Valutazione sperimentale di algoritmi di adattamento per video streaming DASH

di

Luca Attanasio

Dipartimento di Ingegneria dell'informazione (DEI)

Tesi triennale in Ingegneria dell'informazione

16 Luglio 2018

Autore.....

Luca Attanasio
Dipartimento di Ingegneria dell'informazione (DEI)
Anno Accademico 2017-2018

Relatore.....

Andrea Zanella
Dipartimento di Ingegneria dell'informazione (DEI)
Professore Associato

Indice

Capitolo 1

Introduzione

Lo streaming di video attraverso internet è una tecnologia sviluppata nell'ultimo ventennio aggiungendosi a canali tradizionali di trasmissione come televisione e radio. Lo streaming permette di trasmettere video e audio che vengono visionati dall'utente destinatario attraverso un riproduttore di contenuti multimediali (*player*) sul proprio dispositivo. Tre fattori principali permettono lo streaming: un server che memorizza i contenuti, una linea di trasmissione che connette il destinatario con il server, un algoritmo che decide a quale qualità scaricare il video.

Gli algoritmi che implementano il protocollo *Dynamic Adaptive Streaming over HTTP (DASH)* memorizzano i video sul server dividendoli in segmenti (*chunk*) per facilitarne l'invio attraverso una linea di trasmissione variabile. Ogni chunk è codificato a bitrate diversi; un bitrate più alto implica qualità maggiore e quindi anche una dimensione maggiore. I chunk che verranno scaricati dal client sono sequenziali.

Un esempio di modello della rete è composto da un server che si collega tramite ethernet al router e poi tramite fibra ottica ad internet. L'utente, invece, utilizzando un computer o un cellulare può scegliere quale tecnologia utilizzare per collegarsi ad internet come fibra ottica o canali radio 3G o 4G ma non ha alcun controllo sulla capacità di scaricamento. Oltre che sulla linea di trasmissione, si può agire sull'algoritmo che decide come scaricare il video durante lo streaming.

La richiesta di sviluppare algoritmi efficienti per la riproduzione di video durante uno streaming emerge dal fatto che gli utenti abbandonano la sessione di riproduzio-

ne del video se la qualità non è sufficiente [1]. L'utilizzo di DASH per lo streaming adattativo dinamico attraverso *HyperText Transfer Protocol (HTTP)* è diventato uno standard per la sua ottimalità. Inoltre, HTTP è un protocollo che permette la trasmissione dei dati attraverso una struttura semplice. La richiesta di contenuti video attraverso il web è la sorgente dominante di traffico su Internet [2]. I più semplici algoritmi *DASH* sono basati sulla velocità di scaricamento della rete *(bitrate)* e sullo spazio di memorizzazione dell'utente destinatario (*buffer*) e usano regole prefissate, basate su modelli non accurati della realtà. Recentemente, sono stati sviluppati due algoritmi più interessanti, *Pensieve* [3] e *D-DASH* [4] che utilizzano numerose metriche temporali precedenti come bitrate, buffer, qualità dei segmenti precedenti. Lo scopo di questi ultimi è ottimizzare la qualità dell'esperienza *Quality of Experience (QoE)* dell'utente che è la percezione di soddisfazione nella riproduzione del video. Per basarsi sulle metriche temporali osservate precedentemente, questi due algoritmi utilizzano tecniche di apprendimento per rinforzo *(Reinforcement Learning)* e apprendimento profondo (*Deep Learning*). Il Reinforcement Learning consente di imparare dall'ambiente applicativo come effettuare le scelte. Il Deep Learning invece consiste nell'utilizzo di reti neurali per ricavare un modello accurato dell'ambiente.

Queste tecniche avanzate consentono a tali algoritmi di effettuare le migliori scelte possibili per selezionare la qualità nella riproduzione di video durante lo streaming. Questa tesi si pone come obiettivo la valutazione delle prestazioni di alcuni algoritmi che utilizzano il protocollo DASH per trasmettere video attraverso la rete in condizioni reali. Per verificare la qualità dell'esperienza dell'utente finale, sono state effettuate varie simulazioni in due condizioni di canale diverse: mobilità e hostpot pubblico. In mobilità, l'utente può ricevere e trasmettere informazioni attraverso le reti 3G o 4G. Per l'accesso ad un hotspot pubblico, è stata scelta una rete wireless di una compagnia telefonica italiana. Queste condizioni molto variabili in termini di velocità di scaricamento evidenziano il motivo per cui è importante che l'algoritmo per scaricare il video sia efficiente.

Per produrre i risultati sulle prestazioni degli algoritmi in condizioni reali è stato impostato un server casalingo reso accessibile ad internet. Il server realizzato dispone

di contenuti video suddivisi in segmenti a varie qualità. Il player dell'utente connesso secondo le modalità sopra elencate scarica ciascuno dei segmenti e per ciascun segmento il server o il client, memorizza i dati relativi che comprendono: bitrate, dimensione del buffer, qualità e altre caratteristiche analizzate nel Capitolo 4. Per i dati raccolti sono stati prodotti dei grafici che permettono l'analisi delle prestazioni di ciascun algoritmo scelto.

I capitoli della tesi sono stati organizzati nel modo seguente: nel secondo capitolo, è spiegato il funzionamento dello streaming adattativo attraverso DASH. In aggiunta, sono introdotte le tecniche di apprendimento per rinforzo e apprendimento profondo sfruttate da *Pensieve* e *D-DASH*. Nel terzo capitolo, sono presentati i principali algoritmi adattativi per lo streaming di video. Nel quarto capitolo, sono valutatate le prestazioni degli algoritmi indicati tramite l'analisi grafica della QoE. Infine, il quinto capitolo riassume i risultati raggiunti dall'elaborato e discute i possibili sviluppi futuri.

Capitolo 2

Contesto teorico

Nel seguente capitolo, verrà esposto il concetto di streaming dinamico attraverso il protocollo DASH. A seguire, verranno riassunte le tecniche di apprendimento per rinforzo e apprendimento profondo. Queste tecniche vengono utilizzate da *Pensieve* [3] e *D-DASH* [4], algoritmi che fanno ricorso all'intelligenza artificiale al fine di sfruttare al meglio le capacità della rete per massimizzare la *QoE* dell'utente.

2.1 Streaming adattativo dinamico attraverso DASH

Lo streaming di contenuti video è la sorgente dominante di traffico proveniente da Internet e costituisce attualmente il 55% del traffico mobile [5]. *DASH* è diventato uno standard nel 2011 per innovare lo streaming video sfruttando non solo il protocollo HTTP ma anche un'infrastruttura chiamata *Content Delivery Network (CDN)* [4].

L'infrastruttura CDN è una rete di server proxy ovvero di server intermediari tra il server principale e l'utente finale e i relativi data center. I CDN distribuiscono contenuti mantenendo alta disponibilità e prestazioni.

HyperText Transfer Protocol (HTTP) è un protocollo di comunicazione attraverso cui il mondo web comunica e scambia informazioni [6]. In particolare, HTTP gestisce la comunicazione tra client e server permettendo la trasmissione di qualsiasi tipo di contenuto, compresi i video. Poiché HTTP si affida al *Transmission Control Protocol (TCP)*, che è un protocollo affidabile, HTTP garantisce che i dati non siano danneg-

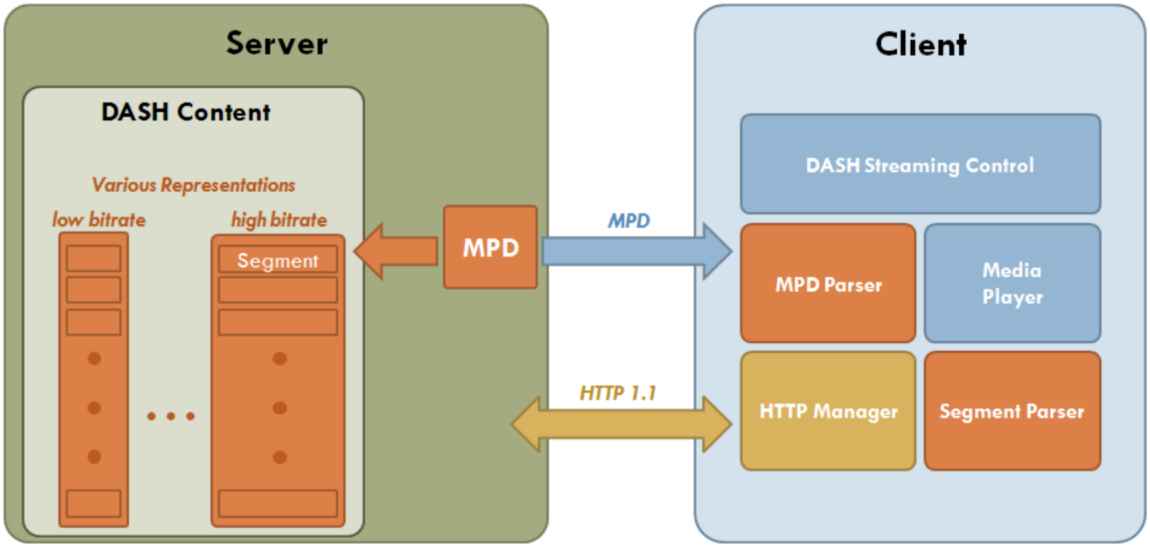


Figura 2-1: Illustrazione del funzionamento di DASH. ©bitmovin.net

giati durante la trasmissione. I server HTTP memorizzano informazioni, dette anche risorse, come i segmenti di un video che sono contenuti statici. I server provvedono a distribuire risorse ai client quando vengono richieste. I client comunicano con il server effettuando una richiesta chiamata *HTTP request* e rimangono in attesa di una risposta detta *HTTP response*.

Le HTTP requests principali sono: *GET* e *POST*. La prima effettua la richiesta di una specifica risorsa disponibile sul server mentre la seconda invia dati dal client ad una specifica risorsa del server.

Prima di iniziare la trasmissione del video sfruttando DASH, il client deve richiedere un file chiamato *Media Presentation Description (MPD)* contenente informazioni per la trasmissione, decodifica e riproduzione dei video tramite una *GET request*. Successivamente, le *GET request* verranno utilizzate per scaricare un segmento del video memorizzato sul server. Una *POST request*, invece, può essere utilizzata per inviare le statistiche della rete, raccolte dal client, al server.

L'MPD è un documento *eXtensive Markup Language (XML)* generalmente memorizzato sul server, contenente informazioni riguardanti i segmenti del file video. L'MPD è fondamentale perché contiene informazioni necessarie per la scelta di segmenti da scaricare che un utente probabilmente richiede dopo aver scaricato il primo

segmento del file. I video memorizzati sul server HTTP, accessibili ai client, sono divisi in segmenti di pochi secondi, ognuno dei quali viene codificato a livelli di compressione diversi per generare un set di adattamento ovvero un insieme di segmenti di qualità diverse che compongono il video. I segmenti di qualità migliore occupanno più memoria. Il file MPD permette al video player di conoscere i bitrate disponibili e quindi di cambiare bitrate, quando opportuno, per ogni segmento scaricato [7].

I blocchi principali di un file MPD sono i periodi *(Periods)* che descrivono il tempo d'inizio e la durata del video. Periodi multipli possono essere utilizzati per scene o capitoli oppure per separare la pubblicità dal contenuto del video. Un periodo può avere uno o più set di adattamento che contengono audio o video. Per favorire la trasmissione del video si possono separare audio e video in set di adattamento diversi. Per scegliere la lingua di un video, ad esempio, l'utente richiede un set di adattamento che contenga tale lingua come audio. I segmenti sono i contenuti multimediali che il player DASH riproduce. Le rappresentazioni di un set di adattamento permettono di avere lo stesso contenuto codificato in maniera diversa, ad esempio, con risoluzioni diverse per dispositivi differenti e anche per larghezze di banda variabili oppure per codec diversi. Se l'algoritmo DASH utilizzato è efficiente, permette agli utenti di raggiungere la massima QoE senza problemi di *rebuffering* ovvero senza che il video interrompa la propria riproduzione per attendere il download del segmento successivo. Se si prende in considerazione il player di YouTube, le rappresentazioni possono essere scelte manualmente dall'utente tra le qualità disponibili, tra cui 144p, 240p, 360p, 480p, 720p, 1080p, 1440p, 4K, 8K oppure la selezione può essere in 'auto' ovvero si può affidare al player la scelta di selezionare la qualità più appropriata. All'inizio dell'MPD, sono specificati vari set di adattamento con le relative rappresentazioni riguardanti il video in cui viene specificata la larghezza di banda e le dimensioni del video. I set di adattamento contengono $<$*SegmentTemplate*$>$ in cui è indicato dove si possono trovare i segmenti video in memoria. Le informazioni relative ai segmenti sono contenute in un tag $<$*VideoSegments*$>$ che include tutti i segmenti del video $<$*Segment*$>$. Ogni segmento è composto da $<$*chunk*$>$ ovvero da frammenti con relativa qualità (da 0 a 1) e rappresentazione. Nel player che utiliz-

ziamo, ci sono due proprietà per ogni segmento di complessita e indice della qualità percepita [8]. La complessità è una misura di correlazione tra un segmento e i successivi ovvero di dipendenza tra i contenuti del video. L'indice della qualità percepità è una quantificazione della qualità dell'immagine.

Ogni video è caratterizzato da una curva qualità-rate, descritta dalla funzione $F_t(q_t)$ che fornisce la dimensione del segmento t con qualità q_t [4]. $F_t(q_t)$ è nota prima di scaricare il segmento t; infatti, può essere ricavata dall'MPD oppure può essere calcolata da scene precedenti, in quanto correlate tra loro. Il tempo per scaricare il segmento t è dato da

$$\tau_t = \frac{F_t(q_t)}{C_t} \tag{2.1}$$

in cui C_t è la capacità media del canale in cui viene trasmesso il segmento t. La costante T indica il tempo di riproduzione del segmento del video. Il tempo di buffer di un segmento B_t è il tempo tra l'inizio dello scaricamento del segmento t e l'istante in cui il segmento viene riprodotto dal cliente. Eventi di rebuffering del video avvengono quando il buffer si svuota prima che il prossimo segmento sia stato scaricato completamente ovvero quando $\tau_t > B_t$. Contrariamente, se $\tau_t < B_t$, il segmento è stato scaricato e non avviene il rebuffering. Di conseguenza, se non avviene il rebuffering, si può iniziare a scaricare subito il segmento successivo, aggiungendo un tempo che è stato guadagnato per il buffer del prossimo segmento $t + 1$. Il tempo di rebuffering di un segmento t è dato da

$$\phi_t = max(0, \tau_t - B_t) \tag{2.2}$$

mentre il buffer del prossimo segmento è calcolato come

$$B_{t+1} = T + max(0, B_t - \tau_t) \tag{2.3}$$

Il buffer è limitato a 20s per occupare meno memoria.

La complessità degli algoritmi DASH risiede nel fatto che quando la selezione della qualità è automatica, bisogna passare da una rappresentazione all'altra in modo efficiente; questo è il motivo per cui è importante sviluppare algoritmi ottimi che

risolvano il problema più efficientemente possibile, in base alle condizioni dell'utente, fornendo la migliore QoE raggiungibile.

2.2 Apprendimento per rinforzo

Le tecniche di apprendimento per rinforzo *(RL)* sono utilizzate per imparare dall'ambiente circostante come effettuare azioni migliorative per il raggiungimento della soluzione di un problema. Per questo motivo, l'apprendimento per rinforzo è applicabile allo scenario presentato, allo scopo di sviluppare algoritmi DASH efficienti. Il modello comune che verrà descritto è il *Markov Decision's process (MDP)*. L'attore principale del RL è l'agente. L'agente che impara non sa quali azioni sono migliori, ma lo scopre con il tempo. Ad ogni azione dell'algoritmo viene attribuita una ricompensa *(reward)*. Le azioni migliori nei casi più complessi influiscono anche sulle azioni successive. Le due caratteristiche principali degli algoritmi di RL sono: *"trial-and-error" e "delayed reward"*. *"Trial-and-error"* è la strategia che punta a risolvere un problema ripetendo azioni fino ad ottenere un successo. *"Delayed reward"* è un modello che ritarda l'attribuzione della ricompensa in quanto le decisioni possono influire anche sulle azioni successive [7]. I quattro elementi importanti sfruttati da un algoritmo di RL sono: una policy, un segnale di reward, una funzione di valori, un modello dell'ambiente.

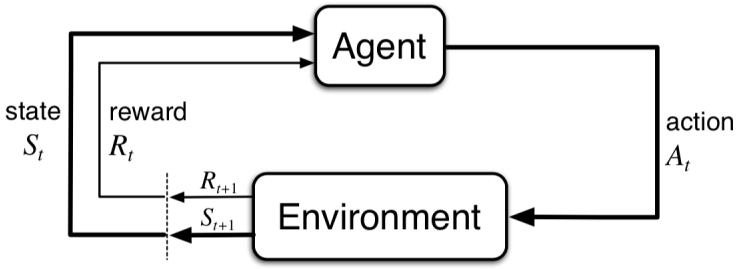


Figura 2-2: Illustrazione del concetto base dell'apprendimento per rinforzo. [7]

La *policy* è una politica che definisce il comportamento dell'agente ad un tempo definito. È una mappa che a stati dell'ambiente collega azioni da mettere in atto.

Un *segnale di reward* definisce l'obiettivo da raggiungere in un problema di RL. Ad ogni passo dell'algoritmo, viene inviato all'agente un valore chiamato reward, ovvero ricompensa. Le ricompense assegnate possono essere positive nel caso in cui l'agente stia procedendo in modo soddisfacente e negative in caso contrario. L'obiettivo dell'agente è massimizzare il reward. Un segnale di reward può influenzare direttamente la policy da adottare. Questi segnali sono, in generale, funzioni stocastiche dello stato dell'ambiente e delle azioni compiute e indicano come è opportuno agire per azioni intermedie.

Una *funzione di valore* specifica che cosa è opportuno fare in caso di azioni successive. Il valore di uno stato è la somma totale attesa delle ricompense che un agente può accumulare nel futuro, partendo da quello stato.

Per ottenere la massima ricompensa, un algoritmo di RL preferisce sfruttare azioni che ha compiuto in passato ed esplorare l'ambiente per capire quali azioni sono migliori per il futuro. Le azioni, spesso, sono poste in essere basandosi sui valori. Purtroppo, i valori sono più difficili da determinare rispetto alle ricompense, in quanto i valori devono essere stimati dalla sequenza di osservazione di un agente nel corso del suo intero periodo di esecuzione, mentre le ricompense sono fornite quasi direttamente dall'ambiente. La componente più importante degli algoritmi di RL è, infatti, la determinazione dei valori in modo efficiente. Ottenere un modello dell'ambiente affidabile permette di capire come l'ambiente si comporterà in futuro. Ad esempio, dato uno stato ed una ricompensa, il modello permette di prevedere lo stato successivo e la prossima ricompensa. Questo è utile per considerare azioni future prima di procedere su tale percorso.

Lo stato è un parametro utilizzato per comunicare all'agente l'informazione relativa a come l'algoritmo sta agendo sull'ambiente, in un dato momento. Lo stato è l'input della policy e della funzione dei valori, oltre ad essere input e output del modello [9].

Analiticamente, l'agente, negli istanti $t = 0,1,2,3,...$, riceve in input uno stato $s_t \in A(s_t)$ in cui $A(s_t)$ è l'insieme delle possibili azioni che può compiere l'agente, dato tale stato. Ad ogni passo, l'associazione stato-azione identifica la policy π con

cui l'agente intende massimizzare la ricompensa nel lungo termine. Migliorare la policy permette l'apprendimento e l'accumularsi dell'esperienza. Di conseguenza, viene quantificata la reward r_{t+1} e ciò permette di passare allo stato successivo s_{t+1}.

Gli algoritmi sviluppati sul RL consentono di superare due problemi: il carico computazionale e la necessità di avere a disposizione in anticipo statistiche sullo stato della rete e del video. Gli algoritmi basati sul RL imparano le statistiche della rete dall'esperienza e la loro complessità è bassa. Il loro limite sta nel fatto che necessitano di una certa quantità di esperienza per prendere una decisione soddisfacente. Il secondo problema si presenta quando il numero di stati è alto poiché cresce il tempo necessario all'algoritmo per imparare. Per essere adattabile, l'algoritmo deve visitare e aggiornare il proprio stato frequentemente ma questo comporta che la conoscenza dell'ambiente risulti peggiore, in presenza di pochi stati [4].

2.3 Apprendimento profondo

L'apprendimento profondo (Deep Learning) consiste nell'utilizzo di reti neurali per ricavare un modello che in DASH può essere sfruttato per ottenere la qualità del prossimo segmento da scaricare. Le reti neurali sono composte da unità interconnesse chiamate neuroni. I neuroni sono caratterizzati da:

- ingressi a valori reali $x_1, x_2, ..., x_n$ filtrati attraverso i pesi $w_1, w_2, ..., w_n$;
- un sommatore da cui si ottiene la combinazione lineare u degli ingressi pesati;
- una funzione d'uscita (o di attivazione) determinata da una funzione non lineare σ di variabile $u + b$ in cui b è il *bias* associato.

La funzione di attivazione si può esprimere come:

$$y = \sigma(u + b) \tag{2.4}$$

in cui la combinazione lineare u è data da:

$$u = \sum_{i=1}^{n} w_i \cdot x_i \tag{2.5}$$

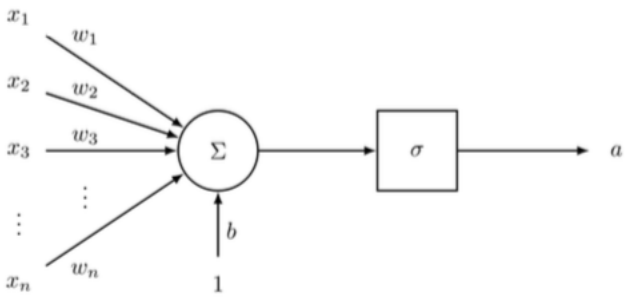


Figura 2-3: Modello di un neurone [8].

Tali reti sono sfruttate dall'apprendimento per rinforzo allo scopo di fornire all'agente l'azione migliore da compiere, dato un certo stato. Le reti sono composte da vari livelli detti *layer*. Le reti utilizzate nel nostro caso sono *feedforward* non perché contengono loop ovvero l'output di un certo livello della rete non può influenzare l'input di livelli precedenti o se stesso. In base all'algoritmo utilizzato, la rete riceve in input un certo numero di parametri e fornisce un output. Inoltre, a parte il primo livello che è attivato dai dati in ingresso, ogni livello ha un input dato dai neuroni precedentemente attivati e innesca un certo numero di neuroni al livello successivo tramite la funzione sopra indicata [10].

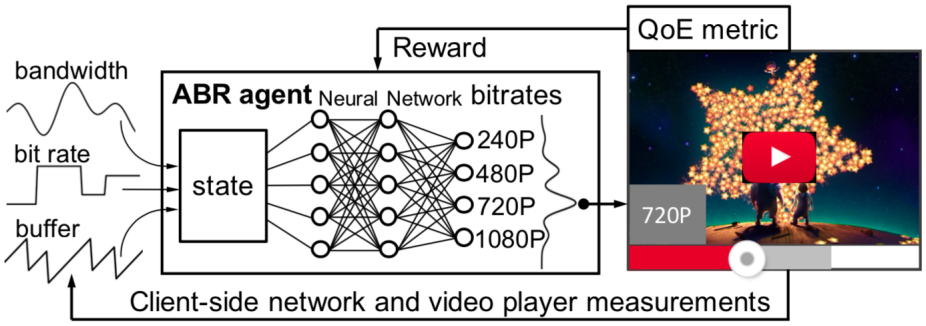


Figura 2-4: Il concetto delle reti neurali applicato a DASH. [3]

Una fase importante per l'utilizzo della rete neurale consiste nell'allenamento dei livelli intermedi. Allenare i livelli intermedi significa valutare i pesi dei collegamenti. Questa fase è importante perchè la funzione $f(x)$, data da una catena di funzioni ciascuna per ogni livello, deve convergere ad una funzione target. Al fine di raggiungere la funzione target, la tecnica di apprendimento varia i pesi degli ingressi dei neuroni.

Capitolo 3

Introduzione agli algoritmi adattativi

In questo capitolo verrà presentato il funzionamento dei più importanti algoritmi DASH. Al fine di simulare i risultati esposti nel capitolo seguente, sono stati utilizzati due player. Il primo è stato sviluppato da Davide Talon [8]. Alternativamente, Hongzi Mao ha sviluppato un player in javascript [3].

Per sviluppare algoritmi DASH attraverso il player prodotto da Davide Talon in Java, si estende la classe *"DashAlgorithm"* che esegue un *"Thread"*. Ogni algoritmo DASH svolge delle operazioni comuni. La prima operazione che compie è scaricare il file MPD dal server. Successivamente, vengono estratti e memorizzati attraverso una lettura del file MPD i bitrate, le complessità e le qualità disponibili. Vengono impostati il massimo bitrate, la durata del segmento e le qualità in base all'algoritmo utilizzato. Ogni algoritmo DASH scarica il primo segmento per ogni qualità disponibile quindi entra in un loop al fine di ottenere il segmento successivo disponibile. La fase in cui si ottiene il segmento successivo è la parte chiave di ogni algoritmo DASH. In questa fase, viene utilizzato il MDP per elaborare i dati ottenuti dall'algoritmo. Dopo l'inizializzazione, il MDP si muove allo stato successivo, salvando dati e impostando i valori iniziali per l'array degli stati. Il fulcro dell'algoritmo consiste nel calcolare la migliore azione possibile da eseguire. La migliore azione è associata ad un indice di valore intero che specifica a quale bitrate scaricare il segmento corrente, puntando a massimizzare QoE. Il segmento viene scaricato al bitrate specificato e può essere riprodotto visivamente. Il MDP elabora lo stato successivo. Se il buffer è

vuoto, avviene un evento di rebuffering.

Gli algoritmi prodotti da Hongzi Mao sfruttano dash.js, un player scritto in linguaggio javascript. Attraverso pagine web HTML, l'utente può accedere al contenuto video che viene riprodotto attraverso il player. In base all'algoritmo scelto, il player invierà richieste multiple al server per richiedere i segmenti successivi da scaricare. Il server, oltre a memorizzare i segmenti del video e il relativo file MPD, avvia un gestore di richieste HTTP *(HTTP request handler)* basato sul linguaggio python che si pone in ascolto di richieste da parte degli utenti. Il server risponderà all'utente che richiede segmenti tramite *POST requests*, fornendo l'informazione sull'azione da compiere ovvero sulla qualità da utilizzare per il segmento da scaricare.

Dopo aver richiesto il file MPD, il client richiede chunk del video uno dopo l'altro, usando algoritmi di adattamento per il bitrate *(Adaptive Bitrate Algorithm - ABR)*. Questi algoritmi utilizzano diversi input come l'occupazione del buffer, la misura del throughput e altro, per selezionare la qualità dei chunk successivi.

Il vantaggio dell'approccio di Hongzi Mao consiste nell'evitare il calcolo computazionale e lo spazio di memoria da parte dell'utente che, in caso di algoritmi come D-DASH o Pensieve, è considerevole in quanto entrambi fanno ricorso ad una rete neurale che memorizza grandi quantità di dati.

3.0.1 Algoritmo basato sul Bit-Rate

Il fulcro di ogni algoritmo consiste nella scelta dell'azione da compiere. Algoritmi basati sul Bit-Rate compiono un'azione semplice il cui calcolo della strategia che porta alla scelta dell'azione è complesso. Inizialmente, l'azione viene impostata con l'indice riferito al bitrate più basso. Se l'ultimo bitrate calcolato, dopo aver scaricato l'ultimo segmento, è superiore o uguale al bitrate più basso, l'azione cambia indice e si riferisce via via al bitrate superiore fino a quando la condizione non si verifica più. In questo modo, l'azione si riferirà ad un bitrate disponibile, basandosi sul valore del bitrate precedente. Riassumendo, il bitrate è calcolato nel seguente modo:

- Se $R_{b_i} \leq R_{current}$ allora scelgo R_{b_i} come bitrate per scaricare il prossimo segmento.

- altrimenti il bitrate sarà il peggiore possibile.

Come notazione, si è utilizzato R_{b_i} per indicare il bitrate della rappresentazione i-esima. $R_{current}$, invece, indica il bitrate dell'ultimo segmento scaricato.

3.0.2 MPC

Per affrontare decisioni basandosi su misure future è stato sviluppato un sistema detto *Model Predictive Control (MPC)* [11]. MPC si affida ad un predittore del throughput per effettuare decisioni ottime in un orizzonte finito di 5 segmenti futuri. MPC combina misure di throughput e occupazione del buffer ma se il predittore delle statistiche del canale è errato, anche la logica di decisione della qualità per il prossimo segmento sarà errata. La decisione sul bitrate da selezionare per il prossimo segmento è funzione dell'occupazione del buffer, della capacità stimata dei 5 segmenti futuri e dal bitrate dei segmenti scaricati precedentemente. Questo approccio comporta un miglioramento in termini di QoE rispetto all'algoritmo basato sul Bit-Rate in quanto tiene in considerazione più aspetti relativi alle statistiche della rete. L'algoritmo compie i seguenti passi:

1. *Predizione:* Predizione del throughput per i prossimi N segmenti usando un predittore di throughput.

2. *Ottimizzazione:* Nella fase di startup, ottimizza il tempo di start-up mentre nella fase di playback, utilizza un'equazione per il calcolo del bitrate che punta a massimizzare la QoE calcolata matematicamente.

3. *Applicazione:* Viene scaricato il segmento successivo con bitrate specificato e l'orizzonte è spostato avanti.

3.0.3 FESTIVE

FESTIVE utilizza una funzione di stabilità dei costi e limita la frequenza d'incremento del bitrate per privilegiare la stabilità rispetto alla qualità istantanea. FESTIVE ha la caratteristica di essere più equo in termini di QoE, riducendo gli eventi di rebuffering come conseguenza del suo approccio conservativo sull'incremento del bitrate e puntando a ridurre le variazioni di qualità [12]. Il suo algoritmo esegue questi passaggi:

1. *Stima armonica della bandwidth:* Compie una stima armonica di 20 misure di throughput passate.

2. *Aggiornamento del bitrate:* In base alla bandwidth calcolata nel primo punto, viene calcolato il bitrate di riferimento. Il bitrate scelto è il primo bitrate disponibile di valore inferiore alla bandwidth. Il bitrate viene aumentato solo dopo un certo numero di segmenti ma se necessario viene diminuito dopo ogni segmento.

3. *Scheduling:* Il prossimo segmento viene scaricato immediatamente se la capienza del buffer è inferiore a quella desiderata, altrimenti il suo download è rimandato dopo un tempo casuale.

3.0.4 PANDA

Probe and Adapt (PANDA) usa una strategia per valutare la capacità del canale e calcola il bitrate in base a tale valore, incrementando la qualità progressivamente e contando di prevenire le fluttuazioni. L'algoritmo descritto in [13] può essere riassunto come segue:

1. *Stima del throughput:* Predizione del throughput

2. *Smoothing:* Alla predizione del throughput, viene rimosso il rumore in modo da avere una stima più accurata.

3. *Quantizzazione:* Il valore trovato viene mappato in un possibile bitrate discreto, tenendo in considerazione la dimensione attuale del buffer.

4. *Scheduling:* L'algoritmo imposta un intervallo di tempo fino al prossimo segmento da scaricare.

3.1 Funzionamento di Pensieve

Pensieve è un algoritmo che usa tecniche di RL [3]. Pensieve allena il modello di una rete neurale che seleziona il bitrate per i chunk successivi del video, basandosi sull'osservazione di risultati precedenti. È stato dimostrato che l'utilizzo di Pensieve migliora la QoE del 12%-15% in media rispetto agli altri algoritmi presenti in letteratura. Pensieve, attraverso il RL, impara una policy per l'adattamento del bitrate attraverso l'esperienza, assegnando delle ricompense, come esposto nel Capitolo 2. Pensieve può anche imparare quanto buffer atto alla riproduzione è necessario al fine di evitare il rischio di rebuffering, basandosi sulla variabilità del throughput. Inoltre, può sapere se conviene affidarsi al throughput oppure al buffer per effettuare le scelte successive. La policy di controllo è rappresentata sotto forma di modello di una rete neurale che mappa osservazioni grezze *(raw)* dello stato della rete in bitrate scelto per il chunk successivo. Il modello della rete neurale creato è memorizzato sul server ABR; quindi, il calcolo computazionale effettuato dal client è minimo perchè deve solo richiedere al server a quale bitrate scaricare il segmento successivo. Le richieste da parte del client al server includono osservazioni sulle statistiche della rete riguardanti il throughput, l'occupazione di buffer e le proprietà del video.

3.2 Implementazione di Pensieve

Pensieve genera una policy e una strategia basata sul RL e le applica alle sessioni di video streaming. In questa sezione, verranno esposti la metodologia di allenamento e l'algoritmo utilizzati da Pensieve. Pensieve, inizialmente, avvia una fase di allenamento nella quale l'agente esplora l'ambiente del video streaming. Per velocizzare

la fase di training, lo sviluppatore dell'algoritmo ha preferito utilizzare un semplice ambiente di simulazione che modella le dinamiche dello streaming video. Il simulatore mantiene una rappresentazione interna del buffer di riproduzione. Per ogni chunk da scaricare, il simulatore assegna un tempo di scaricamento basato solamente sul bitrate del chunk e le tracce del *throughput* in input. Successivamente, il simulatore ripulisce il buffer del tempo di riproduzione del segmento che viene riprodotto durante lo scaricamento e aggiunge la durata di riproduzione del chunk scaricato al buffer. Il simulatore tiene anche traccia degli eventi di rebuffering ovvero di quando il tempo di scaricamento è superiore all'occupazione del buffer, all'inizio dello scaricamento; in tal caso, attende 500ms prima di riprovare a scaricare. Dopo aver scaricato ogni chunk, il simulatore fornisce uno stato, descritto nella prossima sezione, all'agente che lo processa.

3.2.1 Algoritmo di allenamento

L'algoritmo per allenamento di Pensieve utilizza A3C [14] che allena due reti neurali: actor network e critic network. L'input e la policy di tale algoritmo verranno approfonditi nella sezione seguente.

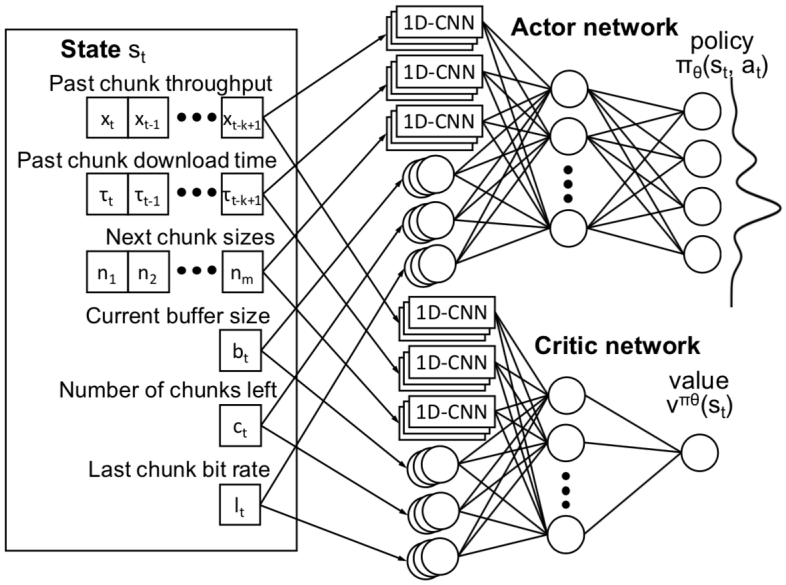


Figura 3-1: L'algoritmo che Pensieve usa per generare Policy ABR. [3]

Input: Dopo aver scaricato ogni chunk t, l'agente immette in input uno stato $s_t = (\vec{x}_t, \vec{\tau}_t, \vec{n}_t, b_t, c_t, l_t)$ alla rete neurale. $\vec{x}_t$ è il vettore delle k misure dei throughput precedenti, $\vec{\tau}_t$ è il vettore del tempo di download dei k chunk precedenti, $\vec{n}_t$ è il vettore delle m dimensioni del chunk successivo a tutti i bitrate disponibili, b_t è il livello del buffer attuale, c_t è il numero di chunk rimanenti nel video, l_t è il bitrate dell'ultimo chunk scaricato.

Policy: La policy è una distribuzione di probabilità che ha come parametri s_t e a_t. Matematicamente, si esprime come $\pi(s_t, a_t) \to [0, 1]$. La policy è la probabilità che l'azione a_t venga intrapresa allo stato s_t. Il problema consiste nel fatto che esistono molte coppie stato-azione ed è quindi necessario l'uso di una rete neurale *(neural network - NN)* per rappresentare la policy con alcuni parametri θ detti parametri della policy.

Usando θ, è quindi possibile rappresentare la policy come $\pi_\theta(s_t, a_t)$.

Gradiente di allenamento della policy: Dopo aver applicato ogni azione, l'ambiente simulato genera una ricompensa r_t per il chunk considerato. La ricompensa è impostata per riflettere le prestazioni di scaricamento di ogni chunk in accordo con le metriche della QoE che si desiderano ottimizzare. L'algoritmo utilizzato per allenare la policy di Pensieve è detto *policy gradient method* [15]. L'idea chiave di questo metodo consiste nello stimare il gradiente del valore atteso in merito al totale della ricompensa, osservando le traiettorie di esecuzione ottenute dalla policy. Il gradiente del totale della ricompensa, in base ai parametri θ, è fornito da [14]:

$$\bigtriangledown_\theta \mathbb{E}_{\pi_\theta}[\sum_0^\infty \gamma^t r_t] = \mathbb{E}_{\pi_\theta}[\bigtriangledown_\theta \ \log \pi_\theta(s, a) A^{\pi_\theta}(s, a)] \tag{3.1}$$

$A^{\pi_\theta}(s, a)$ è la funzione di vantaggio (*advantage*) che rappresenta la differenza nel valore atteso della ricompensa totale quando scegliamo in modo deterministico un'azione a allo stato s, messo a confronto con il valore atteso della ricompensa per azioni intraprese dalla policy π_θ. La funzione di vantaggio $A^{\pi_\theta}(s, a)$ indica quanto un'azione risulti a confronto con l'azione media ottenuta in base alla policy.

In pratica, l'agente campiona una traiettoria di bitrate di decisione e usa *A(s,a)*

come una stima unbiased di $A^{\pi_\theta}(s,a)$. Ogni aggiornamento di θ ovvero la regola con cui si aggiorna l'agente avviene come segue:

$$\theta \longleftarrow \theta + \alpha \sum_t \bigtriangledown_\theta \ \log \pi_\theta(s_t, a_t) A(s_t, a_t) \tag{3.2}$$

in cui α è il tasso di apprendimento. $\bigtriangledown_\theta \ \log \pi_\theta(s_t, a_t)$ specifica come cambiare i parametri della policy per aumentare l'argomento del logaritmo. L'ampiezza del salto dipende dal valore della funzione di vantaggio per l'azione a_t allo stato s_t.

Per calcolare il vantaggio $A(s_t, a_t)$ per una data esperienza, serve stimare la funzione dei valori e il valore atteso totale della ricompensa che parte allo stato s e applica la policy π_θ.

Infine, è necessario assicurarsi che l'agente esplori adeguatamente lo spazio delle azioni durante l'allenamento per scoprire le policy ottimali. Per farlo, occorre modificare l'aggiornamento di θ, aggiungendo l'entropia della policy $H(\cdot)$ ad ogni step:

$$\theta \longleftarrow \theta + \alpha \sum_t \bigtriangledown_\theta \ \log \pi_\theta(s_t, a_t) A(s_t, a_t) + \beta \bigtriangledown_\theta H(\pi_\theta(\cdot)|s_t) \tag{3.3}$$

Il termine aggiunto incoraggia l'esplorazione verso un'entropia più alta. Il parametro β è impostato ad un valore alto all'inizio dell'allenamento, per incoraggiare l'esplorazione e il parametro decresce nel tempo per enfatizzare i miglioramenti della ricompensa.

Allenamento parallelo: Per migliorare e velocizzare ulteriormente la fase di allenamento, Pensieve usa 16 agenti in parallelo, ognuno dei quali riceve in input parametri diversi. Gli agenti inviano tuple stato, azione, ricompensa di continuo all'agente centrale che li aggrega in un singolo modello di algoritmo ABR. Per ogni tupla ricevuta, l'agente centrale usa un algoritmo *actor critic* per calcolare un gradiente [15]. L'agente centrale aggiorna successivamente l'*actor network* e invia il nuovo modello all'agente che ha inviato la tupla.

3.3 Implementazione di D-DASH

D-DASH è un framework che, analogamente a Pensieve, combina tecniche di RL e apprendimento profondo. Anche D-DASH offre una convergenza, nella funzione di reward, più veloce per la scelta della qualità di un video.

3.3.1 Algoritmo di allenamento

D-DASH si basa sul Markov Decision Process (MDP). Il MDP è un modello del servizio di streaming adattativo se si modella il contenuto del video come una sequenza di scene con durata distribuita esponenzialmente [4].

Input: Dopo aver scaricato ogni chunk t, l'agente immette in input uno stato $s_t = (q_{t-1}, \vec{C_t}, \phi_t, B_t)$ alla rete neurale. q_{t-1} è la qualità del segmento precedentemente scaricato, $\vec{C_t}$ è un vettore di tutte le capacità precedenti dall'inzio della sessione di scaricamento, ϕ_t è il tempo di rebuffering definito nel Capitolo 2 e infine B_t il buffer attuale.

Policy: La policy in D-DASH ha come parametri lo stato s_t, la distribuzione statistica dello stato successivo s_{t+1} e l'azione $q_t \in A$. La policy è la probabilità che l'azione a_t venga intrapresa allo stato s_t, mappa quindi stati in azioni. La *long term utility* è il limite della media della funzione di reward pesata su un esponenziale che ne garantisce la convergenza:

$$R(s_0; \Pi) = \lim_{x \to \inf} E[\sum\nolimits_{t=0}^{h} \lambda^t \rho(s_t, s_{t+1}, \Pi(s_t) | s_0, \pi] \tag{3.4}$$

La policy è ottimale se massimizza la *long term utility*.

$$\arg\max_{\Pi} R(s; \Pi) \tag{3.5}$$

Lo spazio delle azioni è l'insieme delle possibili rappresentazioni dei segmenti del video. La policy rappresenta l'azione corrispondente alla qualità del segmento da scaricare $q_t = \Pi(s_t)$.

3.4 Confronto tra Pensieve e D-DASH

Entrambi D-DASH e Pensieve utilizzano tecniche di RL e apprendimento profondo al fine di ottimizzare la qualità dei video trasmessi e migliorare la QoE. Pensieve, a differenza di D-DASH, non richiede calcolo computazionale da parte del client poichè l'algoritmo viene eseguito sul server. Il client deve indicare al server il bitrate del segmento successivo. Come dimostrato in [3], il tempo di latenza dovuto al RTT (Round Trip Time) ha un impatto minimo sulla QoE in quanto viene mascherato dal vantaggio ottenuto dall'occupazione di buffer e dal tempo di scaricamento dei segmenti. Questo è un vantaggio da non sottovalutare in quanto l'algoritmo sviluppato in tal modo può essere applicato ad un numero di servizi molto più vasto, come televisori e dispositivi mobili. Un'altra differenza tra i due algoritmi consiste nell'applicazione di uno stato leggermente diverso fornito in input alla rete neurale. Pensieve utilizza come stato $s_t = (\vec{x}_t, \vec{\tau}_t, \vec{n}_t, b_t, c_t, l_t)$ descritto nella sezione precedente. D-DASH ha uno stato pari a $s_t = (q_{t-1}, \vec{C}_t, \phi_t, B_t)$. Il parametro $\vec{C}_{\mathrm{t}}$ è utile per calcolare la capacità statistica $\mathrm{C_t}$ basandosi sulle capacità precedenti $\vec{C}_t = [C_{t\text{-}n}, C_{t\text{-}n+1}, ..., C_{t\text{-}1}]$. Queste piccole differenze nello stato possono portare a valori leggermente diversi nel calcolo dell'azione finale, in particolare, per l'utilizzo di vettori di misure precedenti da parte di Pensieve, anziché scalari, e del parametro di complessità da parte di D-DASH che non è presente in Pensieve.

Capitolo 4

Simulazioni e risultati

Al fine di simulare i risultati, ho utilizzato il player prodotto da Davide Talon [8] ed il client e server sviluppati per Pensieve da Hongzi Mao [3].

4.0.1 Qualità dell'esperienza

Per analizzare i risultati ottenuti, ho tenuto in considerazione alcune metriche relative ad ogni segmento per i vari algoritmi specificati. Queste metriche influiscono sulla QoE dell'utente. La QoE è la percezione della soddisfazione da parte dell'utente finale che osserva la riproduzione del video. Essa può essere specificata secondo caratteristiche diverse. In [4] sono state scelte le seguenti metriche:

- **qualità**: q_t rappresentata da un'indice reale variabile da 0 a 1, in cui 0 rappresenta la qualità peggiore e 1 la qualità migliore per ogni segmento [16].

- **variazioni di qualità *(delta quality)***: è un'indice reale ricavabile dalla qualità variabile da -1 a 1, in cui più è alto il modulo del valore e più la variazione di qualità tra segmenti successivi è ampia. Variazioni ampie possono rilevarsi fastidiose per l'utente [17]. Il suo valore si può ricavare nel seguente modo:

$$\Delta(q_t) = q_t - q_{t-1} \tag{4.1}$$

- **tempi di rebuffering**: sono i tempi per cui il video interrompe la propria riproduzione per attendere il download del segmento successivo come spiegato più dettagliatamente nel Capitolo 2. La frequenza degli eventi di rebuffering e la loro durata influiscono considerevolmente sulla QoE, quindi è importante anticiparli alla riproduzione del video [18].

Per l'analisi delle prestazioni degli algoritmi, sono state effettuate varie simulazioni in due condizioni di canale diverse: mobilità attraverso reti 3G o 4G e hostpot pubblico tramite connessione Wi-Fi. Queste condizioni fanno variare molto il bitrate. Nel primo caso, la mobilità comporta il cambio di celle telefoniche e la presenza di fenomeni negativi legati alla trasmissione radio come le riflessioni. Nel secondo caso, l'instabilità della rete è condizionata da fattori esterni come il numero di utenti connessi al router e le limitazioni imposte dal provider. Nel player di Davide Talon, è possibile ottenere tutte le metriche specificate mentre nel player di Hongzi Mao [3] sono stati ricavati solo bitrate e tempi di rebuffering perchè l'indice di qualità utilizzato è diverso da quello appena specificato. Come campione, per il RL, ho valutato D-DASH.

4.1 Analisi delle prestazioni in 3G o 4G

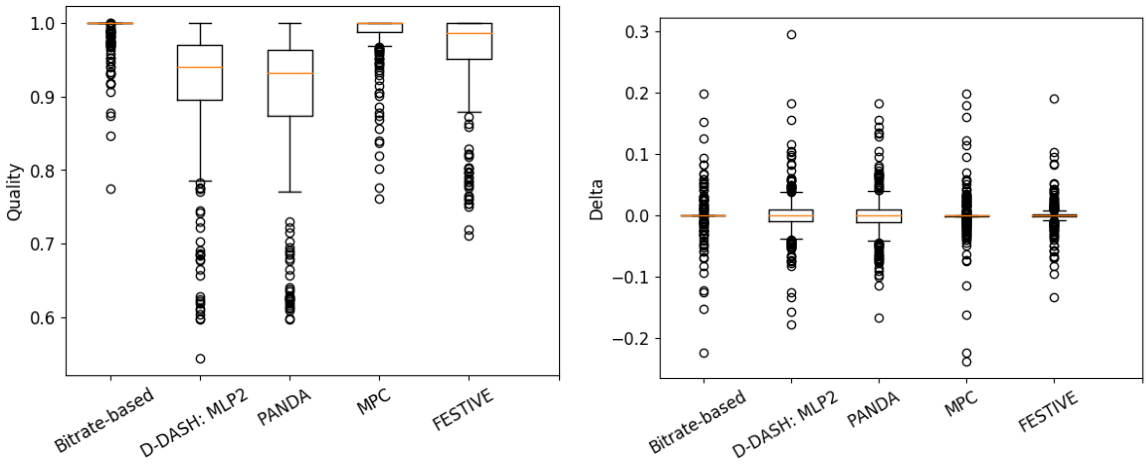
Per le prove in mobilità sono stati scelti due percorsi:

1. il tragitto in tram andata e ritorno da "Cuoco" a "Piazzale Stazione".

2. il tragitto andata e ritorno Padova-Treviso in auto, percorrendo la strada "Via Noalese".

Dal tragitto in tram, in connessione 4G, è stato possibile osservare quanto la capacità della rete sia influente sulla QoE. In ordine decrescente la capacità misurata è: Bitrate-based, D-DASH, MPC, FESTIVE, PANDA.

In termini di qualità, visionando il boxplot, l'algoritmo Bitrate-based ha riportato risultati ottimi, conseguenza dell'alta capacità della rete nell'area urbana in cui è stato eseguito l'algoritmo. A seguire, MPC e FESTIVE hanno ottenuto ottime



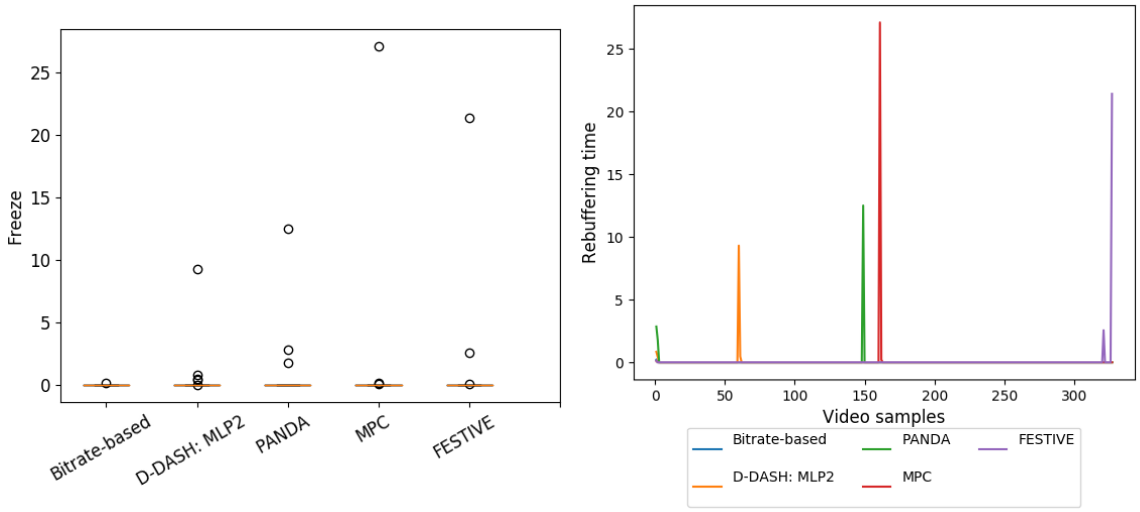
(a) Boxplot della qualità.

(b) Boxplot del delta qualità.

Figura 4-1: Simulazioni sul tram.

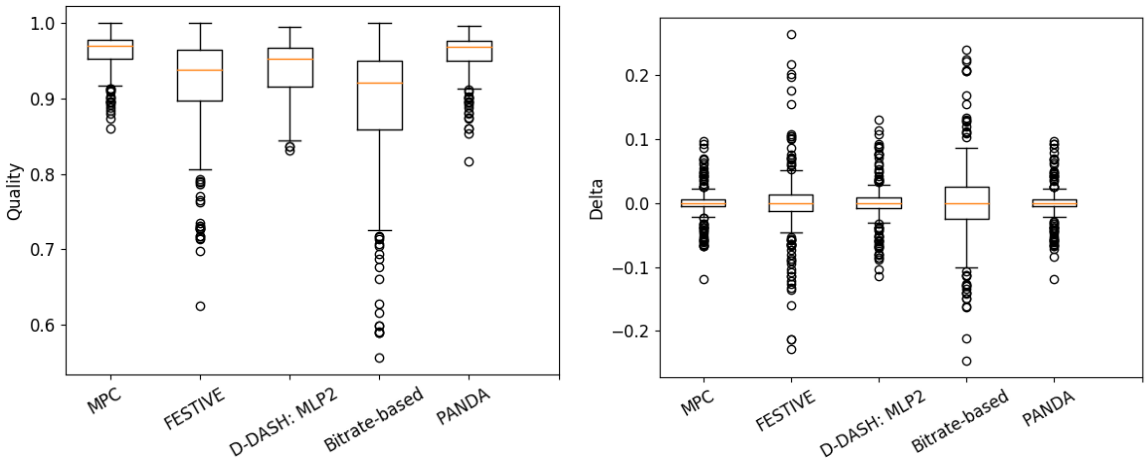
prestazioni mentre D-DASH e PANDA sono risultati peggiori. Complessivamente, Bitrate-based, MPC e FESTIVE hanno avuto variazioni di qualità minori rispetto a D-DASH e PANDA che hanno riportato variazioni equiparabili. Bitrate-based non ha avuto eventi di rebuffering. MPC e FESTIVE hanno compiuto un lungo evento di rebuffering, con durata superiore a 20 secondi, che può rivelarsi molto fastidioso al momento della visione del video. È emerso che FESTIVE lo abbia compiuto alla fine del video. Anche PANDA e D-DASH hanno effettuato un rebuffering intermedio con durata intorno ai 10 secondi. Complessivamente, la QoE misurata in Bitrate-based è stata migliore, ma nelle prove successive, ho constatato come Bitrate-based non abbia prestazioni ottime con velocità della rete diversa.

Nel tragitto Treviso-Padova mostrato nelle seguenti figure, D-DASH ha ottenuto prestazioni leggermente inferiori in termini di qualità e variazioni di qualità rispetto ad MPC e PANDA. Il motivo può essere dovuto alla rete neurale di D-DASH che non è stata allenata prolungatamente in condizioni di mobilità ma attraverso rete Wi-Fi. FESTIVE, invece, appare inferiore in termini di qualità, variazioni di qualità ed eventi di rebuffering. Si può notare come FESTIVE e Bitrate-based abbiano eventi di rebuffering intermedi che influiscono negativamente sulla QoE. D-DASH, MPC e PANDA sono complessivamente le scelte migliori nell'esperimento effettuato.



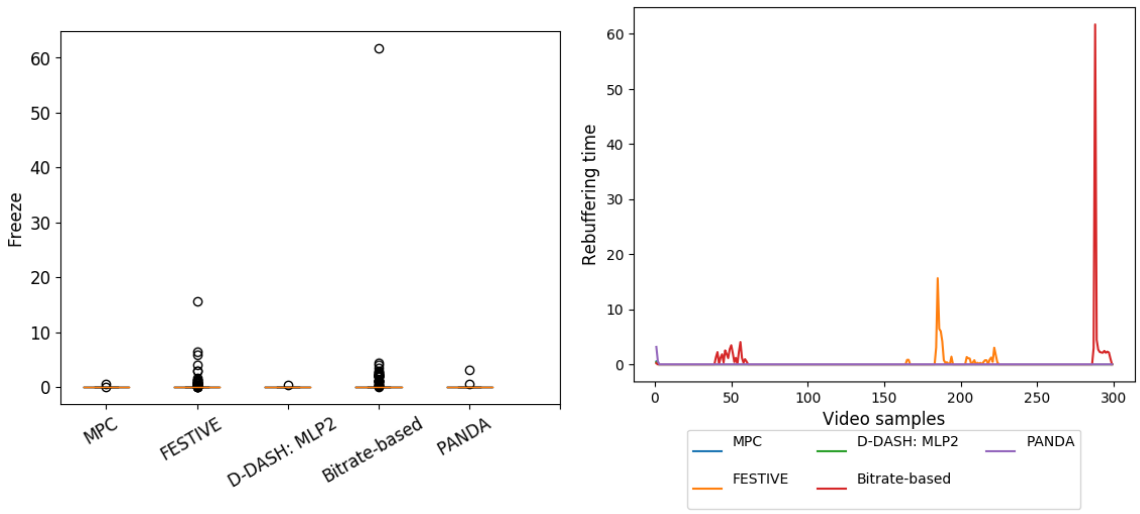
(a) Boxplot dei tempi di rebuffering. (b) Grafico dei tempi di rebuffering.

Figura 4-2: Simulazioni sul tram.



(a) Boxplot della qualità. (b) Boxplot del delta qualità.

Figura 4-3: Simulazioni sul tragitto Treviso-Padova.



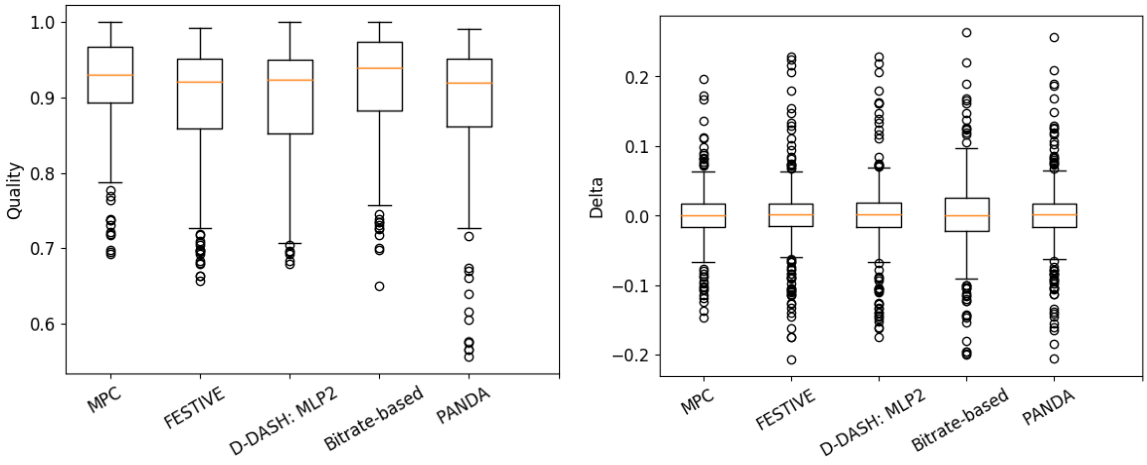
(a) Boxplot dei tempi di rebuffering.

(b) Grafico dei tempi di rebuffering.

Figura 4-4: Simulazioni sul tragitto Treviso-Padova.

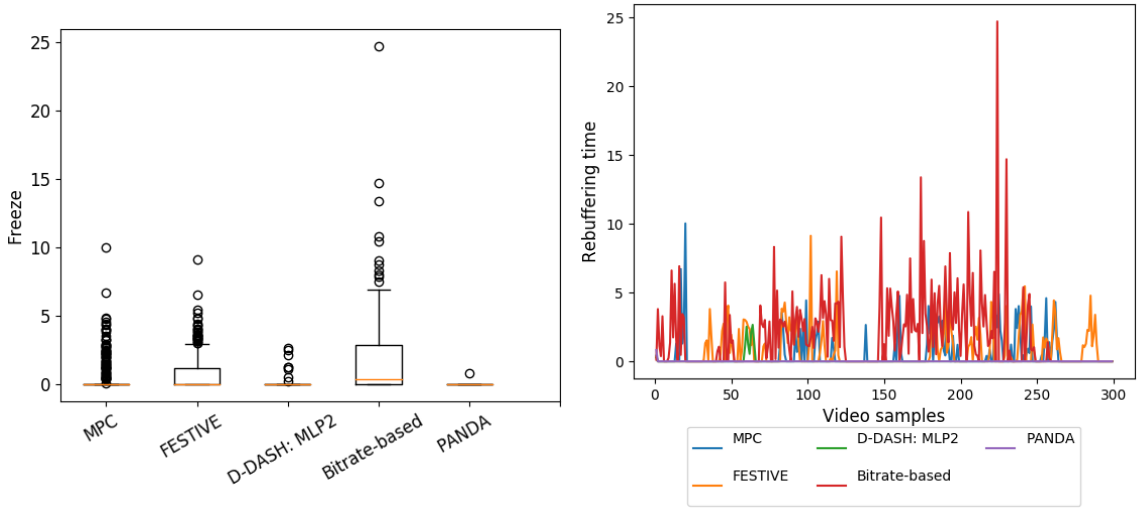
4.2 Analisi delle prestazioni tramite hotspot pubblico

Dai grafici prodotti, è possibile osservare come la qualità sia ottima in ogni algoritmo. MPC e D-DASH, però, sono risultati migliori in termini di variazioni di qualità. Dal punto di vista del rebuffering, invece Bitrate-based, FESTIVE ed MPC non sono soddisfacenti, presentando numerosi eventi di rebuffering di lunga durata. PANDA e D-DASH sembrano essere le scelte migliori nelle condizioni in cui è stato effettuato il test, presentando pochi eventi di rebuffering di durata di gran lunga inferiore agli altri algoritmi e garantendo una QoE ottima. Questo test conferma l'ipotesi che l'allenamento di D-DASH in condizioni di rete simili a quella utilizzata, ovvero in Wi-Fi, produca un'efficacia maggiore.



(a) Boxplot della qualità. (b) Boxplot del delta qualità.

Figura 4-5: Simulazioni tramite hotspot pubblico.



(a) Boxplot dei tempi di rebuffering. (b) Grafico dei tempi di rebuffering.

Figura 4-6: Simulazioni tramite hotspot pubblico.

Capitolo 5

Conclusioni e futuro lavoro

In questa tesi, è stato appurato che è necessaria l'analisi della qualità dell'esperienza, in diverse condizioni di rete, al fine di valutare le potenzialità complessive degli algoritmi. L'applicazione di algoritmi DASH basati sul RL e sulle reti neurali ottiene miglioramenti di QoE rispetto agli algoritmi deterministici, se la rete neurale viene allenata in Wi-Fi e utilizzata in Wi-Fi oppure allenata in 4G e usata in 4G. Sfortunatamente, D-DASH e Pensieve richiedono un calcolo computazionale più elaborato, rispettivamente, uno da parte del client e l'altro del server. Dalle misure riportate, MPC e PANDA si sono dimostrati ottimi competitori di D-DASH. I parametri tenuti in considerazione, tra cui capienza del buffer, qualità e variazioni di qualità sono di fondamentale importanza. Tuttavia, un possibile sviluppo nella valutazione della QoE consiste nella stima della fairness. La fairness misura l'equità dell'algoritmo ovvero il suo comportamento in presenza di più utenti che scaricano lo stesso video da un unico server. Nonostante la fairness sia complessa da valutare, in quanto richiede l'utilizzo di più macchine client, può essere utile per identificare algoritmi consoni all'implementazione da parte di compagnie che forniscono servizi di streaming video. Come ulteriore linea di ricerca, per migliorare la QoE degli algoritmi inventati fino ad oggi, può essere interessante una loro implementazione ibrida.

Bibliografia

[1] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. Understanding the impact of video quality on user engagement. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 362–373. ACM, 2011.

[2] Visual Networking Index Cisco. Global mobile data traffic forecast update, 2015–2020 white paper. *Document ID*, 958959758, 2016.

[3] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural adaptive video streaming with pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 197–210. ACM, 2017.

[4] Matteo Gadaleta, Federico Chiariotti, Michele Rossi, and Andrea Zanella. D-dash: A deep q-learning framework for dash video streaming. *IEEE Transactions on Cognitive Communications and Networking*, 3(4):703–718, 2017.

[5] Cisco VNI Forecast et al. Cisco visual networking index: Global mobile data traffic forecast update 2009-2014. *Cisco Public Information*, 9, 2010.

[6] David Gourley and Brian Totty. *HTTP: the definitive guide*. " O'Reilly Media, Inc.", 2002.

[7] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[8] Davide Talon. Studio sperimentale di algoritmi di streaming video adattativi. Master's thesis, Facolta' di Ingegneria, Dipartimento di Ingegneria dell'Informazione, 2017.

[9] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[11] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. A control-theoretic approach for dynamic adaptive video streaming over http. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 325–338. ACM, 2015.

[12] Junchen Jiang, Vyas Sekar, and Hui Zhang. Improving fairness, efficiency, and stability in http-based adaptive video streaming with festive. *IEEE/ACM Transactions on Networking (TON)*, 22(1):326–340, 2014.

[13] Zhi Li, Xiaoqing Zhu, Joshua Gahm, Rong Pan, Hao Hu, Ali C. Begen, and David Oran. Probe and adapt: Rate adaptation for http video streaming at scale. *IEEE Journal on Selected Areas in Communications*, 32(4):719–733, 2014.

[14] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pages 1928–1937, 2016.

[15] Richard S. Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[16] Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[17] Johan De Vriendt, Danny De Vleeschauwer, and David Robinson. Model for estimating qoe of video delivered using http adaptive streaming. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pages 1288–1293. IEEE, 2013.

[18] Tobias Hoßfeld, Sebastian Egger, Raimund Schatz, Markus Fiedler, Kathrin Masuch, and Charlott Lorentzen. Initial delay vs. interruptions: Between the devil and the deep blue sea. In *Quality of Multimedia Experience (QoMEX), 2012 Fourth International Workshop on*, pages 1–6. IEEE, 2012.