Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

IMPLEMENTAZIONE E ANALISI DI ALGORITMI DINAMICI PER TRASMISSIONE MPEG-DASH SU CLIENT ANDROID

Relatore:
Chiar.mo Prof.
Luciano Bononi
Correlatore:
Dott. Luca Bedogni

Presentata da:
Stefano Mazza

Sessione II
Anno Accademico 2015/2016

Abstract

Attualmente, la maggior parte dei dati che transitano sulla rete appartiene a contenuti multimediali. Più nello specifico, è lo *Streaming Video* ad avere la predominanza nella condivisione di Internet; vista la crescita che tale servizio ha subìto negli ultimi anni, si sono susseguiti diversi studi volti allo sviluppo di tecniche e metodologie che potessero migliorarlo. Una di queste è sicuramente l'*Adaptive Video Streaming*, tecnica utilizzata per garantire all'utente una buona *Quality of Experience* (QoE) mediante l'utilizzo dei cosiddetti *algoritmi di rate adaptation*.

Il lavoro svolto in questi studi si è voluto concentrare su due filoni distinti, ma allo stesso tempo confrontabili: la prima parte della tesi riguarda lo sviluppo e l'analisi di alcuni algoritmi di rate adaptation per DASH, mentre la seconda è relativa all'implementazione di un nuovo algoritmo che li possa affiancare, migliorando la QoE nel monitorare lo stato della connessione.

Si è quindi dovuta implementare un'applicazione Android per lo streaming video, che fosse conforme allo standard MPEG-DASH e potesse fornire le informazioni di testing da utilizzare per le analisi.

La tesi è suddivisa in quattro capitoli: il primo introduce l'argomento e definisce la terminologia necessaria alla comprensione degli studi; il secondo descrive alcuni dei lavori correlati allo streaming adattivo e introduce i due filoni principali della tesi, ovvero gli algoritmi di rate adaptation e la proposta di algoritmo per la selezione dinamica del segmento; il terzo presenta l'app *SSDash*, utilizzata come mezzo per le analisi sperimentali; infine, il quarto ed ultimo capitolo mostra i risultati delle analisi e le corrispondenti valutazioni.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

1.1 Panorama attuale

L'incremento del traffico globale di Internet che ha caratterizzato gli ultimi anni è sotto gli occhi di tutti: abbiamo assistito ad una vera e propria "rivoluzione" dovuta alla costante crescita della quantità di dati caricati e richiesti sulla rete.

Ciò che ha portato avanti questa crescita è per lo più la condivisione di contenuti multimediali (audio e video). In questo campo ha guadagnato sempre più rilevanza lo *streaming video*; come si evince da [1], lo streaming video copre ad oggi il 55% delle interazioni con la rete da parte dei dispositivi mobili, ed è stata fatta una previsione che afferma la crescita di questa percentuale, fino al 75% nel 2020 (vedi fig. 1.1). Questa situazione sarebbe poco sostenibile se nel frattempo non fossero state fatte innovazioni tecnologiche, sia dal punto di vista hardware che dal punto di vista software. In questo studio vogliamo soffermarci maggiormente sulle seconde, in particolare sulle tecniche di distribuzione dei contenuti multimediali nello streaming video; tutti quei metodi cioè, che sono volti a fornire una migliore *Quality of Experience* all'utente.

In questo contesto, nutre grande importanza il cosiddetto *Adaptive Video Streaming*: si tratta di una tecnica che permette di cambiare, durante la

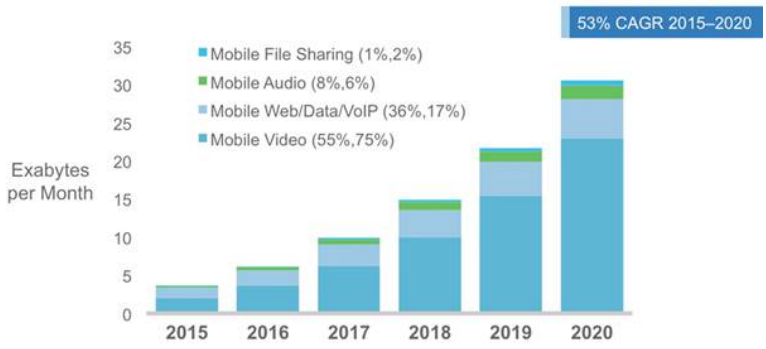


Figura 1.1: Previsioni relative alla crescita del traffico dei dati multimediali sulla rete fino al 2020 [1]

riproduzione, la qualità del video visualizzato monitorando lo stato della connessione, in modo da adattare ad esso la quantità di dati da scaricare. Questo metodo è utilizzato da diverse piattaforme, come YouTube, Hulu e Netflix.

Lo standard MPEG-DASH (o Dynamic Adaptive Streaming over HTTP), che sfrutteremo per questi studi, fa uso della suddetta tecnica; nel paragrafo 1.3.1 sono riportate le sue caratteristiche principali.

In sintesi, lo scopo principale di questa tesi è quello di creare un client Android che garantisca la trasmissione in streaming di contenuto multimediale e che rispetti i vincoli e le caratteristiche dello standard DASH. Tale client sarà usato come strumento per compiere importanti analisi su alcune tecniche di streaming e sulla nostra proposta di algoritmo, che presenteremo nel capitolo 2.

1.2 Streaming

Quello che è comunemente definito *streaming* indica in generale un flusso di dati audio/video trasmessi da una sorgente a una o più destinazioni tramite una rete. Solitamente viene effettuato un meccanismo di *bufferizzazione* per poter riprodurre i dati mano a mano che arrivano al richiedente.

1.2.1 Tipologie di streaming

Possiamo identificare principalmente due tipologie di streaming:

- Streaming on demand
- Streaming live

Streaming on demand

I contenuti audio/video sono memorizzati su un server dopo essere stati compressi: in questo modo è possibile alleggerire il carico dei pacchetti che transitano sulla rete. L'utente che vuole beneficiare di tale servizio può richiedere al server i dati in questione che verranno poi decompressi dal codec e riprodotti, a seguito di un particolare delay (questo ritardo serve a prevedere eventuali malfunzionamenti della rete).

Appartengono a questa categoria i flussi streaming di Windows Media Player, QuickTime, Adobe Flash Video (che è utilizzato da YouTube) e alcune emittenti televisive.

Streaming live

Simile alla tradizionale trasmissione radio o video in broadcast. Come sopra, i dati sono trasmessi utilizzando opportune compressioni che introducono nel flusso un ritardo di qualche secondo.

Sebbene a volte per trasmissioni via web si utilizzi il termine streaming come sinonimo di "diretta" è da chiarire che un contenuto in streaming possa essere live oppure no.

Questo tipo di servizio viene offerto anche da parte di diverse reti televisive, come la RAI.

1.2.2 Standard e Protocolli

Introduciamo ora un po' di terminologia, per chiarire bene su cosa si focalizza lo spazio d'azione di questi studi.

Standard

Uno *Standard* è un insieme di regole che un determinato sistema o prodotto deve necessariamente soddisfare a livello logico-funzionale. Tali norme sono fissate allo scopo di ottenere l'unificazione delle caratteristiche del sistema. In questo modo è garantita l'interoperabilità tra il software, e i produttori di tale servizio hanno maggiori libertà.

Vi sono due diverse categorie di standard:

Standard de jure standard approvato da apposite organizzazioni a seguito di un processo di analisi tecnica. La sua applicazione non è obbligatoria. Il modello ISO/OSI ne è un esempio.

Standard de facto modelli di riferimento che vengono comunemente chiamati "standard" a causa della loro diffusione. Non hanno quindi subìto un processo di standardizzazione e le organizzazioni non li considerano come tali. Il modello TCP/IP ne è un esempio.

Protocollo

Un *protocollo* è un insieme di regole formalmente descritte, definite al fine di favorire la comunicazione tra una o più entità. Per scambiarsi dati due agenti devono attenersi ad un protocollo predefinito e ben specificato.

I seguenti sono quelli più conosciuti ed utilizzati:

TCP protocollo di rete a pacchetto; si occupa del controllo di trasmissione, ovvero di rendere affidabile la comunicazione dati in rete tra mittente e destinatario.

UDP come il precedente è un protocollo di livello di trasporto a pacchetto. Solitamente è usato insieme al protocollo IP.

IP protocollo di rete a pacchetto.

Nel contesto dello streaming risulta necessario spostare l'attenzione su un livello più alto: infatti i protocolli appena presentati non sono altro che il

"nucleo" dell'infrastruttura di Internet, poichè appartengono ai livelli centrali dello standard.

In generale, ogni livello risolve una serie di problemi nella trasmissione dei dati, fornendo uno specifico servizio a quello successivo. Se vogliamo studiare il funzionamento dello streaming dobbiamo parlare dei protocolli relativi all'*Application Level*, quelli cioe più "vicini" all'interfaccia utente:

RTP Real-time Transport Protocol: protocollo del "livello applicazione" utilizzato per servizi di comunicazione in tempo reale su Internet. È basato, in particolar modo, sul protocollo UDP al livello di trasporto.

RTCP Real-time Transport Control Protocol: protocollo che fornisce statistiche sulla qualità del servizio di RTP e trasporta le informazioni riguardo ai partecipanti ad una sessione.

Si tratta di un protocollo *stateful*: stabilita la connessione con il server, quest'ultimo monitora lo stato del client fintanto che esso rimane collegato.

HTTP Hypertext Transfer Protocol: protocollo a livello applicativo utilizzato come principale sistema per la trasmissione di informazioni sul web. È di fatto il protocollo più diffuso e la sua logica si basa sul concetto client-server: il client manda una richiesta, il server fornisce una risposta.

Si tratta di un protocollo *stateless*: ogni richiesta del client è indipendente da quelle precedenti ed è conclusa al momento della ricezione dei dati.

HTTP utilizza il protocollo TCP al livello di trasporto.

Lo streaming con HTTP, in particolare nella sua forma più semplice (il *progressive download*), si è diffuso a vista d'occhio negli ultimi anni, nonostante alcuni precedenti studi preferissero l'utilizzo di UDP e RTP alle variazioni di *throughput* e ritardi di ritrasmissione del protocollo TCP [2].

Questo utilizzo massiccio di HTTP per lo streaming ha fatto sì che l'attuale infrastruttura di Internet, comprendente proxy, cache e CDN, potesse essere utilizzata senza problemi, a differenza della soluzione con protocollo RTP. Un altro beneficio derivante dalla scelta di HTTP è che il flusso dati può oltrepassare i firewall o i gateway di NAT.

1.3 Adaptive Video Streaming

Lo streaming video può fa sorgere alcune problematiche: una di queste è sicuramente il *buffering* (pause che possono verificarsi durante la riproduzione del video). Tale questione è particolarmente fastidiosa per chi sfrutta il servizio fornito dallo streaming: diversi studi mostrano infatti che un'interruzione nel video possa convincere l'utente a stopparne la riproduzione, mentre in generale è considerato accettabile un decremento alla qualità [3].

In questo contesto si inserisce quindi l'*Adaptive Video Streaming*: si tratta di una tecnica (basata su HTTP) per cui la qualità del video viene definita in modo dinamico mediante il controllo continuo della connessione.

A livello di programmazione, questo metodo trova espressione nei cosiddetti algoritmi di *rate adaptation*, volti al raggiungimento di alcuni benefici: riduzione (talvolta anche assenza) del buffering, riproduzione iniziale veloce e buona *Quality of Experience* indipendentemente dalla potenza della connessione [4].

Vi sono diversi standard che sviluppano l'adaptive video streaming, come Microsoft SmoothStreaming, Adobe ADS, Apple HLS e MPEG-DASH. I nostri studi saranno focalizzati su quest'ultimo, ma le nostre proposte sono generiche e adattabili ad ognuno di essi.

1.3.1 MPEG-DASH

Dynamic Adaptive Streaming over HTTP (DASH) è uno standard sviluppato da MPEG e 3GPP ed istituito nel Novembre 2011; di fatto nacque per risolvere i problemi principali relativi allo streaming video tradizionale.

Sostanzialmente, DASH è una tecnica di adaptive video streaming in cui il file multimediale richiesto dall'utente si trova partizionato in uno o più segmenti all'interno del server. Tali frammenti sono codificati in svariate qualità e dimensioni, in modo che il client possa scaricare quelli più appropriati in relazione alle condizioni della sua connessione.

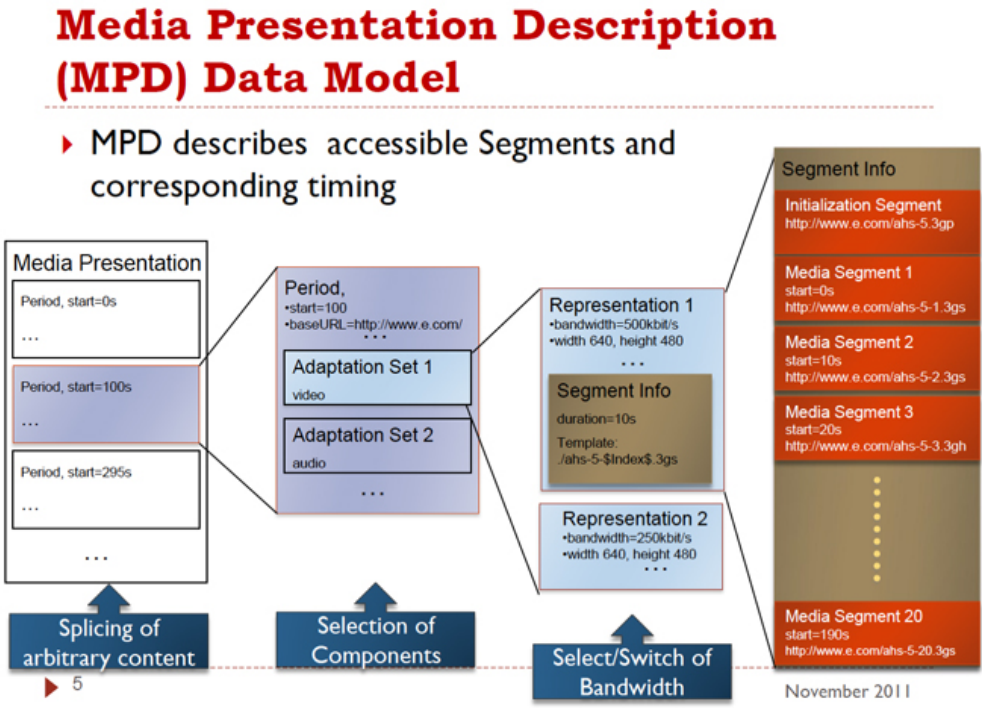


Figura 1.2: Gerarchizzazione del Media Presentation Description (MPD manifest) per DASH [14]

Media Presentation Description

Svolge un ruolo fondamentale il file *Media Presentation Description* (MPD): prima che lo streaming possa cominciare, il client ha l'obbligo di richiederlo al server; si tratta del manifest del video, che contiene alcune meta-informazioni, come ad esempio la lista delle possibili qualità in cui sono codificati i segmenti. Il manifest viene sottoposto a parsing in modo da estrapolarne le informazioni, dopodichè il client può cominciare a scaricare i pacchetti.

In generale l'insieme delle informazioni contenute all'interno dell'MPD può essere gerarchizzato nel modo seguente (fig. 1.2):

Media Presentation level contiene le informazioni generali del contenuto multimediale, come durata, minimo buffering iniziale ecc. ed è strutturato in una sequenza di uno o più periodi (*Period*) consecutivi.

Period level fornisce la descrizione di un singolo periodo mediante le seguenti informazioni: identificatore unico, istante iniziale, durata ecc. Ogni periodo è suddiviso a sua volta in uno o più segmenti (*Segment*).

Representation level descrive le caratteristiche delle *representations* (ovvero dei formati disponibili), quali bitrate, risoluzione e qualità.

Segment level descrive le locazioni da cui è possibile ottenere il segmento richiesto.

Il MPD è rappresentato concretamente da un file XML. Di seguito ne riportiamo un esempio [9]: si tratta delle prime righe del manifest relativo alla lunghezza di segmento di 2 secondi.

```
<MPD xmlns="urn:mpeg:dash:schema:mpd:2011" minBufferTime="PT1.500000S"
type="static" mediaPresentationDuration="PT0H9M56.46S"
profiles="urn:mpeg:dash:profile:isoff-on-demand:2011">
 <ProgramInformation moreInformationURL="http://gpac.sourceforge.net">
  <Title>
    dashed/BigBuckBunny_2s_onDemand_2014_05_09.mpd generated by GPAC
  </Title>
 </ProgramInformation>
 <Period duration="PT0H9M56.46S">
  <AdaptationSet segmentAlignment="true" group="1" maxWidth="480" maxHeight="360"
  maxFrameRate="24" par="4:3" subsegmentStartsWithSAP="1">
    <Representation id="320x240 46.0kbps" mimeType="video/mp4" codecs="avc1.42c00d"
    width="320" height="240" frameRate="24" sar="1:1"
    startWithSAP="1" bandwidth="45652">
```

```
    <BaseURL>bunny_45652bps/BigBuckBunny_2snonSeg.mp4</BaseURL>
    <SegmentBase indexRangeExact="true" indexRange="885-4504">
      <Initialization range="0-884"/>
    </SegmentBase>
  </Representation>
...
```

È bene precisare che lo standard DASH definisce solamente MPD e formato dei segmenti; tutte le altre specifiche sono indicate dallo sviluppatore, che ha quindi piena libertà nell'implementazione del software.

Presentiamo ora brevemente il funzionamento del Dynamic Adaptive Streaming.

Funzionamento di DASH

Supponiamo che il contenuto multimediale richiesto sia disponibile in tre diverse qualità, i cui relativi bitrate sono 5Mbps, 3Mbps e 0.5Mbps. In figura 1.3 è riportato l'esempio considerato: il client comincia lo streaming richiedendo il primo segmento a qualità massima, accorgendosi però di non poter sostenere un bitrate così alto; si verifica quindi uno switch di qualità, scegliendo di scaricare il pacchetto successivo in un formato di livello minore.

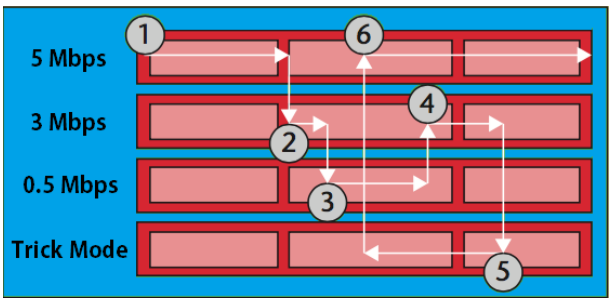


Figura 1.3: Esempio di Dynamic Adaptive Streaming

Dopo un ulteriore switch a 0.5Mbps notiamo che nel punto 4 l'algoritmo di rate adaptation scelga di aumentare la qualità, probabilmente a seguito di un miglioramento del segnale. Un particolare interessante è quello relativo

al punto 5: si tratta del *trick mode*, ossia quella modalità che viene attivata nel momento in cui l'utente agisce sulla timeline del playback, cercando di richiedere una porzione di video diversa da quella in riproduzione. L'algoritmo sceglie di far ripartire il video dalla posizione richiesta, ma alla qualità migliore.

Capitolo 2

Stato dell'Arte

La tecnica dell'*Adaptive Video Streaming* è stata presa di mira da molteplici studi negli ultimi anni. DASH non è infatti l'unica soluzione creata per implementare tale metodologia, ma ne esistono altre: in questo capitolo tratteremo i principali sistemi proprietari che lavorano sull'AVS, forniremo un'overview sugli algoritmi di rate adaptation e introdurremo la nostra proposta di algoritmo per la selezione dinamica della lunghezza del segmento.

2.1 Related Works

I più famosi sistemi proprietari sono i seguenti:

1. Microsoft Smooth Streaming;
2. Adobe HTTP Dynamic Streaming (ADS);
3. Apple HTTP Live Streaming (HLS);

Una delle caratteristiche che più li lega è, sorprendentemente, il fatto che utilizzino tutti una sorta di *manifest file*, seguendo così la stessa architettura con logica client-side per la scelta dei contenuti multimediali (partizionati in segmenti e memorizzati a differenti bitrate o risoluzioni). Si differenziano

però sul fatto che HLS si avvale di MPEG2-TS per quanto riguarda la codifica dei segmenti video, mentre gli altri due usano altri formati contenitori (ISOBMFF).

2.1.1 Esperimenti

Nell'articolo [8] sono riportati alcuni esperimenti effettuati per compiere studi sui tre sistemi proprietari.

Tutti i test sono stati eseguiti sul video Big Buck Bunny [9] e in condizioni di mobilità sull'autostrada A2 di Carinthia/Austria. Il contenuto multimediale (codificato secondo x264) risultava disponibile in 14 diversi bitrate (100, 200, 350, 500, 700, 900, 1100, 1300, 1600, 1900, 2300, 2800, 3400, e 4500 kbit/s); inoltre, come richiesto da Microsoft Smooth Streaming i segmenti del video erano considerati solamente di lunghezza pari a 2 secondi.

Come è possibile notare dai grafici dell'articolo [8], Microsoft Smooth Streaming si è comportato meglio degli altri per media del bitrate e per minor numero di switch nella qualità; HLS ha invece ottenuto una media minore e allo stesso tempo un buffer più lungo del necessario, dimostrandosi troppo conservativo. Infine ADS è da considerare il peggiore dei tre, in quanto non fornisce uno *smooth playback* (ossia una riproduzione fluida e intelligente), variando molto la risoluzione del video e facendo registrare diverse interruzioni.

2.2 Algoritmi di rate adaptation

Un *algoritmo di rate adaptation* non è altro che un algoritmo che viene eseguito ogni volta che il client deve scaricare un segmento del video. Esso ha il compito di sceglierne la qualità più congeniale in relazione allo stato della connessione, compiendo eventuali calcoli sui dati che prende in input e restituendo la risoluzione ricercata in output.

2.2.1 Terminologia

Lo stato della connessione può essere monitorato in vari modi e mediante diverse variabili: assumendo che (1) il client effettui il download dei segmenti in ordine cronologico e seriale e che (2) ciascuno di essi possa quindi essere numerato da un ID unico, andiamo a presentare i concetti che i nostri studi sfruttano per controllare il collegamento alla rete.

Throughput è un parametro molto importante nel contesto dello streaming video, perchè fornisce una misura (seppur approssimativa) della bontà della connessione: in generale, dato un segmento i del video, si calcola il throughput *T(i)* dividendo l'effettiva quantità di dati trasmessi (l'i-esimo pacchetto) nel tempo di consegna e il tempo di consegna stesso

$$T(i) = \frac{d_i}{t_f - t_0}$$

dove d_i denota la dimensione (in byte) del segmento scaricato, mentre t_0 e t_f indicano rispettivamente istante iniziale e finale di download del pacchetto.

È chiaro che tale variabile possa essere valutata solo per i pacchetti precedenti a quello da scaricare, perciò si parla normalmente di *last throughput*, per intendere il throughput relativo al pacchetto *i-1*.

Buffer level altro parametro rilevante: il *buffer* è una coda di segmenti video scaricati e in attesa di essere riprodotti; sostanzialmente si trova sottraendo all'istante finale dell'ultimo pacchetto scaricato l'istante corrente del playback. Il buffer level indica quindi tale misura, espressa in millisecondi (ms). Anche il buffer level dà una valutazione dello stato della connessione, poichè dalla sua lunghezza è possibile estrarre informazioni interessanti, in relazione al download dei pacchetti.

2.2.2 Tipologie di algoritmi

Ad oggi esistono tre tipi diversi di algoritmi di rate adaptation:

- throughput-based
- buffer-based
- ibridi

Algoritmi throughput-based

Effettuano la scelta della qualità del pacchetto successivo monitorando lo stato della connessione, mediante la misurazione del throughput. Gli esempi più semplici per questa categoria sono sicuramente l'*instantaneous algorithm* e lo *smoothed algorithm*.

Il primo [5] non fa altro che controllare il throughput relativo al pacchetto precedente e in base a questo sceglie il formato di qualità più alta possibile. È banalmente l'algoritmo di rate adaptation più facile da realizzare.

Il secondo [6][7] invece compie una media degli ultimi throughput registrati in modo da fornire un playback più fluido e non dare troppo peso ad inaccurate misurazioni dovute alla variabilità della connessione.

Algoritmi buffer-based

Effettuano la scelta della qualità del pacchetto successivo monitorando lo stato del buffer, solitamente in relazione ad una terna di costanti che vengono scelte a priori o a seguito di svariati test. Un esempio può essere FDash, che sarà descritto e valutato all'interno di questi studi (Cf. 4.2.3 per maggiori dettagli).

Algoritmi ibridi

Sintetizzano nella loro logica le peculiarità delle altre due categorie: per scegliere la qualità tengono sotto controllo sia il throughput che il buffer. Un esempio può essere mostrato dall'algoritmo descritto nell'articolo [10]: in sintesi prende in input le ultime misurazioni del throughput e restituisce in output due argomenti, ovvero la representation da selezionare e il minimo

livello del buffer (in secondi) che il playback dovrebbe avere alla partenza del download. Quest'ultimo parametro è utile per ridurre il livello del buffer cautamente, introducendo un delay tra download consecutivi solo nel caso in cui si sia già raggiunta la qualità massima o il throughput disponibile non consenta di sceglierla.

2.3 Selezione della lunghezza del segmento

La quasi totalità degli studi in letteratura si concentra unicamente sull'analisi degli algoritmi di rate adaptation con una premessa fondamentale: mantenere fissata la lunghezza del segmento da scaricare. Al più, in articoli come [13], si cerca di considerare tale parametro come eventuale variabile da scegliere in base alle condizioni della connessione, mostrando le differenze in termini di buffer level, qualità del playback e numero di richieste fatte al server.

Ciò che le nostre ricerche vogliono invece mostrare è la possibilità di eseguire la riproduzione del contenuto multimediale mediante un particolare algoritmo, *Dynamic SS*, che mira alla variazione della lunghezza dei pacchetti durante il playback [11].

In questa sezione vogliamo concentrarci sulle differenze che si possono evidenziare tra la scelta di una lunghezza di segmento fissata piuttosto che variabile, per poi presentare quelle che sono le nostre proposte.

2.3.1 Lunghezza fissata

Innanzitutto è bene specificare a quale contesto facciamo riferimento: i nostri studi si concentrano sul video Big Buck Bunny, che è disponibile in rete in sei differenti lunghezze di segmento (1, 2, 4, 6, 10, 15). Questo video è abbastanza particolare perchè, oltre ad essere preso in considerazione da molteplici articoli (è quindi una sorta di standard), la sua codifica non è comune: ciascun segmento è formato da una serie di fotogrammi, che possono essere di due tipi, *I-Frame* e *P-Frame*. Un I-Frame è una sorta di anteprima del

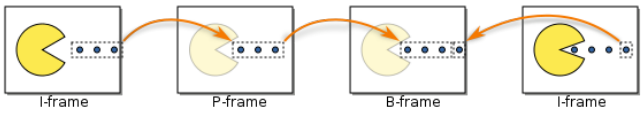


Figura 2.1: Una sequenza di fotogrammi video di codifica MPEG

contenuto multimediale presente all'interno del segmento, mentre i P-Frame sono anch'essi fotogrammi, ma invece di essere memorizzati come semplici jpeg, vengono in qualche modo "ricreati" a partire dall'I-Frame iniziale, in conformità con frame rate e qualità del video. In particolare, un pacchetto comincia sempre con un I-Frame, per poi proseguire nella sua codifica con una sequenza di P-Frame (vedi fig. 2.1).

Questo sistema è molto utile per diversi motivi: essendo i P-Frame calcolati in base ai cambiamenti da effettuare sui pixel dell'I-Frame iniziale, i segmenti del video saranno

- autosufficienti: nel caso in cui la qualità del video cambi durante la riproduzione, è possibile proseguire nel playback senza grossi sforzi in termini di ulteriore download di dati [11] (se l'utente sposta il cursore del playback in una specifica porzione del video, la riproduzione è capace di proseguire dal punto richiesto, senza scaricare i precedenti pacchetti);

- particolarmente leggeri: il fatto che i pixel vengano ricalcolati ogni volta è decisamente utile qualora il video non faccia uso di frequenti cambiamenti di scena; nel caso di Big Buck Bunny infatti non ve ne sono, quindi il contenuto multimediale è abbastanza leggero.

Secondo quanto detto possiamo affermare che, la principale differenza tra segmenti di lunghezza maggiore e minore sta nel fatto che, a parità di durata del video, i primi contengono meno informazioni dei secondi (hanno un bitrate minore), risultando così ancora più leggeri.

Inoltre, a parità di porzione del video, si verifica un trade-off: se da una parte i pacchetti di lunghezza maggiore sono meno utili in presenza di

rapidi cambiamenti allo stato della connessione (è necessario più tempo nel download), dall'altra i pacchetti di lunghezza minore devono essere scaricati in quantità maggiore, instaurando così un ingente numero di connessioni al server. Infatti MPEG-DASH effettua il download dei segmenti mediante HTTP GET, che a sua volta sfrutta TCP; quest'ultimo impone che vi sia un *three way handshake* all'inizio di ogni interazione client-server, per poter avere la conferma da ambo le parti ed inizializzare la connessione. Per questi motivi, il "tempo di recupero" del segmento è dato dalla somma del tempo di three way handshake e il tempo di effettivo download del pacchetto [11]. In conclusione quindi, al diminuire della lunghezza di segmento cresce l'overhead introdotto dalle richieste GET.

2.3.2 Lunghezza variabile

Le considerazioni fatte nel precedente paragrafo (2.3.1) ci fanno capire quanto sia preferibile scegliere segmenti di lunghezza minore nel caso in cui il buffer sia "scarso", e segmenti più grandi per ottimizzare invece l'utilizzo della banda [11]. Questi due concetti fanno riferimento ad un particolare parametro della connessione, ovvero la sua stabilità. Più una connessione è stabile, maggiore sarà la necessità di utilizzare al meglio la banda; più la connessione è instabile (fa registrare diversi cali e picchi nelle misurazioni del throughput), più si dovrà cercare di mantenere il buffer pieno, agendo in modo prudente nel download dei pacchetti (segmenti più piccoli permettono di aumentare velocemente il buffer e allo stesso tempo di chiamare più frequentemente l'algoritmo di rate adaptation, che potrà quindi ridurre la qualità velocemente se necessario).

L'utilizzo di segmenti di lunghezza variabile introduce però alcune problematiche, prima fra tutte la questione dell'*overlapping*.

Overlapping

Questo problema nasce dal fatto che, essendo i pacchetti di misure diverse, uno stesso ID corrisponde a intervalli del video differenti. Per overlapping

si intende quindi la sovrapposizione che scaturisce da questa differenza: assumiamo, ad esempio, che il video sia codificato in due lunghezze diverse - 1s e 10s - e che ad un certo istante del playback si abbia SS=1s e che il buffer sia di 9s; se la connessione si stabilizzasse e decidesse quindi di cambiare in SS=10s, si verificherebbe una sovrapposizione di 9 secondi, tra il segmento da scaricare e l'intero buffer [11].

È evidente quindi che, in tali condizioni, l'overlapping sfrutti troppo male la banda, scaricando pacchetti che erano in realtà già presenti nel playback.

Nella nostra proposta di algoritmo (Dynamic SS), l'overlapping è preso in questione, trovando una formula che lo risolva (Cfr. 4.2.4).

Capitolo 3

L'applicazione SSDash

In questo capitolo tratteremo le caratteristiche principali dell'applicazione SSDash, strumento utilizzato per gli studi compiuti all'interno della tesi.

L'applicazione è stata sviluppata per dispositivi Android, famosa e diffusa piattaforma prodotta da Google e basata su kernel Linux.

Tipicamente le applicazioni per Android sono implementate utilizzando Java (mentre per la gestione dell'interfaccia grafica si usa XML), ma esistono alcune alternative, come ad esempio la possibilità di programmare in C/C++ compilando poi in codice nativo. Nella maggior parte dei casi, rimane conveniente utilizzare Java, poichè conforme alla concezione iniziale di Android.

3.1 Caratteristiche tecniche

SSDash è implementata in Java e sfrutta *Exoplayer* e *GraphView*, librerie fornite da Android. La prima (Cfr. 3.3) è stata strettamente necessaria allo sviluppo delle analisi di questi studi, mentre la seconda, libreria per la creazione di grafici *a run-time*, si è più che altro rivelata un'opzione interessante, che ha permesso un apprendimento veloce di Exoplayer nei primi sviluppi degli studi.

L'applicazione però non è stata creata come pacchetto unico, poichè relativa e funzionante per due differenti tematiche (implementazione degli algoritmi di rate adaptation e di Dynamic SS). Tuttavia è possibile considerarla come singolo prodotto realizzato in due versioni diverse (vedi sez. 3.2).

3.1.1 Permessi richiesti

SSDash necessita di alcuni permessi:

- Accesso ad Internet (connessione Wi-Fi o rete dati):
  *android.permission.*INTERNET
- Accesso allo stato della connessione:
  *android.permission.*ACCESS_NETWORK_STATE
  *android.permission.*ACCESS_WIFI_STATE
- Lettura su memoria esterna:
  *android.permission.*READ_EXTERNAL_STORAGE
- Scrittura su memoria esterna:
  *android.permission.*WRITE_EXTERNAL_STORAGE

Tipicamente tali permessi vengono definiti e richiesti nell'*Android Manifest*, che in modo statico li concede all'utente. Invece, per versioni con SDK superiore a 23, il tutto avviene dinamicamente, con la conferma da parte dell'utente stesso: mediante un particolare *dialog*, Android dà all'utente la possibilità di richiedere esplicitamente i permessi per accedere alle impostazioni di sistema.

3.2 Esecuzione dell'app

Come affermato in precedenza, l'applicazione va considerata in due differenti versioni, quella relativa allo sviluppo degli algoritmi di rate adaptation e quella riguardante l'implementazione della nostra proposta, Dynamic SS.

In ognuno dei due casi, SSDash è suddivisa in due activity: *MainActivity* e *PlayerActivity*.

La prima è l'activity principale e mostra un menù che varia in base alla versione (vedi fig. 3.1): nella prima versione l'activity contiene una semplice lista in cui è possibile selezionare, per ognuno dei quattro video disponibili, la lunghezza del segmento desiderata, eseguendo automaticamente la run del contenuto multimediale secondo quanto scelto; analogamente, per la seconda versione è presente un menù a lista, contenente per ogni video una sola istanza. Il menù a tendina presente in entrambe le versioni permette la selezione dell'algoritmo di rate adaptation da utilizzare, con la differenza che nella seconda versione esso è implementato insieme a Dynamic SS.

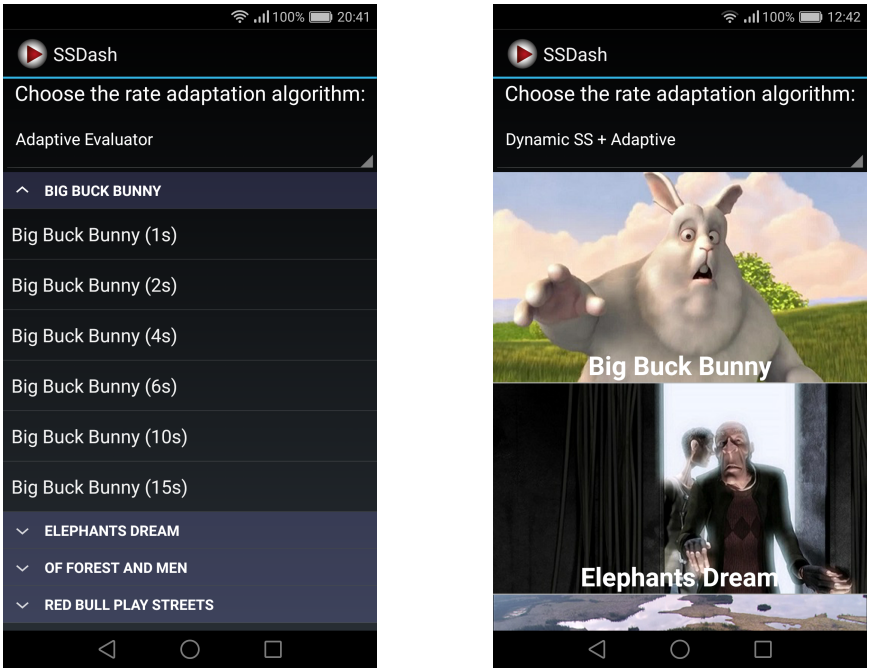


Figura 3.1: *MainActivity*: l'activity principale mostrata nelle due differenti versioni di SSDash

Nel momento in cui l'utente seleziona il video da riprodurre, viene chiamata la seconda activity, ovvero PlayerActivity, che è raffigurata dall'immagine 3.2. PlayerActivity funge da *player* del contenuto multimediale richiesto ed è stata pensata per fornire al programmatore il supporto migliore possi-

bile in termini di debug. A livello grafico l'activity è organizzata in tre parti: l'intero schermo è utilizzato per la riproduzione del filmato, ma in modalità *portrait* il riquadro che ospita il video viene ridotto come in figura 3.2; la parte inferiore dell'activity riguarda la gestione del playback (timeline e pulsanti *pause*, *rewind* e *fastforward*); la parte superiore dell'activity è relativa al debug ed è a sua volta suddivisa in due aree. L'area di debug che si trova più in alto raccoglie le informazioni principali (millisecondi attuali del playback, bitrate del video, last throughput, codec ecc.) che consentono allo sviluppatore un immediato riscontro di ciò che succede; l'area di debug inferiore è invece adibita alla presenza di pulsanti (visibili e non). Il pulsante Graph è stato introdotto per mostrare all'utente un grafico che in tempo reale si aggiorna in base al formato video corrente (per realizzarlo è stata utilizzata la libreria GraphView di Android).

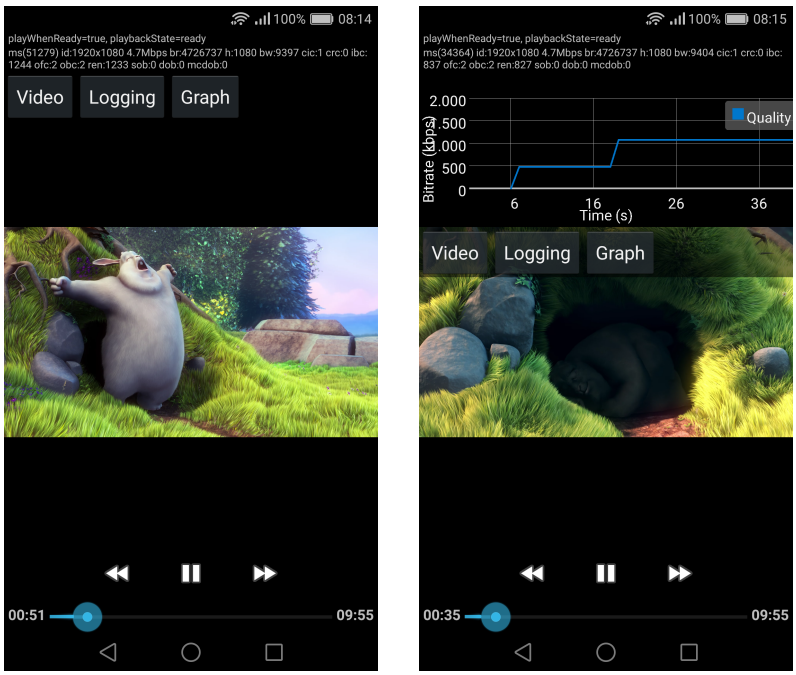


Figura 3.2: *PlayerActivity*: l'activity per la riproduzione del contenuto multimediale con aree di debug

Per le analisi effettuate in questi studi, è stato necessario che PlayerActivity fornisse un ulteriore e più preciso strumento di debug: il *log-file*. Ad

ogni riproduzione del video, SSDash salva nella memoria del dispositivo un file .csv in cui sono memorizzate tutte le informazioni raccolte per ogni pacchetto scaricato. Nella tabella 3.1 ne possiamo trovare un esempio, creato con una run di Big Buck Bunny a lunghezza di segmento pari a 2s: il log-file è organizzato in righe, ciascuna delle quali corrisponde ad un singolo segmento; per ogni riga sono presenti nell'ordine connessione (Wi-Fi/3G-4G), nome del video, lunghezza del pacchetto, id del pacchetto, id del formato video, bitrate del segmento, *instantaneous throughput* (throughput relativo a tale pacchetto), *estimated throughput* (throughput stimato per il segmento successivo), livello del buffer, peso del segmento e tempo di download. A partire da file di questo genere, è stata possibile la creazione veloce di grafici su cui basare le nostre valutazioni (Cfr. cap. 4).

Tabella 3.1: Esempio di log-file per una run di Big Buck Bunny con SS=2s

Connection	Video	Length	ID	Quality	Bitrate	I. throughput	E. throughput	Buffer level	Size (Byte)	DWNLD time
Wi-Fi	BigBuckBunny	2s	0	1280x720 791.0kbps	791182	2939977	800000	0	192201	523
Wi-Fi	BigBuckBunny	2s	1	1280x720 791.0kbps	791182	3050005	2204982	1653	146019	383
Wi-Fi	BigBuckBunny	2s	2	1280x720 791.0kbps	791182	2987502	2287503	3204	219208	587
Wi-Fi	BigBuckBunny	2s	3	1280x720 791.0kbps	791182	2987502	2240626	4561	213009	574
Wi-Fi	BigBuckBunny	2s	4	1280x720 791.0kbps	791182	2987502	2240626	5928	216005	577
Wi-Fi	BigBuckBunny	2s	5	1280x720 791.0kbps	791182	2994675	2240626	7290	218237	583
Wi-Fi	BigBuckBunny	2s	6	1280x720 791.0kbps	791182	2994675	2240626	8652	193075	514
Wi-Fi	BigBuckBunny	2s	7	1920x1080 2.1Mbps	2133691	2994675	2240626	10004	661058	1822
Wi-Fi	BigBuckBunny	2s	8	1920x1080 2.1Mbps	2133691	2902559	2240626	10088	484329	1348
Wi-Fi	BigBuckBunny	2s	9	1920x1080 2.1Mbps	2133691	2902559	2176919	10698	423052	1162

3.3 Exoplayer

Exoplayer è una libreria open-source di circa 16000 righe di codice sviluppata da Google. Si pone come alternativa alle API del *Media Player* di Android, fornendo in più alcuni benefici particolari, quali DASH e Smooth-Streaming. Garantisce la riproduzione di flussi multimediali che siano online piuttosto che fisicamente presenti sul device che ne fa uso.

Una delle peculiarità principali dei client con Exoplayer è quella di essere facilmente personalizzabili ed estendibili: utilizzando il solo Media Player di Android, lo sviluppatore non ha la possibilità, per esempio, di cambiare le

politiche di buffering o aggiungere una cache alla gestione dei dati scaricati; al contrario, importando le librerie di Exoplayer nella propria app, il programmatore può decidere di implementare a suo piacimento le funzionalità dei livelli più bassi, come *networking*, *buffering* e *sample extraction*.

Exoplayer è stata scelta proprio per questi motivi, insieme al fatto che tale libreria è supportata da gran parte dei dispositivi Android (anche se essendo basata sulle API *MediaCodec* non è compatibile con le versioni precedenti a alla 4.1, ossia API livello 16).

3.3.1 Library Overview

Exoplayer non è un singolo oggetto Java, bensì un insieme di moduli, ciascuno dei quali contribuisce nel suo piccolo al funzionamento della libreria e del player. In figura 3.3 ne possiamo vedere una rappresentazione molto dettagliata, relativa all'implementazione secondo lo standard DASH.

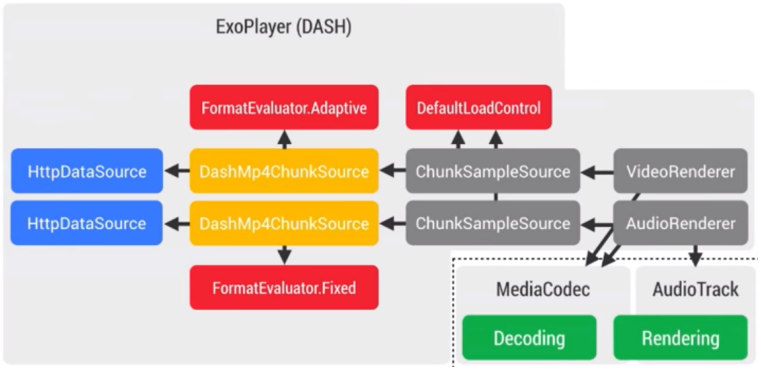


Figura 3.3: Exoplayer 2.x. overview [17]

Il nucleo centrale è ricoperto dal modulo *Exoplayer*, che ha il compito di mantenere il controllo dello stato del playback. La libreria contiene degli oggetti chiamati "renderers", che vengono utilizzati per effettuare il *rendering* (ovvero la resa finale) dei contenuti audio e video (*AudioRenderer* e *VideoRenderer*). Le istanze del modulo *ChunkSampleSource* vengono utilizzate per fornire i contenuti multimediali ai renderers. Ciascuna di esse richiede

un componente *ChunkSource*, che ha il compito di rilasciare le porzioni richieste del contenuto multimediale; nella fattispecie tale compito sarà rivestito dal modulo *DashMp4ChunkSource*. Infine, a loro volta, ogni componente ChunkSource richiede un *DataSource*, che si preoccupa invece di effettuare le richieste HTTP per il download effettivo dei dati da scaricare: Exoplayer utilizza la classe *HTTPDataSource*.

Come si può notare dalla figura, audio e video sono gestite separatamente: è per questo motivo che due istanze della stessa classe sono utilizzate contemporaneamente.

Il modulo *DefaultLoadControl* è incaricato di gestire il buffering dei *media chunk*. Per quanto riguarda invece l'implementazione degli algoritmi di rate adaptation, Exoplayer fornisce al programmatore un'interfaccia chiamata *FormatEvaluator*: la sua struttura è congeniale a quanto richiesto dagli sviluppatori, poichè, mediante l'utilizzo di un'interfaccia, ciascun algoritmo può essere implementato indipendentemente dalla presenza di altri, ed il client ne può contenere più di uno. FormatEvaluator non fa altro che effettuare una scelta tra i formati disponibili, prima che il *media chunk* sia richiesto. Nell'esempio in figura si utilizzano un *FixedEvaluator* per il contenuto audio (la qualità rimane sempre la stessa) e *AdaptiveEvaluator* per il contenuto video (la qualità varia in modo "adattivo").

All'interno di questi studi si è lavorato principalmente nei moduli *DashChunkSource* e *FormatEvaluator*, rispettivamente per l'implementazione di Dynamic SS e degli algoritmi di rate adaptation. Nel primo caso, a livello implementativo, si è deciso di scaricare i MPD delle 6 lunghezze di segmento disponibili alla partenza del video, utilizzando inizialmente quello da 1s. A seguito del parsing iniziale sui manifest, si è scelto di memorizzare le informazioni ottenute in una matrice, che contenesse i formati video disponibili per ogni lunghezza di segmento.

Capitolo 4

Performance Evaluation

L'analisi dei dati compiuta mediante l'utilizzo di SSDash si è voluta concentrare su due contesti differenti: il primo riguarda l'implementazione e la valutazione di algoritmi di rate adaptation, il secondo è relativo alla nostra proposta di algoritmo (Dynamic SS).

In questa sezione vogliamo fornire un quadro generale delle metriche su cui ci baseremo nelle valutazioni, descrivere gli algoritmi implementati e mostrare i risultati delle analisi sottoforma di grafici, presentando successivamente quelle che sono le nostre conclusioni.

4.1 Problematiche e criteri di valutazione

In relazione a quanto detto nell'articolo [10] possiamo affermare che, per un algoritmo di rate adaptation, gli obiettivi principali da raggiungere siano i seguenti:

1. massimizzare la qualità media dello *stream*, minimizzando allo stesso tempo il numero di *shift* (fluttuazioni) nella qualità;

2. minimizzare il numero di interruzioni del plaback, eventualmente monitorando il livello del buffer;

3. minimizzare il *delay* tra la richiesta dell'utente e l'inizio della riproduzione;

4. stimare in modo opportuno le dinamiche del throughput disponibile.

Tabella 4.1: Tabella contenente il bitrate delle representation disponibili per ogni diversa lunghezza del segmento di codifica. Ad ogni livello è associato un particolare *Quality Index* (QI)

QI	1s	2s	4s	6s	10s	15s
0	46980	45652	45226	45514	45373	45351
1	91917	89283	88783	89036	88482	88563
2	135410	131087	128503	128256	127412	127052
3	182366	178351	177437	176827	176780	176031
4	226106	221600	217761	217651	216536	215976
5	270316	262537	255865	255169	252988	252582
6	352546	334349	323047	320564	317328	315047
7	424520	396126	378355	373935	368912	365887
8	537825	522286	509091	506300	503270	500530
9	620705	595491	577751	573252	568500	565835
10	808057	791182	782553	779796	771359	769540
11	1071529	1032682	1008699	1006743	987061	982168
12	1312787	1244778	1207152	1201426	1174238	1165885
13	1662809	1546902	1473801	1461530	1431232	1419205
14	2234145	2133691	2087347	2078678	2070985	2061491
15	2617284	2484135	2409742	2395950	2384387	2372196
16	3305118	3078587	2944291	2919314	2884382	2863389
17	3841983	3526922	3340509	3297198	3245900	3222578
18	4242923	3840360	3613836	3557500	3493765	3464259
19	4726737	4219897	3936261	3858484	3792491	3748236

Gli algoritmi saranno quindi valutati in base a quanto si avvicineranno al raggiungimento di tali *goal*; in particolare, per ogni algoritmo saranno prese in considerazione alcune variabili caratterizzanti: bitrate del video, livello del buffer, numero di interruzioni del playback e quantità di switch nel formato effettuati.

Sorge però una questione di fondamentale importanza nella scelta di tali metriche. La tabella 4.1, contenente i valori del bitrate per tutte le representation possibili, associa ad ognuno di essi un particolare *Quality Index* (QI),

ovvero un indice che determina il livello di qualità a cui il bitrate appartiene. Il QI è compreso tra 0 e 19, dove 0 indica la qualità peggiore, mentre 19 la migliore. Come si può notare dalla tabella, la quantità di dati varia (a parità di QI) in base alla lunghezza del segmento (vedi paragrafo 2.3.1 per maggiori dettagli). Questo fatto va tenuto decisamente in considerazione, in quanto valutazioni che si riferiscano solo all'*average* del bitrate non risulterebbero del tutto corrette, soprattutto se si stesse analizzando la nostra proposta di algoritmo (che varia la lunghezza del segmento).

Per questi motivi faremo spesso riferimento al QI, effettuando le analisi conclusive su tale parametro.

4.2 Descrizione degli algoritmi

In questa sezione vogliamo dare una breve, ma dettagliata descrizione degli algoritmi analizzati, in modo da poterli valutare anche sulla base delle loro differenze progettuali. Per maggiori informazioni consultare l'appendice A, in cui i seguenti algoritmi sono riportati in pseudocodice.

4.2.1 Adaptive Evaluator

Si tratta dell'algoritmo già presente all'interno di Exoplayer.

Il suo funzionamento è molto simile all'*instantaneous algorithm*: monitora anch'esso il throughput relativo al pacchetto scaricato precedentemente, ma decrementa tale valore in relazione ad una costante, in modo da prevedere eventuali malfunzionamenti nella misurazione del throughput. Inoltre gestisce la scelta del formato video controllando anche il buffer, secondo due casi:

1. se il formato ideale sarebbe minore di quello corrente, ma il buffer non è abbastanza grande si preferisce mantenere il corrente;

2. se il formato ideale sarebbe maggiore di quello corrente vi sono due ulteriori sottocasi:

- se il buffer non è abbastanza grande per aumentare la qualità si decide di mantenere il corrente;
- se il buffer è abbastanza grande significa che si sta cambiando da uno *stream* di basso livello ad uno di qualità superiore; si vuole quindi eliminare tutti i segmenti non HD presenti nel buffer.

4.2.2 Wiser Smoothed Evaluator

Questo algoritmo è stato implementato ispirandosi alla letteratura [16]. Può essere visto come una fusione tra l'*instantaneous algorithm* e lo *smoothed algorithm* [6][7]: in generale non fa altro che valutare quale delle due tecniche sia meglio utilizzare, ovvero se far pesare di più le misurazioni recenti piuttosto che le stime passate, e viceversa.

Più precisamente, vengono considerate 2 variabili (entrambe comprese tra 0 e 1), denominate *normalized throughput deviation* e *smoothed parameter* (nell'appendice A sono riportate come p e δ).

La prima ci dà una stima di quanto sia variabile il throughput della connessione: se p tende a 1 significa che si stanno verificando forti cambiamenti, quindi si vuole una reazione simultanea; al contrario, se tende a 0, la soluzione è quella di usare un'approccio "smooth", effettuando una media tra le precedenti misurazioni.

La seconda ci dice invece se la stima del throughput debba dipendere maggiormente dalla misurazione dell'ultimo segmento (se tende a 1) o dalla media di cui sopra. È stato scelto di inizializzare questa variabile con il valore di 0.5, in modo da non dare, almeno in partenza, una preferenza per la tecnica da usare.

4.2.3 FDash Evaluator

Anche *FDash* è stato implementato ispirandosi alla letteratura [15]. Si tratta di un algoritmo molto particolare, in quanto utilizza un modello denominato *fuzzy rule*: il nucleo principale di questo algoritmo è caratterizzato

da una serie di `if` a cascata che, valutando lo stato del buffer riesce a scegliere il formato video migliore possibile. In particolare vengono definite tre costanti e tre variabili booleane: rispettivamente *SHORT*, *CLOSE*, *LONG* e *falling*, *steady*, *rising*. Le prime tre riguardano il livello attuale del buffer, mentre le seconde sono relative alla variazione che tale livello ha avuto dalla precedente misurazione. La *fuzzy logic* lavora cercando, tra le scelte degli `if`, il caso corrispondente a quello attuale, effettuando confronti tra il livello del buffer e le tre costanti e controllando quale dei tre booleani sia vero al momento della chiamata dell'algoritmo (steady, falling e rising indicano lo stato del buffer: se veri, rispettivamente intendono che il buffer è stabile, sta calando, sta crescendo).

Come spiegheremo più avanti (vedi par. 4.3.3), la scelta dei parametri SHORT, CLOSE e LONG è di fondamentale importanza per le prestazioni dell'algoritmo. In questi studi non si è voluta fare un'analisi troppo accurata per determinare la terna migliore, ma si sono svolti alcuni test, a seguito dei quali la terna 10s-40s-50s è risultata la meno deficitaria. Nelle analisi si considereranno quindi tali costanti.

4.2.4 Dynamic SS

Dynamic SS è la nostra proposta di algoritmo: le teorie a suo supporto si basano sul fatto che controllando la stabilità della connessione e variando la lunghezza del segmento sia possibile fornire all'utente una migliore *QoE*, rispetto a quella offerta dalla selezione statica della lunghezza del segmento. A differenza delle tre precedenti descrizioni, questa non riguarda un algoritmo di rate adaptation, bensì un algoritmo che possa accompagnarlo. Dynamic SS si concentra infatti sullo stato della connessione, utilizzandolo per decidere se cambiare o meno SS (e non sulla qualità dei segmenti): se la connessione è stabile la lunghezza può essere aumentata, al contrario deve essere decrementata.

In particolare, Dynamic tiene traccia degli ultimi M throughput registrati e ne effettua una valutazione pesata in relazione alla lontananza della misu-

razione e alla distanza di ogni elemento dalla media. Come output di questa rielaborazione dei dati otteniamo la cosiddetta "changing willingness" (indicata con p nell'appendice A.4), ossia una misura di quanto la connessione necessiti l'aumento o la diminuzione dell'SS. Utilizzando tale valore come probabilità, l'algoritmo può scegliere la nuova lunghezza del segmento. A questo proposito è bene precisare che, in sede di implementazione dell'algoritmo, si sono riscontrati alcuni problemi: effettuando le operazioni appena descritte in linea con ciò che viene specificato in [11], la facilità con cui Dynamic cambia SS varia in base alla lunghezza attuale. Questo è dovuto principalmente al fatto che le differenze tra SS consecutivi (considerando le lunghezze disponibili) non è costante (ad esempio tra 10 e 6 secondi c'è un gap maggiore rispetto a quello presente tra 1 e 2 secondi). Per risolvere questo problema si è utilizzato il vettore fittizio Θ, definito nel modo seguente

$$\Theta = [\theta_1, ..., \theta_N]$$

dove N=6 (rappresenta il numero di SS disponibili) e $\Theta_i = i$. In questo modo creiamo un'associazione tra 1,2,3,4,5,6 e 1s,2s,4s,6s,10s,15s, facendo lavorare l'algoritmo su differenze costanti.

Per quanto riguarda il problema dell'overlapping, discusso nel paragrafo 2.3.2, Dynamic SS sceglie come soluzione quella di stimare la lunghezza della sovrapposizione e decidere di effettuare comunque il download del segmento (eliminando i pacchetti già scaricati) oppure impedire il cambio di SS. Per calcolare l'overlapping δ si usa la seguente formula:

$$\delta = \frac{\sum_{h=start}^{start+SS'} B(h)}{SS'}$$

dove *start* e *SS'* sono l'istante di inizio e la lunghezza del nuovo eventuale segmento, mentre B è definita come segue

$$B(i) = \begin{cases} 1 & \text{se } i \in buffer \wedge i > playback \\ 0 & \text{se } i < playback \end{cases}$$

4.3 Analisi dei dati

Per effettuare il testing e valutare gli algoritmi si è deciso di utilizzare il video Big Buck Bunny, memorizzato sul server di *Itec* e disponibile online [9]. Il numero massimo di representation è 20; esso può variare (diminuire) in base al dispositivo che fa uso del client DASH (non tutte le risoluzioni sono supportate da ciascun device). Considerando un determinato dispositivo e una determinata lunghezza del segmento, saranno quindi disponibili livelli di bitrate pari in numero alle representation supportate.

Gli esperimenti sono stati fatti volutamente in condizioni diverse fra loro, in modo da avere un quadro generale e più preciso possibile dei dati su cui andare a lavorare. Nel nostro caso tutti i test a cui ci riferiremo in questo capitolo sono stati svolti installando SSDash su un Huawei P9 Lite; a differenza dell'altro device a disposizione (Samsung S4-mini), il P9 Lite ha risoluzione Full-HD, perciò tutte le representation saranno supportate.

Suddividiamo l'analisi dei dati in due parti, in base al movimento del device durante i test: *analisi statica* e *analisi dinamica*.

4.3.1 Analisi statica

Per quanto riguarda l'analisi compiuta mantenendo il dispositivo fermo e connesso alla Wi-Fi, si sono prese in considerazione 3 differenti velocità di connessione: *1Mbps*, *3Mbps* e *FreeMbps* (quest'ultima starebbe a significare che non sono state introdotte limitazioni alla banda; si è quindi utilizzata una connessione Wi-Fi di circa 9/10 Mbps). Per limitare la velocità di connessione si è utilizzato un Raspberry Pi versione 2, uno script per la creazione di access point [12] ed infine il comando `tc` per bash.

Per ognuno dei tre algoritmi di rate adaptation e per ogni lunghezza del segmento, sono state effettuate 3 *run* (riproduzioni del video), in modo da limitare il più possibile la randomicità dei dati e delle misurazioni. I dati raccolti da questi test sono stati utilizzati per due diverse tipologie di

Tabella 4.2: Tabella che mostra la media dei secondi di pausa che ciascun algoritmo (A-Adaptive, W-Wiser Smoothed, F-FDash) ha ottenuto nelle 3 run, per ogni velocità e lunghezza di segmento

	A	W	F	A	W	F	A	W	F	A	W	F	A	W	F	A	W	F
	1 s			2 s			4 s			6 s			10 s			15 s		
Free Mbps	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3 Mbps	0	5	0	0	6	0	0	0	0	0	0	0	0	0	0	0	0	0
1 Mbps	0	6	0	0	6	0	0	6	0	0	0	0	0	0	6	0	0	35

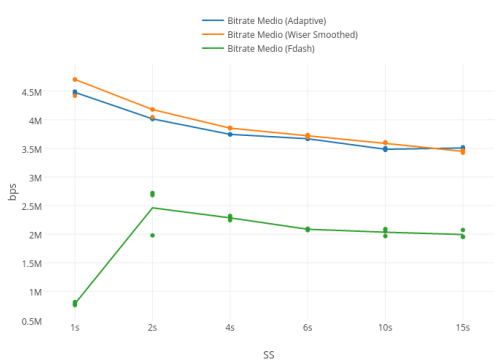
analisi: il confronto tra lunghezze di segmento diverse e l'indagine in vista delle differenze tra l'assenza e la presenza di Dynamic SS.

Test sulle lunghezze del segmento

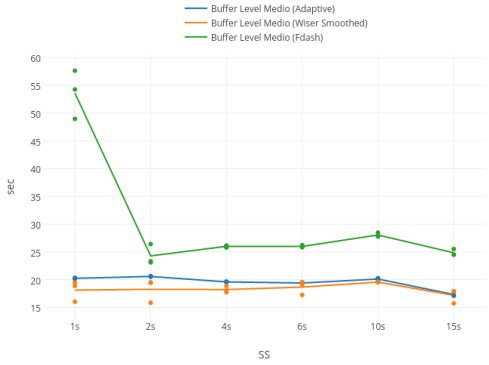
I grafici sottostanti (1) e la tabella 4.2 (2) mostrano i risultati dei primi esperimenti ed in particolare raccolgono per ciascuna velocità (1) la media di bitrate e buffer level e (2) i secondi di buffering accaduti durante la riproduzione (il tutto è stato valutato intersecando le misurazioni delle 3 run). Come spiegato nella sezione 4.1, è stato necessario effettuare una valutazione che prevedesse anche il calcolo del QI; i relativi grafici sono le figure 4.2, 4.3 e 4.4.

Come si può vedere dai grafici 4.1(a) e 4.1(b), in ottime condizioni FDash si comporta abbastanza male, mantenendo una media del bitrate molto bassa per ciascuna lunghezza del segmento e senza raggiungere mai i livelli più alti di qualità. Si può evincere quindi valutando anche il buffer, che FDash sia troppo conservativo, mentre gli altri due algoritmi si comportino decisamente meglio, posto che per FreeMbps non si verifichino pause (Cfr. tab. 4.2). Il grafico 4.2 è la conferma di quanto appena affermato: al variare dell'SS, Adaptive e Wiser mantengono massimo e costante il QI, al contrario di FDash.

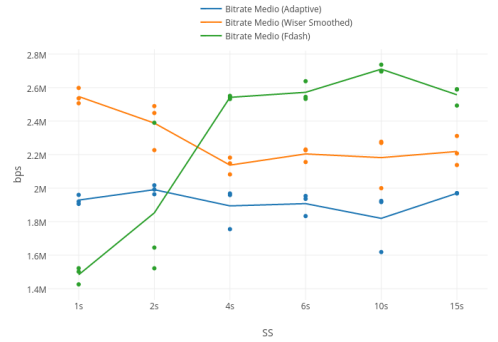
Per quanto riguarda invece la limitazione della connessione a 3 Mbps (grafici 4.1(c) e 4.1(d)), è utile suddividere la valutazione in due sottocasi, in base alla lunghezza del segmento (SS):



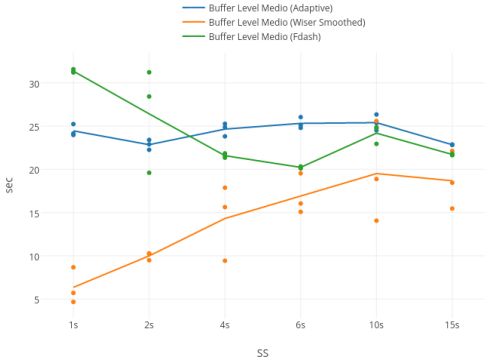
(a) FreeMbps bitrate



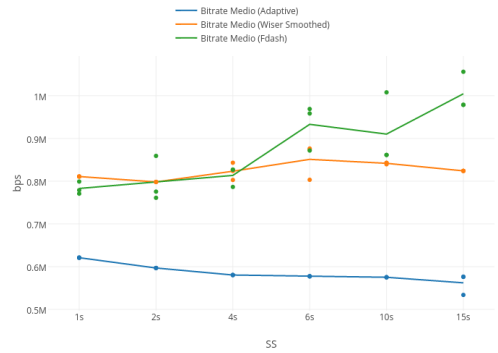
(b) FreeMbps buffer



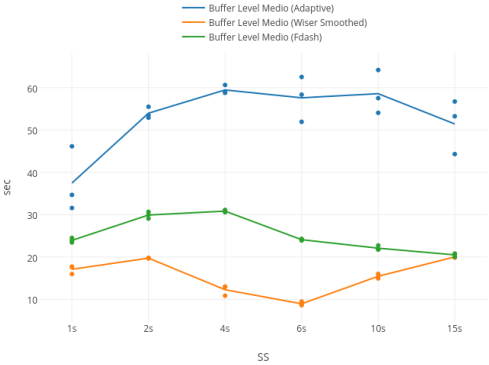
(c) 3 Mbps bitrate



(d) 3 Mbps buffer



(e) 1 Mbps bitrate



(f) 1 Mbps buffer

Figura 4.1: Grafici che riportano la media del bitrate e del buffer level per ogni lunghezza di segmento

SS $\leq$ 2s In questo caso Wiser Smoothed raggiunge il bitrate più alto possibile, ma mantiene il buffer abbastanza corto, facendo registrare così 5s e 6s nella tabella; al contrario FDash ha un bitrate mediamente basso, ma riempie molto velocemente il buffer (caso analogo al precedente). Adaptive si colloca tra i due, comportandosi discretamente sia per bitrate che per buffer. Entrambi (Adaptive e FDash) non ottengono pause nel playback. Notare che la variabilità della media di FDash possa essere molto alta poichè per segmenti da 2 secondi, i tre valori delle run sono decisamente diversi.

SS $\geq$ 4s In questo caso FDash è l'algoritmo migliore, in quanto riesce a far registrare il bitrate medio più alto per ogni SS e allo stesso tempo riempire il buffer velocemente. Adaptive si comporta abbastanza bene, mentre Wiser Smoothed si rivela poco prudente tenendo relativamente alto il bitrate, ma abbastanza corto il buffer (senza però effettuare pause). Anche qui si può notare che è presente randomicità tra le misurazioni.

A supporto di quanto detto si può osservare che, nel grafico in figura 4.3, per lunghezze brevi, FDash abbia un QI medio minore, mentre per SS maggiori si comporti meglio degli altri due algoritmi.

Infine, nei grafici relativi alla limitazione a 1 Mbps (4.1(e) e 4.1(f)), è interessante notare come l'Adaptive riesca a mantenere un bitrate medio costante facendo variare SS; tale bitrate è però abbastanza basso e il buffer tende quindi a riempirsi molto. Wiser Smoothed e FDash riescono invece a fornire qualità alte, ma occorre valutare i secondi di pausa per capire chi "vince" tra i due:

- per lunghezze di segmento da 1s a 6s compresi FDash è sicuramente migliore, per via del buffering medio di 6s che fa registrare Wiser (colonne 1s,2s,4s) e del vantaggio nella media di bitrate e buffer da parte di FDash (6s);

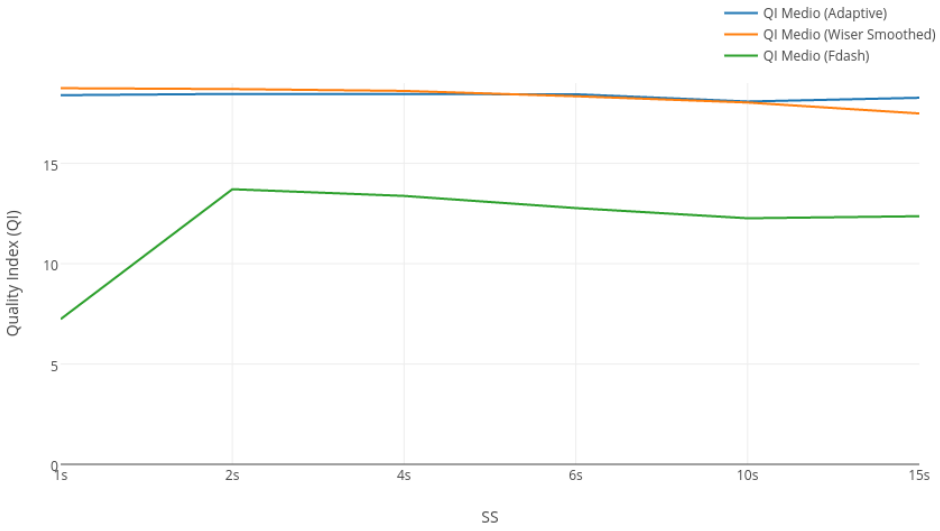


Figura 4.2: Media del QI con *FreeMbps* per ogni lunghezza del segmento e relativa a 3 run diverse del video

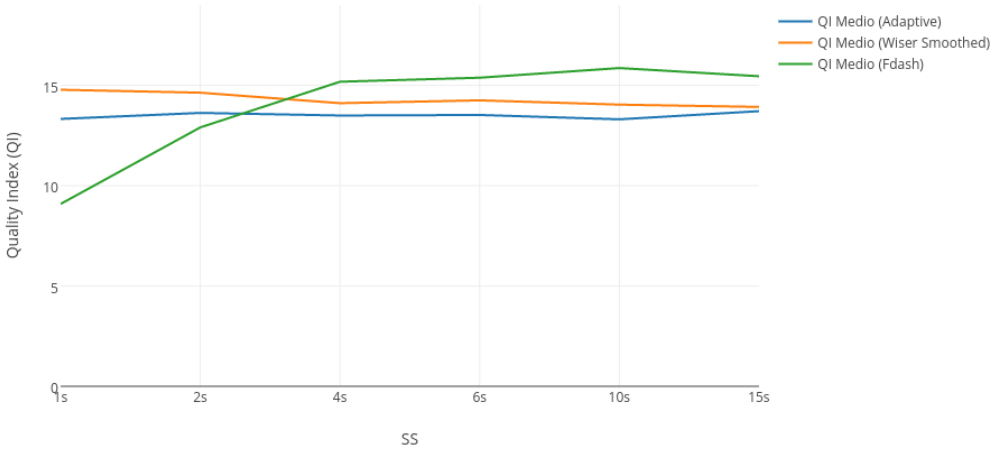


Figura 4.3: Media del QI con *3Mbps* per ogni lunghezza del segmento e relativa a 3 run diverse del video

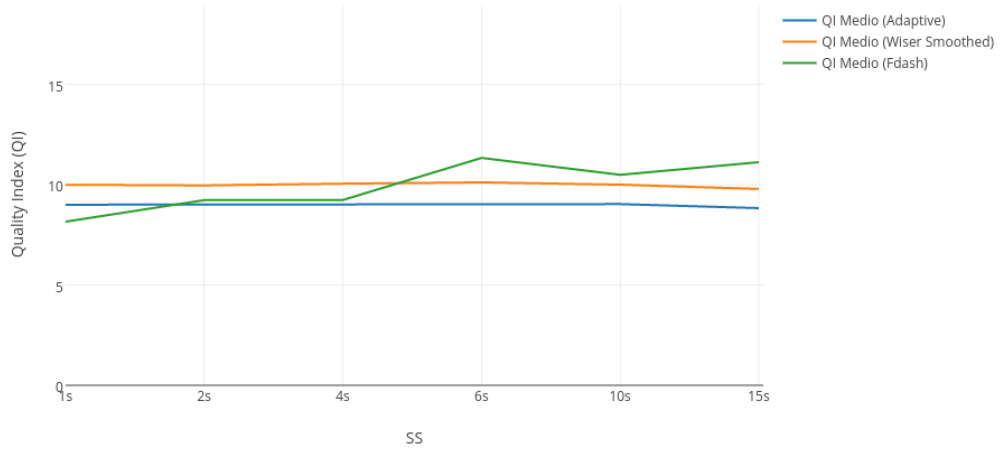


Figura 4.4: Media del QI con *1Mbps* per ogni lunghezza del segmento e relativa a 3 run diverse del video

- per lunghezze di segmento pari a 10s e 15s FDash fornisce bitrate mediamente più alti, ma ottiene rispettivamente 6 e 35 secondi di pausa. Ciò implica il netto vantaggio di Wiser per segmenti di 15s, mentre per quello che riguarda i 10s possiamo affermare che i valori dei grafici sono abbastanza simili (considerando anche la probabile randomicità dei dati).

Dynamic SS vs. Algoritmi di rate adaptation

La seconda parte di analisi statica si è concentrata sul confronto tra Dynamic SS, implementato assieme ad uno dei tre algoritmi di rate adaptation, e gli algoritmi di rate adaptation stessi, per cercare eventuali risultati che mostrassero che l'apporto di Dynamic SS sia un beneficio.

Con ognuna delle tre velocità discusse nel paragrafo precedente, si sono effettuate 5 run per ogni implementazione del Dynamic SS (Dynamic+Adaptive, Dynamic+Wiser, Dynamic+FDash), in modo da limitare la randomicità dei

dati; non si è voluto però effettuare una media tra i valori, considerando invece come valida la run che più si avvicinava a tale media.

La valutazione di Dynamic segue dal confronto di ogni sua implementazione con il rispettivo algoritmo di rate adaptation, considerato con le due lunghezze più differenti, ossia 1s e 15s. Per ogni ampiezza di banda e per ogni algoritmo avremo quattro grafici, che mostrano le analisi di bitrate, buffer, QI e lunghezza del segmento (quest'ultimo grafico sarà ovviamente riferito solo al Dynamic). In particolare si possono trovare bitrate e buffer nelle figure 4.5, 4.6 e 4.7 mentre le analisi relative a QI e lunghezza sono nelle immagini 4.8, 4.9 e 4.10.

Prima di procedere è necessario specificare un dettaglio importante ai fini dell'analisi. Per motivi progettuali e problematiche sorte a seguito del testing, è stato scelto di riadattare Adaptive se implementato insieme a Dynamic: si è impostato che per i primi 300ms l'algoritmo scarichi segmenti alla qualità minima; al contrario, limitando la connessione, Dynamic sarebbe andato in conflitto, causando alcuni istanti di buffering. Questa scelta si riflette nella valutazione di bitrate e QI: infatti i primi pacchetti abbassano di molto la media, facendo sì che il confronto con Adaptive-1s e Adaptive-15s sia poco significativo. Considereremo perciò questo problema in sede di analisi.

Prendiamo quindi in considerazione ciascun caso, effettuando le nostre valutazioni in base ai risultati dei grafici e delle tre tabelle 4.2, 4.3 e 4.4.

Tabella 4.3: Medie dei secondi di pausa relativi al testing di Dynamic SS in analisi statica

	Dynamic SS + Adaptive	Dynamic SS + Wiser Smoothed	Dynamic SS + FDash
Free Mbps	0 s	0 s	0 s
3 Mbps	0 s	12 s	0 s
1 Mbps	0 s	7 s	25 s

Free Mbps È in realtà un contesto non particolarmente interessante, ma utile per valutare gli algoritmi in condizioni ottimali. Come si può no-

tare dalle tabelle, in nessun caso si verificano pause, quindi tale discorso non sarà preso in considerazione.

- per ciò che riguarda Adaptive possiamo dire che entro i 40 sec circa (vedi fig. 4.8(b)) il Dynamic rileva che la connessione sia molto stabile, cambiando SS fino a 10s e successivamente a 15s. In questo modo il bitrate da scaricare è minore, ma il QI è al massimo (Cfr. fig. 4.5(a), 4.8(a)). Dynamic riesce a raggiungere la qualità massima prima degli altri, ma guardando la tabella ha una media di QI minore: questo deriva da quanto spiegato in precedenza. Possiamo dire che Dynamic non sia però inferiore agli altri due in termini di qualità offerta. Il buffer del Dynamic è mediamente maggiore di quello dell'Adaptive con 15s (18,5s contro 17,12s), ma non di 1s (20,1s). In generale riscontro positivo.

- per ciò che riguarda Wiser possiamo osservare molto bene come Dynamic sfrutti le peculiarità degli SS minori e maggiori (vedi fig. 4.5(c), 4.9(a)): facendo partire il video con SS=1s, è possibile ottenere un *delay di start* quasi insignificante, raggiungendo così simultaneamente QI massimo; una volta che la connessione si è stabilizzata, Dynamic può sfruttare al meglio la banda scaricando pacchetti con meno bitrate, ma di QI massimo.

  Il buffer del Dynamic è mediamente più basso degli altri (14,9s contro 17,6s e 16,5s), ma riesce comunque a mantenere un livello discreto. Banalmente nei primi secondi si discosta da Wiser-1s perchè vengono effettuati switch di SS, e il buffer si riempie ad una velocità sempre minore. Dopo i primi 50s il comportamento del Dynamic ricade su quello di Wiser-15s (anzi, è migliore). Non si verificano pause, quindi bene in generale.

- per ciò che riguarda FDash, si nota discretamente che il Dynamic è mediamente superiore agli altri nel bitrate (Cfr. tab 4.4) e nel QI (fig. 4.10(a)), riuscendo a rimanere in modo abbastanza costante

a qualità massima. Il suo buffer è all'incirca come quello di Fdash-15s. In generale, sebbene sia l'ultimo a raggiungere un QI alto, Dynamic è da preferire rispetto agli altri due.

3 Mbps Cominciamo a limitare la banda e a decrementare la possibilità di raggiungere bitrate alti.

- guardando i grafici 4.6(a), 4.6(b) e 4.8(c) possiamo dire che il migliore algoritmo per numero di switch nella qualità è sicuramente Adaptive-15s; anche il Dynamic, ma solo dopo che si sia stabilizzato (dall'istante 25 circa in poi). Nei primi 100s il Dynamic ha un buffer mediamente minore di Adaptive-1s, ma in certe situazioni va meglio di Adaptive-15s; a partire da 100 in poi si comporta più o meno come Adaptive-15s. Sebbene il buffer medio del Dynamic sia minore di quello di Adaptive-1s ($22,8 < 23,0$), possiamo affermare che, tralasciando il sussulto dell'istante 20 e considerando quanto detto in precedenza per l'Adaptive, il Dynamic agisca bene anche perchè non si verificano in nessun caso pause (vedi tab 4.2 e 4.3).
- dai grafici 4.6(c) e 4.9(c) vediamo che il Dynamic acquisisce la massima qualità in poco tempo e la mantiene in modo costante per tutto il playback. Sebbene il suo buffer sia in media migliore di quello di Wiser-1s, Dynamic fa registrare 12s di pausa, contro i 5 di Wiser-1s. Visti i dati registrati nelle tabelle, dobbiamo affermare che in questo caso il migliore dei tre è probabilmente Wiser-15s, in quanto unico a non avere interruzioni e a mantenere alti e costanti bitrate e buffer, a discapito di un inizio prolungato del video a qualità minima.
- In generale Dynamic si comporta abbastanza male: raggiunge la massima qualità, ma la mantiene per poco (fig. 4.6(e) e 4.10(c)). All'istante 90 circa, infatti, si nota che il buffer (fig. 4.6(f)) scende sotto i 10s e questo fa sì che la qualità venga drasticamente abbas-

sata (Cfr. app. A.3). Inoltre il buffer di Dynamic ha una media compresa tra quelle di FDash-1s e FDash-15s, ma, posto che in nessuno dei tre casi si sono verificate pause, possiamo affermare che per media del bitrate e numero di switch il migliore sia stato FDash-15s.

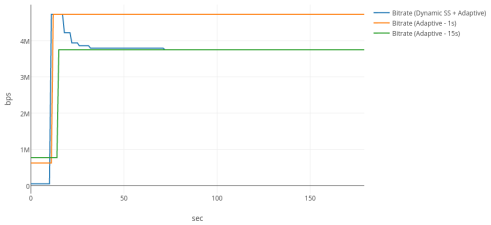
1 Mbps Limitazione massima, condizioni da considerarsi come critiche.

- Il Dynamic si comporta bene: osservando i grafici 4.7(a) e 4.8(e) vediamo che mantiene alta e costante la qualità per tutto il playback (tranne che all'inizio, come già spiegato), analogamente ad Adaptive-1s e Adaptive-15s. Molto bene anche in termini di buffer (fig. 4.7(b)), che mantiene praticamente sempre superiore a quello di Adaptive-1s e soprattutto Adaptive-15s ($33,4s > 30,7s > 26,8s$). Vista anche l'assenza di pause in tutti e tre gli algoritmi possiamo affermare che in queste circostanze il migliore sia stato Dynamic SS.
- Il Dynamic è assimilabile al comportamento di Wiser-15s, ma allo stesso tempo fa registrare diversi picchi che portano il bitrate a un livello maggiore. Il buffer è però molto basso e questo ci porta a far prevalere Wiser-15s sugli altri due, vista anche la quantità di interruzioni registrate (0s per Wiser-15s mentre 7s e 6s rispettivamente per Dynamic e Wiser-15s).
- Da segnalare il fatto che Dynamic pecchi in termini di velocità nell'acquisire la qualità massima: possiamo vedere infatti dal grafico 4.7(e) che è decisamente l'ultimo dei tre ad innalzare il bitrate (istante 60 circa). In generale si comporta mediamente come Fdash-15s: per QI risulta migliore Dynamic, mentre per media del buffer FDash-15s; in entrambi i casi il gap è relativamente contenuto. FDash-1s invece, ha un bitrate medio abbastanza basso, ma un buffer mediamente migliore degli altri due (22,9s). Dynamic ed FDash-15s, al contrario dell'assenza di pause di FDash-1s,

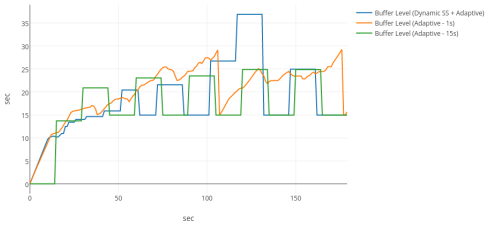
fanno registrare picchi molto bassi nel grafico del buffer (4.7(f)), ottenendo così rispettivamente 25 e 35 secondi di interruzione, che incidono molto nella nostra valutazione. Possiamo dire quindi che, in questa circostanza Dynamic si è rivelato poco utile ed il migliore dei tre sia stato FDash-1s.

Tabella 4.4: Medie di bitrate e buffer level per ogni velocità e algoritmo

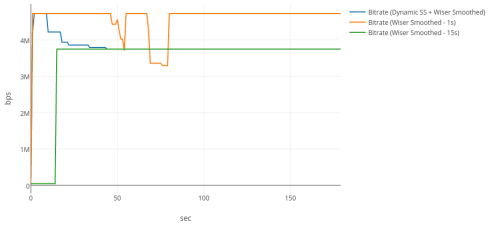
			Average Bitrate (bps)	Average QI	Average Buffer Level (sec)
Free Mbps	Dynamic + Adaptive *		3588270	17,8	18,5
	Adaptive	1s	4453001	18,3	20,1
		15s	3519105	18,2	17,12
	Dynamic + Wiser Smoothed		3808860	18,8	14,9
	Wiser Smoothed	1s	4591080	18,7	17,6
		15s	3463398	17,3	16,5
	Dynamic + FDash		2584089	13,5	24,4
	FDash	1s	710383	6,8	49,4
		15s	2029840	12,3	25,0
3 Mbps	Dynamic + Adaptive *		1907022	13,0	22,8
	Adaptive	1s	1901780	13,1	23,0
		15s	1953828	13,6	21,6
	Dynamic + Wiser Smoothed		2365519	14,8	14,3
	Wiser Smoothed	1s	2572097	14,7	9,2
		15s	2197766	13,7	17,9
	Dynamic + FDash		987080	6,4	33,0
	FDash	1s	1388215	8,6	31,7
		15s	2392833	13,2	21,5
1 Mbps	Dynamic + Adaptive *		557393	8,5	33,4
	Adaptive	1s	620705	9	26,8
		15s	582810	8,75	30,7
	Dynamic + Wiser Smoothed		874028	10,4	6,5
	Wiser Smoothed	1s	796588	9,97	9,8
		15s	726909	9,66	26,2
	Dynamic + FDash		995091	6,27	21,6
	FDash	1s	809531	5,42	22,9
		15s	978635	5,41	20,7



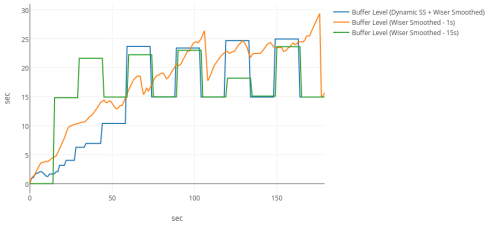
(a) Free Mbps bitrate (Adaptive)



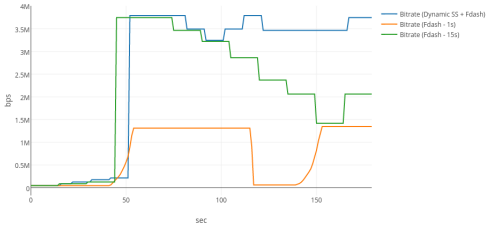
(b) Free Mbps buffer (Adaptive)



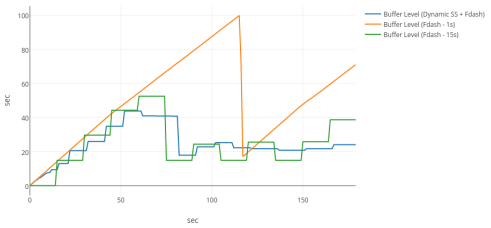
(c) Free Mbps bitrate (Wiser Smoothed)



(d) Free Mbps buffer (Wiser Smoothed)

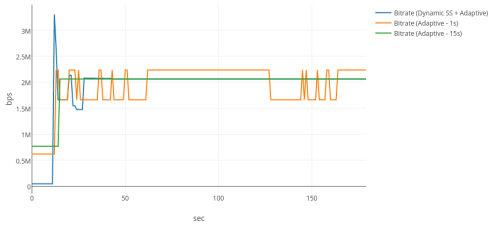


(e) Free Mbps bitrate (FDash)

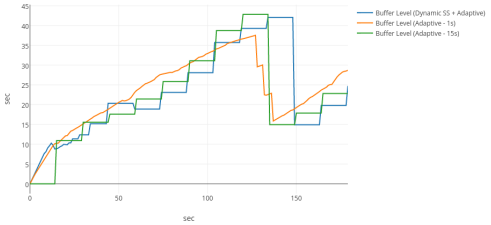


(f) Free Mbps buffer (FDash)

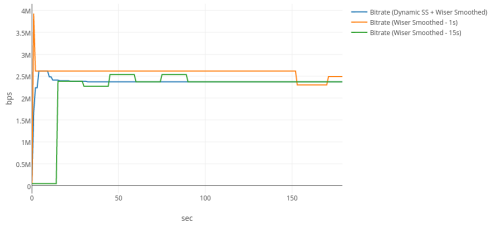
Figura 4.5: Grafici relativi a Free Mbps che riportano bitrate e buffer level per ogni algoritmo



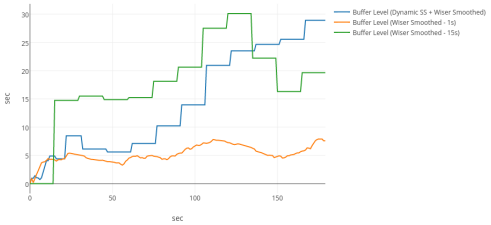
(a) 3 Mbps bitrate (Adaptive)



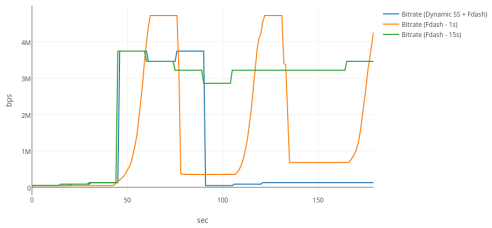
(b) 3 Mbps buffer (Adaptive)



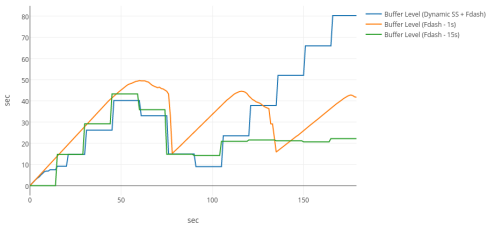
(c) 3 Mbps bitrate (Wiser Smoothed)



(d) 3 Mbps buffer (Wiser Smoothed)

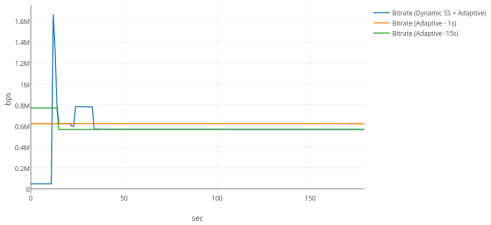


(e) 3 Mbps bitrate (FDash)

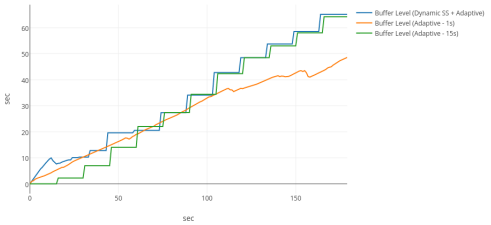


(f) 3 Mbps buffer (FDash)

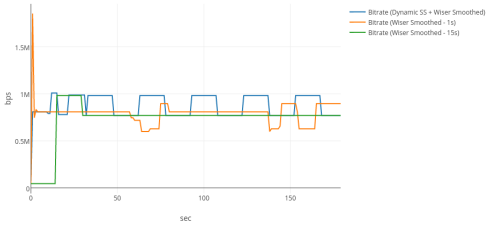
Figura 4.6: Grafici relativi a 3 Mbps che riportano bitrate e buffer level per ogni algoritmo



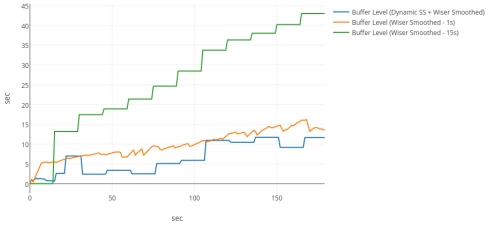
(a) 1 Mbps bitrate (Adaptive)



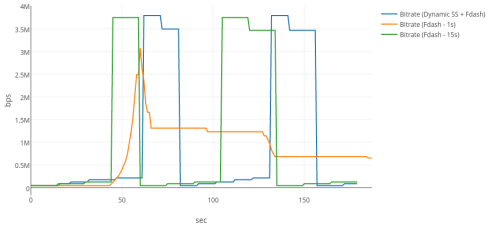
(b) 1 Mbps buffer (Adaptive)



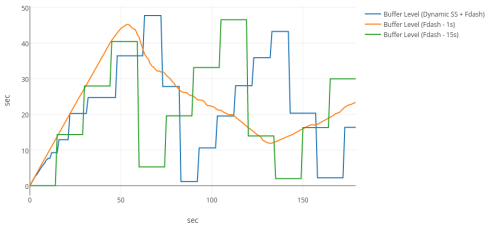
(c) 1 Mbps bitrate (Wiser Smoothed)



(d) 1 Mbps buffer (Wiser Smoothed)

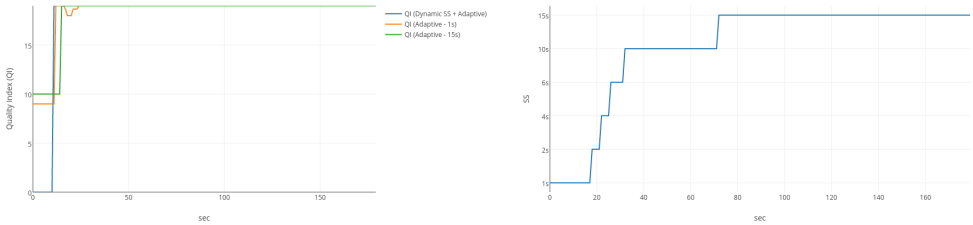


(e) 1 Mbps bitrate (FDash)



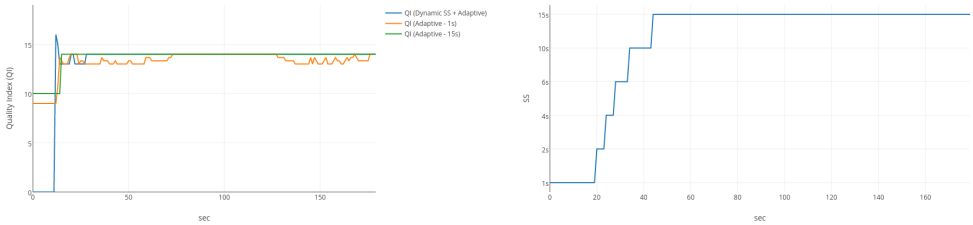
(f) 1 Mbps buffer (FDash)

Figura 4.7: Grafici relativi a 1 Mbps che riportano bitrate e buffer level per ogni algoritmo



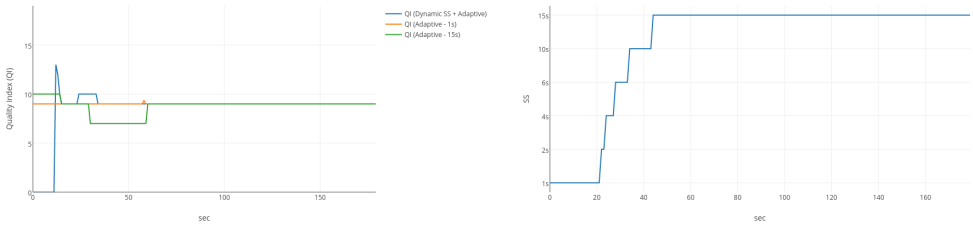
(a) Free Mbps QI (Adaptive)

(b) Free Mbps length (Adaptive)



(c) 3 Mbps QI (Adaptive)

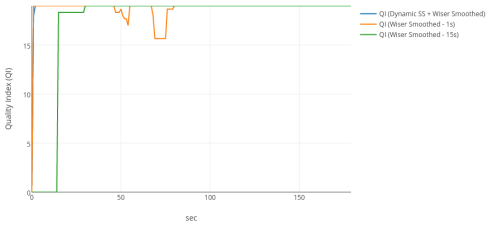
(d) 3 Mbps length (Adaptive)



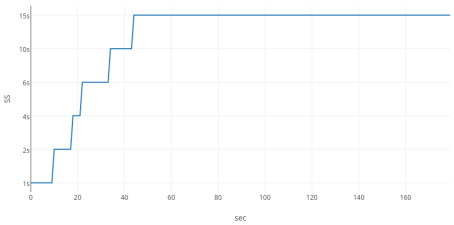
(e) 1 Mbps QI (Adaptive)

(f) 1 Mbps length (Adaptive)

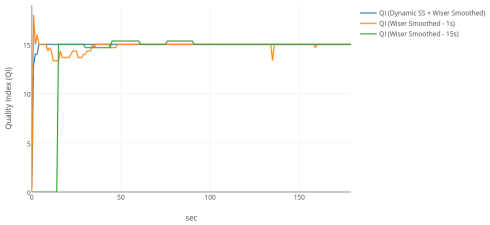
Figura 4.8: Grafici che riportano QI e cambio di lunghezza del Dynamic per ogni velocità (Adaptive)



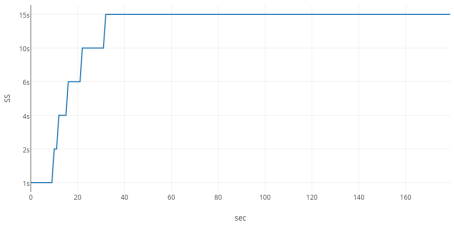
(a) Free Mbps QI (Wiser Smoothed)



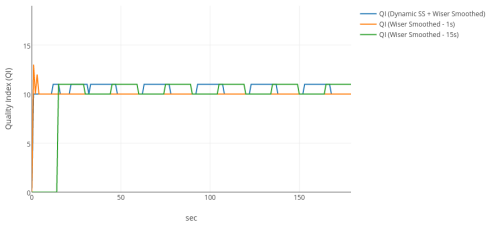
(b) Free Mbps length (Wiser Smoothed)



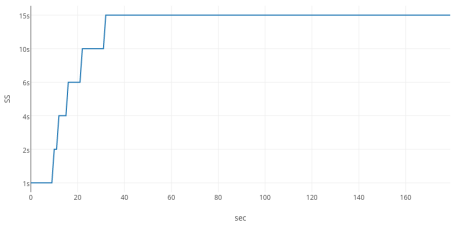
(c) 3 Mbps QI (Wiser Smoothed)



(d) 3 Mbps length (Wiser Smoothed)

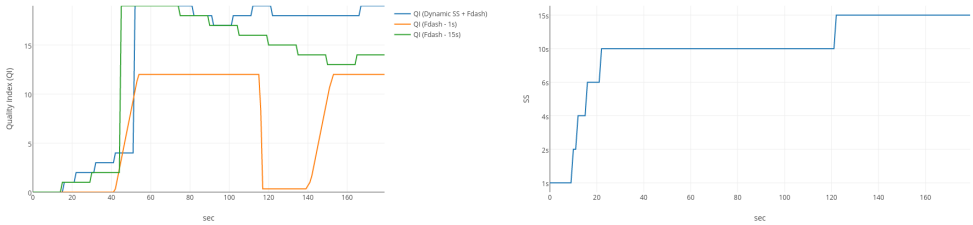


(e) 1 Mbps QI (Wiser Smoothed)



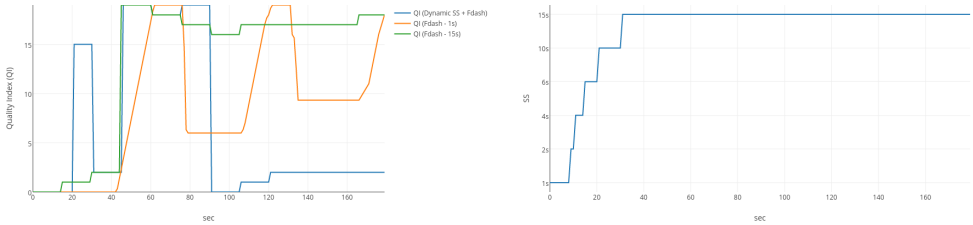
(f) 1 Mbps length (Wiser Smoothed)

Figura 4.9: Grafici che riportano QI e cambio di lunghezza del Dynamic per ogni velocità (Wiser Smoothed)



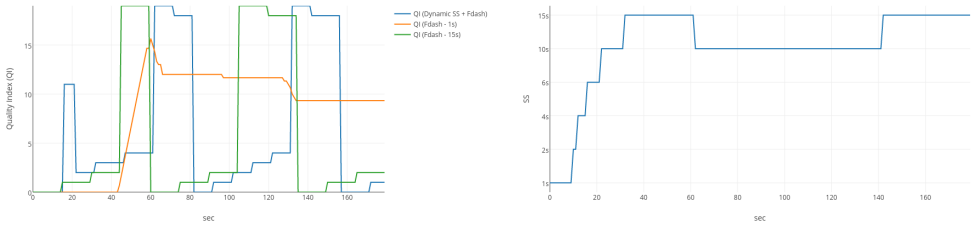
(a) Free Mbps QI (FDash)

(b) Free Mbps length (FDash)



(c) 3 Mbps QI (FDash)

(d) 3 Mbps length (FDash)



(e) 1 Mbps QI (FDash)

(f) 1 Mbps length (FDash)

Figura 4.10: Grafici che riportano QI e cambio di lunghezza del Dynamic per ogni velocità (FDash)

4.3.2 Analisi dinamica

Con ”analisi dinamica” vogliamo intendere tutto quell'insieme di esperimenti e valutazioni che sono stati effettuati in condizioni di mobilità, ovvero con un device connesso ad una rete Wi-Fi, ma in movimento.

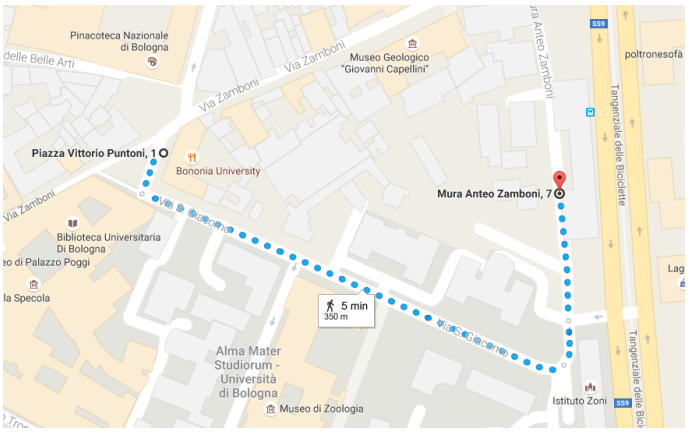


Figura 4.11: Percorso scelto per effettuare i test relativi all'analisi dinamica

Nella fattispecie, i test di cui parleremo in questo paragrafo sono stati svolti mediante la connessione Wi-Fi dell'Università di Bologna (*ALMA Wi-Fi*), effettuando a piedi un percorso prestabilito: in figura 4.11 è rappresentato tale percorso (dalla facoltà di Informatica alla mensa di Piazza Vittorio Puntoni), che per ogni esperimento è stato affrontato sia all'andata che al ritorno, eseguendo così una run quasi completa di Big Buck Bunny.

Queste condizioni sono state necessarie per poter testare al meglio il funzionamente di Dynamic SS nella rilevazione dell'instabilità della connessione (passando da una facoltà all'altra il segnale dell'Alma Wi-Fi subisce alcuni picchi).

Si è deciso di fare un confronto delle run con presenza e assenza del Dynamic, raccogliendo i dati relativi a Dynamic+Adaptive, Dynamic+Wiser, Dynamic+FDash e ai soli algoritmi di rate adaptation con lunghezza fissa (SS=1s,15s). Le nostre analisi si baseranno sulla tabella 4.5 e sui grafici presenti nelle figure 4.12, 4.13, 4.14, 4.15, 4.16, 4.17, 4.18, 4.19, 4.20. In particolare, i primi tre raffigurano il cambiamento di SS che Dynamic+Adaptive, Dynamic+Wiser e Dynamic+FDash hanno ottenuto, mentre le restanti figure, suddivise in gruppetti da tre grafici, contengono il confronto degli algoritmi in base a bitrate, buffer level e QI.

Per quanto concerne il cambiamento di SS, possiamo notare dai grafici

4.12, 4.13 e 4.14 che nessuna delle tre coppie di algoritmi abbia raggiunto (e mantenuto per un tempo considerevole) lunghezze di segmento alte, in linea con l'idea che Dynamic controlli la stabilità della connessione: si sono mantenute per lo più infatti, lunghezze di segmento di taglio inferiore. È bene specificare però che Dynamic+Wiser abbia registrato un picco nel grafico, raggiungendo per qualche secondo SS=6s e SS=10s.

Tabella 4.5: Medie di bitrate, QI e secondi di pausa ottenute a seguito del testing per l'analisi dinamica

		Average Bitrate (bps)	Average QI	Pause Time (s)
Dynamic + Adaptive		3216389	15,95	15
Adaptive	1s	3545126	16,46	16
	15s	2775129	16,05	21
Dynamic + Wiser Smoothed		3877041	18,18	52
Wiser Smoothed	1s	3974103	17,38	88
	15s	2967942	15,18	39
Dynamic + FDash		1771608	10,3	5
FDash	1s	1244297	8,60	0
	15s	1535923	9,72	13

Analizziamo ora i grafici 4.15, 4.16, 4.17, 4.18, 4.19 e 4.20, cercando di fare valutazioni più precise possibili.

Nei test relativi ad Adaptive possiamo osservare come il bitrate di Dynamic+Adaptive sia superiore a quello di Adaptive-15s, ma inferiore a quello di Adaptive-1s (Cfr. tab. 4.5). In realtà, come già affermato in precedenza, in sede di confronto tra Dynamic e algoritmi con lunghezza fissata, il bitrate è una variabile poco utile in termini di analisi, perchè fortemente dipendente dalla lunghezza del segmento. Valutiamo quindi mediante i grafici 4.15(c) e 4.16(c) e la tabella 4.5 relativi al QI. In relazione anche a quanto riportato nella tabella 4.5 possiamo dire che Dynamic+Adaptive perde dagli altri due

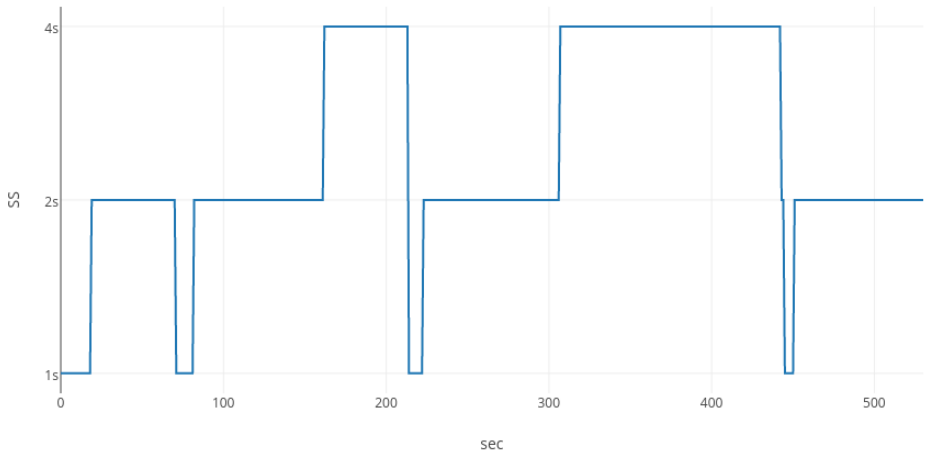


Figura 4.12: Variazione di SS per Dynamic+Adaptive nella run relativa alla figura 4.11

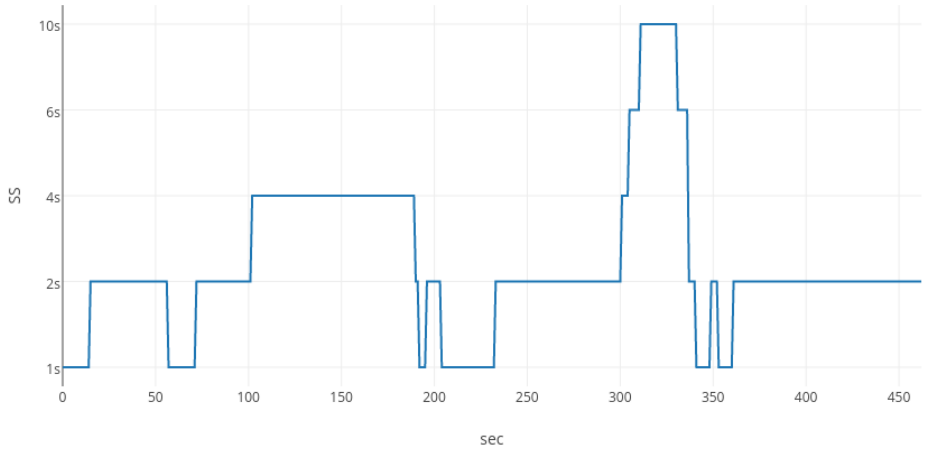


Figura 4.13: Variazione di SS per Dynamic+Wiser nella run relativa alla figura 4.11

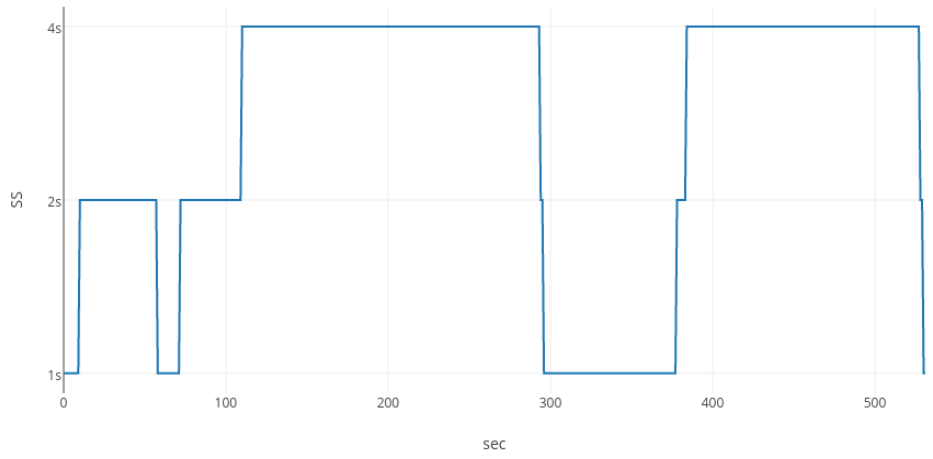


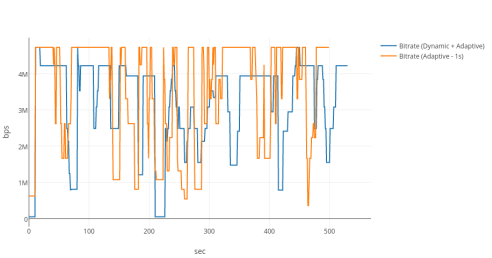
Figura 4.14: Variazione di SS per Dynamic+FDash nella run relativa alla figura 4.11

a livello di qualità offerta, anche considerando il riadattamento dell'algoritmo (vedi par. precedente); per quanto riguarda il buffer invece, vince su entrambi. Possiamo affermare che, sebbene vi sia un gap nel QI tra Dynamic e gli altri due algoritmi, questo sia molto basso e quindi approssimativamente nullo; Dynamic+Adaptive può quindi essere valutato bene, anche per quanto riguarda i secondi di interruzione. Essendo stata fatta un'unica run per ogni test, la variabilità dei dati può essere ingente, considerando anche che una prova del genere possa essere sottoposta ad eventi più o meno casuali.

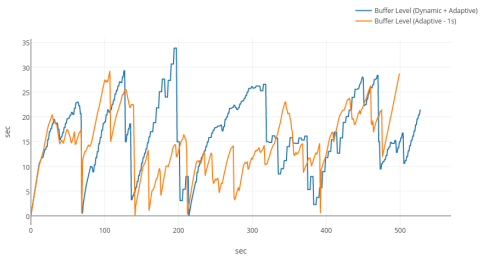
Nei test relativi a Wiser Smoothed possiamo vedere come, a livello di bitrate, questo caso sia analogo al precedente; valutando invece i grafici relativi al QI e le rispettive medie (riportate in tab.4.5), notiamo che Dynamic+Wiser vince sia su Wiser-1s che su Wiser-15s. Anche per quanto riguarda il buffer Dynamic è migliore degli altri, ma in termini di interruzioni perde da Wiser-15s (su cui però stravince in qualità offerta). Guardando i grafici 4.18(a) e 4.18(c), si nota un particolare interessante: all'istante 300, Wiser-15s ha subìto un netto calo nella qualità, mentre Dynamic no. Questo perchè Dy-

namic ha potuto cambiare gradualmente SS, passando a lunghezze maggiori (che hanno meno problemi in termini di shift). In conclusione possiamo dire che Dynamic+Wiser si sia comportato meglio degli altri due algoritmi.

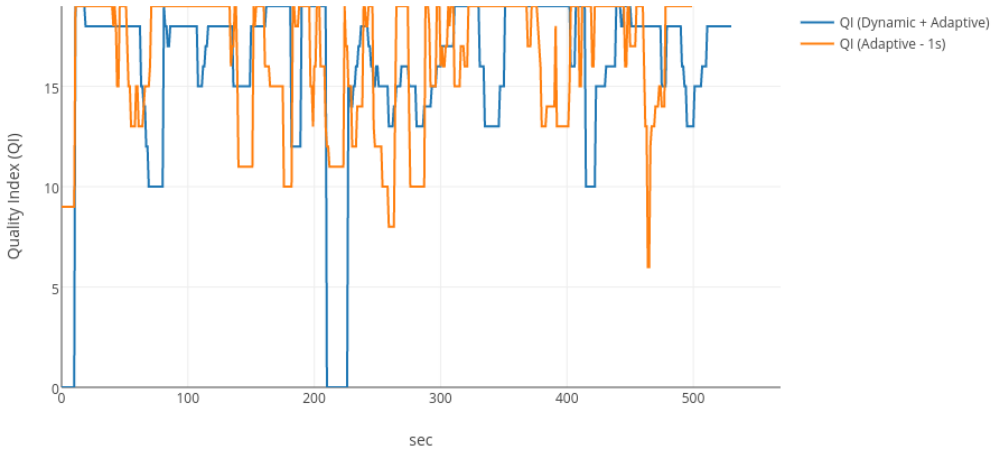
Infine, nei test relativi a FDash vediamo come Dynamic riesca a migliorare nettamente l'utilizzo di tale algoritmo, in quanto vince sugli altri due sia a livello di bitrate che in termini di QI. Per quanto riguarda il buffer, Dynamic+FDash è mediamente minore degli altri due, ma considerando i secondi di interruzione riportati in tabella 4.5 possiamo affermare che FDash si sia comportato in generale meglio di FDash-1s e FDash-15s.



(a) Bitrate (Adaptive-1s)

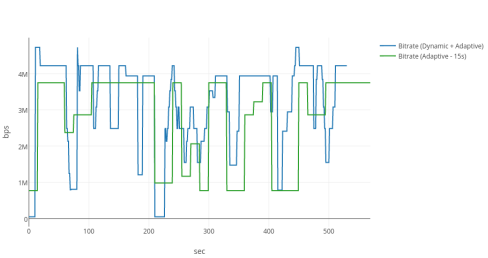


(b) Buffer Level (Adaptive-1s)

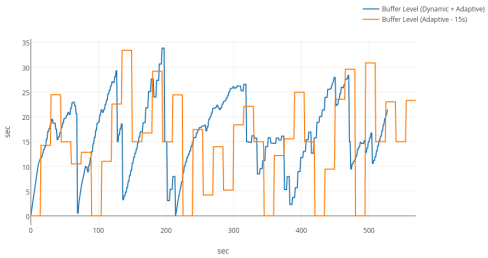


(c) QI (Adaptive-1s)

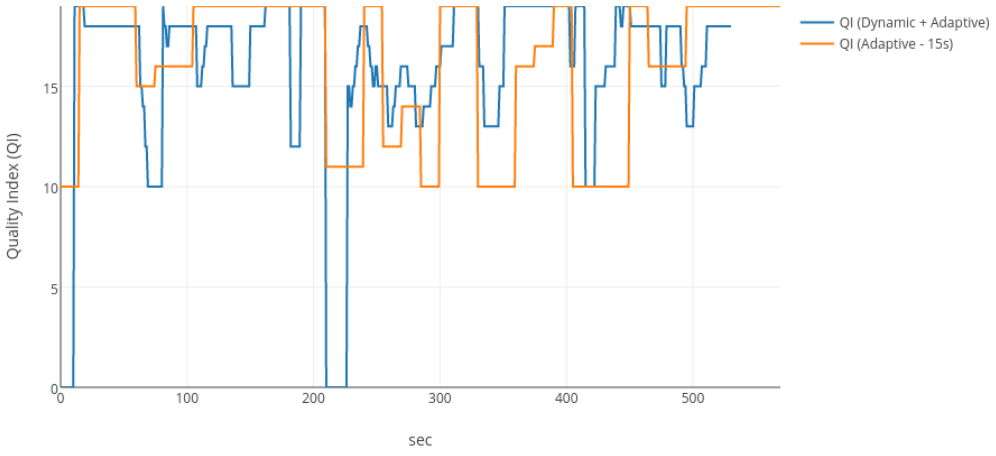
Figura 4.15: Grafici del confronto tra Dynamic+Adaptive e Adaptive-1s



(a) Bitrate (Adaptive-15s)

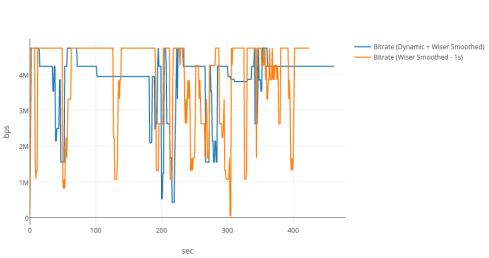


(b) Buffer Level (Adaptive-15s)

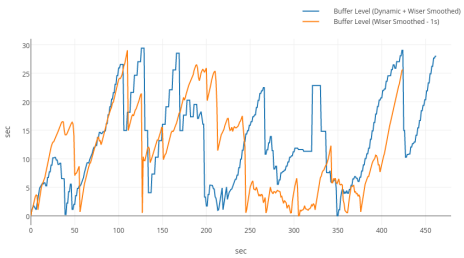


(c) QI (Adaptive-15s)

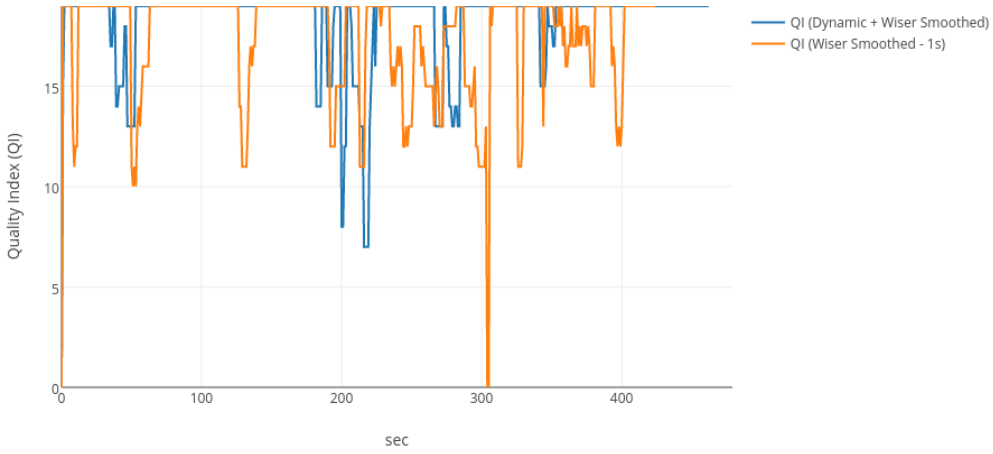
Figura 4.16: Grafici del confronto tra Dynamic+Adaptive e Adaptive-15s



(a) Bitrate (Wiser-1s)

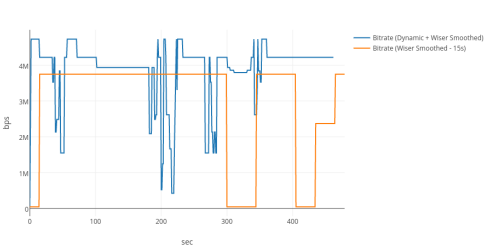


(b) Buffer Level (Wiser-1s)

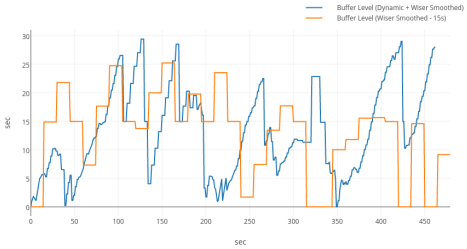


(c) QI (Wiser-1s)

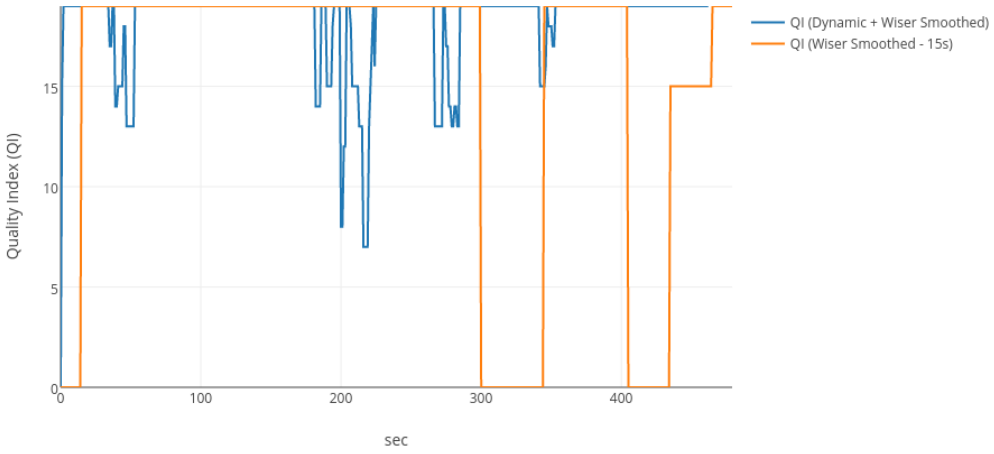
Figura 4.17: Grafici del confronto tra Dynamic+Wiser e Wiser-1s



(a) Bitrate (Wiser-15s)

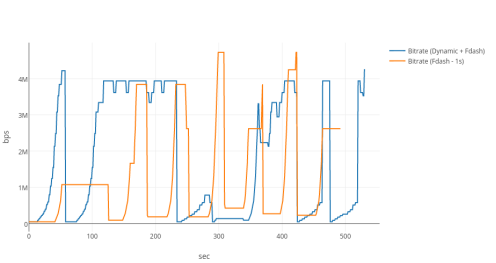


(b) Buffer Level (Wiser-15s)

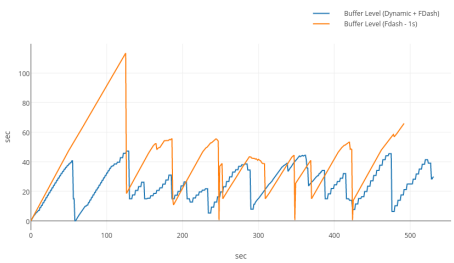


(c) QI (Wiser-15s)

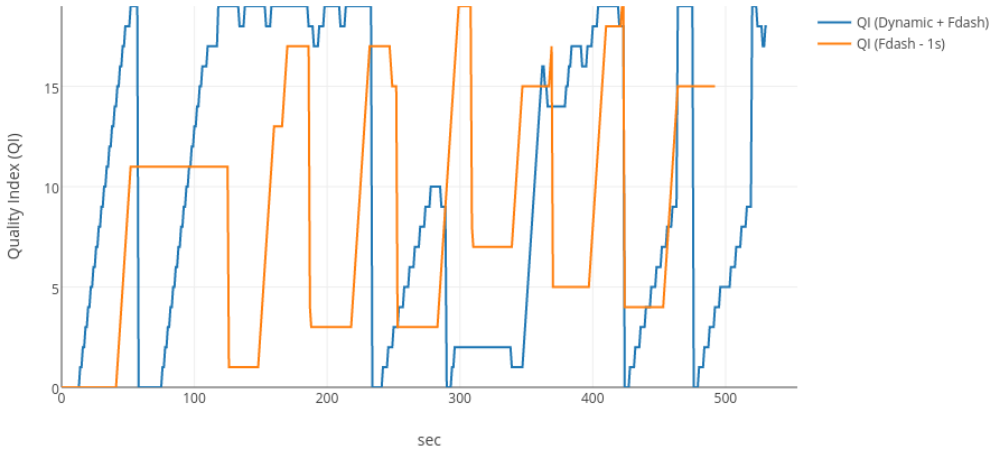
Figura 4.18: Grafici del confronto tra Dynamic+Wiser e Wiser-15s



(a) Bitrate (FDash-1s)

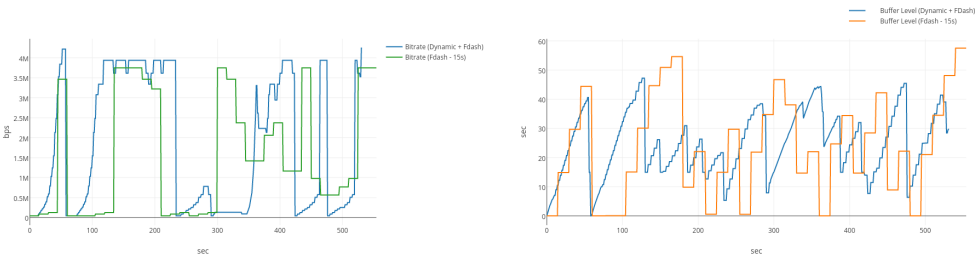


(b) Buffer Level (FDash-1s)



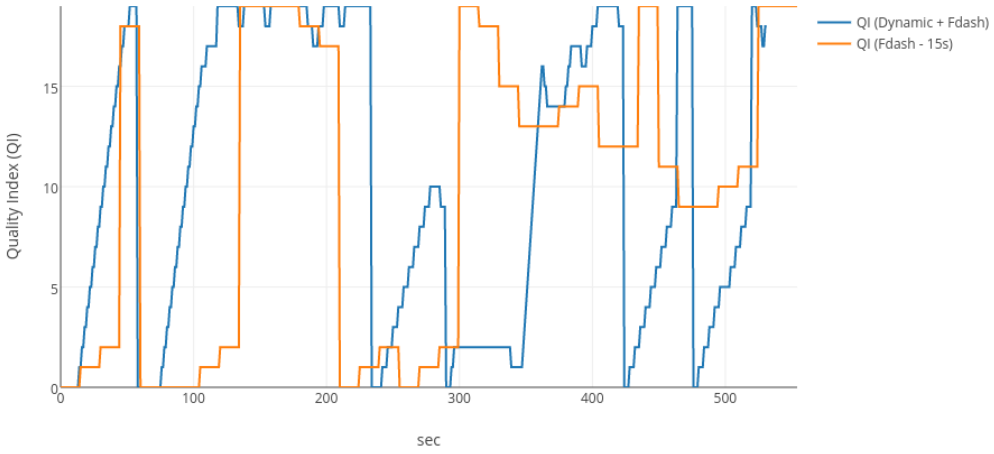
(c) QI (FDash-1s)

Figura 4.19: Grafici del confronto tra Dynamic+FDash e FDash-1s



(a) Bitrate (FDash-15s)

(b) Buffer Level (FDash-15s)



(c) QI (FDash-15s)

Figura 4.20: Grafici del confronto tra Dynamic+FDash e FDash-15s

4.3.3 Risultati e valutazioni

In conclusione alle analisi affrontate in questa sezione, cerchiamo di controllare i risultati ottenuti raggruppando e riassumendo caso per caso le valutazioni finali.

Analisi Statica: test sulle lunghezze

Abbiamo visto che in generale non si è verificata una vittoria schiacciante da parte di uno dei tre algoritmi, ma che, in base alle condizioni del segnale

e alla lunghezza del segmento, le prestazioni sono state migliori o peggiori: in particolare, in ottime condizioni affermiamo che Wiser Smoothed e Adaptive vadano molto bene, mentre in condizioni di limitazione sia necessaria una suddivisione in casistiche: con 3Mbps e basse lunghezze Adaptive ha la supremazia, mentre per lunghezze maggiori il migliore è certamente FDash; con 1Mbps e basse lunghezze FDash è andato molto bene, ma per lunghezze maggiori Wiser Smoothed si è comportato meglio.

Analisi Statica: Dynamic SS

A seguito di quanto ottenuto dalle riflessioni del paragrafo 4.3.1, possiamo affermare che la presenza di Dynamic SS non abbia cambiato drasticamente le prestazioni dei tre algoritmi analizzati, anzi: in condizioni critiche Dynamic si è comportato peggio in generale, facendo propendere le nostre scelte sulla sua assenza. In particolare, il solo algoritmo Adaptive è riuscito a sfruttarlo al meglio, mentre gli altri due si sono mostrati poco prestanti. In condizioni ottimali di connessione però, Dynamic ha funzionato bene, rivelandosi una discreta tecnica utile ad accompagnare l'algoritmo di rate adaptation implementato all'interno del client DASH.

In conclusione è bene precisare che i parametri utilizzati all'interno dei due algoritmi (Wiser e FDash) incidano abbastanza sui risultati e che quindi avrebbero bisogno di essere scelti in modo più analitico.

Analisi Dinamica

In ambito di analisi con mobilità, possiamo concludere che Dynamic si sia rivelato un buon algoritmo che però non si mostra in netto vantaggio rispetto alla selezione statica di SS. È invece opportuno affermare che vi sia un trade-off, ovvero che il funzionamento di Dynamic non sia indipendente dall'algoritmo di rate adaptation e che in base alle preferenze dell'utente possa essere scelto insieme ad uno di essi. Precisiamo però che non si è data troppa importanza alla possibile presenza di randomicità nei dati, dovuta al fatto che si è svolto un singolo test per ciascun caso.

Secondo quanto ottenuto dalle analisi, possiamo dire che Dynamic possa migliorare Wiser Smoothed e FDash, mentre sull'Adaptive restano alcuni dubbi, legati al fatto che tale algoritmo sia stato riadattato (anche se in termini di pause è migliore).

Conclusioni

In conclusione al lavoro svolto per questa tesi, possiamo affermare che la libreria Exoplayer si sia rivelata molto utile nello svolgimento di implementazione e analisi degli algoritmi, in quanto facilmente customizzabile. Questo fatto riguarda soprattutto lo sviluppo di Dynamic SS, che essendo un algoritmo "nuovo" ha richiesto uno studio approfondito della libreria, per avere sotto controllo eventuali discordanze progettuali dovute alla coabitazione dell'algoritmo e di un ambiente mai stato pensato per procedimenti di quel tipo.

In termini di analisi possiamo dire che Adaptive è stato, tra gli algoritmi di rate adaptation valutati, quello che si è comportato meglio in generale. Wiser Smoothed è preferibile solo in certe situazioni, per il resto si è rivelato troppo imprudente, alzando al massimo il bitrate da richiedere a discapito di un buffer molto corto, che ha portato spesso ad ingenti pause. FDash, al contrario, si è dimostrato troppo conservativo, offrendo in certi casi una qualità bassa e un buffer inaspettatamente pieno; come già affermato, questo fatto dipende fortemente dalla scelta dei parametri SHORT, CLOSE e LONG. Per quanto riguarda la valutazione di Dynamic SS possiamo dire che, da quanto visto, non dovrebbe migliorare drasticamente le prestazioni dell'algoritmo di rate adaptation e che la sua presenza possa essere utile o meno in base a tale algoritmo e alle condizioni del segnale: dalle analisi vediamo come Dynamic riesca a migliorare Adaptive in stabilità, al contrario di ciò che mostrano i test dinamici; Wiser e FDash invece hanno avuto il comportamento opposto, ottenendo prestazioni migliori in mobilità (se accompagnati da Dynamic)

piuttosto che nell'analisi statica.

Precisiamo infine che i test relativi al Dynamic sono stati svolti mantenendo probabilmente troppa randomicità nei dati, soprattutto per l'analisi dinamica; per questo motivo le piccole differenze riscontrate nelle valutazioni potrebbero variare di molto in base alle condizioni del testing.

4.4 Possibili estensioni

Come è possibile notare dall'ingente quantità di dati e grafici su cui si è lavorato all'interno di questi studi, è chiaro che gli ambiti "toccati" dalla tesi siano molto ampi e necessitino in realtà di fasi sperimentali più durature, possibilmente affrontate in team, in modo da aumentare i test e ridurre così randomicità ed eventuali errori.

Di seguito si elencano le principali estensioni e i miglioramenti su cui si potrebbe lavorare in futuro:

- Valutazione più analitica e precisa dei parametri utilizzati per l'algoritmo Wiser Smoothed (δ e p);
- Valutazione più analitica e precisa dei parametri utilizzati per l'algoritmo FDash (*SHORT*, *CLOSE* e *LONG*);
- Miglioramento dell'impatto prestazionale del vettore dei throughput misurati in Dynamic SS;
- Miglioramento dell'algoritmo Dynamic SS a livello matematico, per dare maggiore priorità al cambio di SS (eventualmente dando maggiore peso al gap tra le misurazioni e la media);
- Implementazione di Dynamic+Adaptive possibilmente senza riadattamento, in modo da avere un confronto più significativo;
- Aumento della quantità di run effettuate nel testing delle lunghezze di segmento, per ridurre al minimo la randomicità dei dati;

- Aumento della quantità di run effettuate nel testing di Dynamic SS, cercando di effettuare su di esso uno studio più completo e meno dipendente dalle misurazioni effettuate;
- Implementazione di Dynamic SS in altri contesti, per valutarne le differenze principali (ad esempio su altri dispositivi con un diverso sistema operativo - iOS, WP ecc.);

Appendice A

Pseudo-codice degli algoritmi

A.1 Adaptive Evaluator

Algorithm 1: Pseudo-codice dell'algoritmo *Adaptive*

Data: τ
Result: formato richiesto (Representation)
currentFormat $\leftarrow$ formato del segmento precedente;
idealFormat $\leftarrow$ determineIdealFormat(τ) ;
isHigher $\leftarrow$ $idealFormat > currentFormat$;
isLower $\leftarrow$ $idealFormat < currentFormat$;
if *isHigher* **then**
    if $buffer < minDurationToRetainAfterDiscardMs$ **then**
        idealFormat $\leftarrow$ currentFormat;
    else
        elimina tutti i segmenti non HD già scaricati;
else
    idealFormat $\leftarrow$ currentFormat;
return idealFormat
Function *determineIdealFormat (τ)*
    idealBitrate $\leftarrow \tau \cdot 0.75$;
    return miglior formato in base a idealBitrate;

A.2 Wiser Smoothed Evaluator

Algorithm 2: Pseudo-codice dell'algoritmo *Wiser Smoothed*

Data: τ

Result: formato richiesto (Representation)

currentFormat ← formato del segmento precedente;

if *la coda dei segmenti è vuota* **then**

  idealFormat ← formato peggiore di tutti;

else

  $idealFormat \leftarrow$

  $determineIdealSmoothedFormat(getEstimatedThroughput(\tau))$;

return idealFormat

Function *getEstimatedThroughput* (τ)

  if *la coda ha 1 o 2 segmenti caricati* **then**

    estimatedT ← τ;

  else

    estimatedT ← (1 - s) · lastThroughput[1] + ($\delta \cdot \tau$);

    if *la coda ha almeno 4 segmenti* **then**

      $p \leftarrow | \frac{\tau - lastThroughput[1]}{lastThroughput[1]} |$;

      $\delta \leftarrow \frac{1}{1+exp(-21 \cdot (p-0.2))}$;

  lastThroughput[0] ← lastThroughput[1];

  lastThroughput[1] ← estimatedT;

  return estimatedT;

Function *determineIdealSmoothedFormat (estimatedT)*

  return miglior formato in base a estimatedT;

A.3 FDash Evaluator

Algorithm 3: Pseudo-codice dell'algoritmo *FDash*

Data:

Result: formato richiesto (Representation)

$currentFormat \leftarrow$ formato del segmento precedente;

$idealFormat \leftarrow getIdealFormatOnFuzzyLogic(currentFormat)$;

return idealFormat;

Function *getIdealFormatOnFuzzyLogic (current)*

- **if** $buffer < SHORT$ *and falling* **then**
  - return formato peggiore;
- **if** $buffer < CLOSE$ *and falling* **then**
  - return formato di qualità precedente;
- **if** $buffer < LONG$ *and falling* **then**
  - return current;
- **if** $buffer < SHORT$ *and steady* **then**
  - return formato di qualità precedente;
- **if** $buffer < CLOSE$ *and steady* **then**
  - return current;
- **if** $buffer < LONG$ *and steady* **then**
  - return formato di qualità successiva;
- **if** $buffer < SHORT$ *and rising* **then**
  - return current;
- **if** $buffer < CLOSE$ *and rising* **then**
  - return formato di qualità successiva ;
- **if** $buffer < LONG$ *and rising* **then**
  - return formato migliore;
- return current;

A.4 Dynamic SS

Algorithm 4: Pseudo-codice dell'algoritmo *Dynamic SS*

Data: τ, lastThroughputExperienced, actualLength,
Ω=[1,2,4,6,10,15], Θ=[1,2,3,4,5,6]

Result: newSegmentLength

$lastThroughputExperienced.remove(0)$;

$lastThroughputExperienced.add(\tau)$;

$\mu \leftarrow average(lastThroughputExperienced)$;

$\sigma_{weight} \leftarrow \sum_{i=1}^{M} \frac{i}{M} \mid lastThroughputExperienced(i) - \mu \mid$;

$p \leftarrow \frac{\mu}{\mu + \sigma_{weight}}$;

$\alpha \leftarrow (1-p) \cdot \Theta[max(0, actualLength - 1)]$;

$\beta \leftarrow p \cdot \Theta[min(M, actualLength + 1)]$;

$newSegmentLength \leftarrow argmin_i(\Theta[i] - (\alpha + \beta))$;

$SS' \leftarrow \Omega[newSegmentLength]$;

$eventuallyLoadID \leftarrow round(\frac{playback+buffer}{SS'})$;

$start \leftarrow eventuallyLoadID \cdot SS'$;

$\delta \leftarrow \frac{\sum_{h=start}^{start+SS'} B(h)}{SS'}$;

$random \leftarrow Random()$;

$probToSwitch \leftarrow random < (1 - \delta)$;

if *probToSwitch* **then**
    cambia la lunghezza del segmento da actualLength a newSegmentLength;

Bibliografia

[1] The Cisco. Cisco Visual Networking Index : Global Mobile Data Traffic Forecast Update , 2014 - 2019. *Growth Lakeland*, 2011(4):2010-2015, 2011.

[2] Christopher Muller, Stefan Lederer, Cristian Timmerer, 2012, *An Evaluation of Dynamic Adaptive Streaming over HTTP in Vehicular Environments*.

[3] Nicola Cranley, Philip Perry, and Liam Murphy. User perception of adapting video quality. *International Journal of Human Computer Studies* 64:637-647, 2006.

[4] Gannes, Liz, 10 June 2009, *The Next Big Thing in Video: Adaptive* Bitrate Streaming.

[5] Michael Seùfert, Sebastian Egger, Martin Slanina, Thomas Zinner, Tobias Hobfeld, and Phuoc Tran-Gia. A Survey on Quality of Experience of HTTP Adaptive Streaming. *IEEE Communications Surveys & Tutorials*, 17(1):469-492, jan 2015.

[6] S. Gouache, G. Bichot, A. Bsila, C. Howson, *Distributed & adaptive HTTP streaming*, in Proc. IEEE International Conference on Multimedia and Expo (ICME), Barcelona, Spain, July 2011.

[7] L. R. Romero, *A dynamic adaptive HTTP streaming video service for Google Android*, M.S. Thesis, Royal Institute of Technology (KTH),Stockholm, Oct. 2011.

[8] Christopher Mueller, Stefan Lederer and Christian Timmerer, *An Evaluation of Dynamic Adaptive Streaming over HTTP in Vehicular Environments*, in ACM Multimedia Systems, 2012.

[9] Big Buck Bunny Movie, https://peach.blender.org/, http://www-itec.uni-klu.ac.at/ftp/datasets/DASHDataset2014/BigBuckBunny/ .

[10] Miller, Germany Quacchio, E., Gennari, G. Wolisz, 2012, *Adaptation algorithm for adaptive streaming over HTTP*

[11] Luca Bedogni, Marco di Felice, Luciano Bononi, *Dynamic Selection of the Segment Size in HTTP Based Adaptive Video Streaming*(sottomesso, non ancora pubblicato).

[12] https://github.com/oblique/create_ap .

[13] Dung M.Nguyen, Long B. Tran, Hung T. Le, Nam Pham Ngoc, and Truong Cong Thang. An evaluation of segment duration effects in HTTP adaptive streaming over mobile networks. In *2015 2nd National foundation for Science and Technology Developement Conference on Information and Computer Science (NICS), pages 248-253. IEEE, sep 2015.*

[14] https://blog.streamroot.io/encode-multi-bitrate-videos-mpeg-dash-mse-based-media-players-22/
.

[15] Dimitrios J. Vergados, Angelos Michalas, Aggeliki Sgora, and Dimitrios D. Vergados. A fuzzy controller for rate adaptation in MPEG-DASH clients. In *2014 IEEE 25th Annual International Symposium on Personal, Indoor, and Mobile Radio Communication (PIMRC)*, pages 2008-2012. IEEE, sep 2014.

[16] Truong Thang, Quang-Dung Ho, Jung Kang, and Anh Pham. Adaptive streaming of audiovisual content using MPEG-DASH. *IEEE Transactions on Consumer Electronics*, 58(1):78-85, feb 2012.

[17] https://www.youtube.com/watch?v=6VjF638VObA .

Ringraziamenti

Vorrei ringraziare il mio correlatore Dr. Luca Bedogni per la grande disponibilità mostratami nel corso degli studi relativi alla tesi. Ringrazio la mia famiglia per il prezioso sostegno e per avermi permesso di compiere gli studi universitari.

Ringrazio tutti gli amici e le persone che mi sono state vicine, anche solo con un "in bocca al lupo"; in particolare il mio amico e compagno di corso Mattia Maldini che mi ha aiutato concretamente fornendomi parte della strumentazione richiesta dalla tesi.