro modalità video a bassa latenza. E' testato che gli streamer vedono le latenze a partire da 1 secondo rispetto ai 7-8 secondi precedenti. Ma anche se si parla di latenza Youtube è avanti rispetto a Twitch, poiché offre le opzioni di latenza normale (per: buffering di riproduzione ridotto per gli spettatori), bassa (Ideale per: interattività quasi in tempo reale.) e ultra bassa (Ideale per: live streaming altamente interattivi con coinvolgimento in tempo reale) per gli streamer. Anche con una connessione buona Twitch buffera mentre YouTube no. Gli utenti hanno riscontrato inaspettatamente problemi di buffering con low latency sia su Youtube sia su Twitch: questo perché mentre la bassa latenza riduce il buffer read-ahed per i client degli utenti che stanno visualizzando aumenti i rischi di problemi di rete ai client dei broa-

dcaster, i quali diventano inclini alla riproduzione del buffering [12].
In tabella sono riassunte le differenze in performance tra Youtube e Twitch[8]

Features	Twicth	Youtube
Qualità	Max: • 1920x1080px • 60 fps • 6000 kbps	Max: • 3840x2160px • 60 fps • 51000 kbps
Latenza	7-8 s	3-4 s
Low Latency	Low Latency	Normal Latency, Low Latency, Ultra Low Latency
Buffering	Si	No

Figura 5.2:

Bibliografia

[1] Twitch dashboard analaysis: https://www.twitch.tv/dashboard/channel-analytics. 2019.

[2] Youtube live analytics: https://www.youtube.com/analytics. 2019.

[3] B. C. B. Churchill and W. Xu. The modem nation: A first study on twitch.tv social structure and player/game relationships. In *2016 IEEE International Conferences on Big Data and Cloud Computing (BDCloud), Social Computing and Networking (Social-Com), Sustainable Computing and Communications (SustainCom) (BDCloud-SocialCom-SustainCom)*, pages 223–228, Oct 2016.

[4] M. Claypool, D. Farrington, and N. Muesch. Measurement-based analysis of the video characteristics of twitch.tv. In *2016 IEEE Games Entertainment Media Conference (GEM)*, pages 1–4, Oct 2015.

[5] J. Deng, F. Cuadrado, G. Tyson, and S. Uhlig. Behind the game: Exploring the twitch streaming platform. In *2016 International Workshop on Network and Systems Support for Games (NetGames)*, pages 1–6, Dec 2015.

[6] Jie Deng, Gareth Tyson, Félix Cuadrado, and Steve Uhlig. Internet scale user-generated live video streaming: The twitch case. In *PAM*, 2017.

[7] H. Pang, Z. Wang, C. Yan, Q. Ding, K. Yi, J. Liu, and L. Sun. Content harvest network: Optimizing first mile for crowdsourced live streaming.

IEEE Transactions on Circuits and Systems for Video Technology, pages 1–1, 2018.

[8] Karine Pires and Gwendal Simon. Youtube live and twitch: a tour of user-generated live streaming systems. In *MMSys*, 2016.

[9] Roman Poyane. Toxic communication during streams on twitch.tv. the case of dota 2. pages 262–265, 10 2018.

[10] X. Wang, Y. Tian, R. Lan, W. Yang, and X. Zhang. Beyond the watching: Understanding viewer interactions in crowdsourced live video broadcasting services. *IEEE Transactions on Circuits and Systems for Video Technology*, pages 1–1, 2018.

[11] Cong Zhang and Jiangchuan Liu. On crowdsourced interactive live streaming: a twitch. tv-based measurement study. In *Proceedings of the 25th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 55–60. ACM, 2015.

[12] D. Y. Zhang, J. Badilla, H. Tong, and D. Wang. An end-to-end scalable copyright detection system for online video sharing platforms. In *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 626–629, Aug 2018.