Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

Un middleware multi-interfaccia per l'offloading di video dinamici auto-adattanti

Relatore:
Chiar.mo Prof.
Luciano Bononi

Presentata da:
Ciro Finizio

Correlatore:
Dott.
Luca Bedogni

Sessione II
Anno Accademico 2014-15

Indice

Indice **1**

1 Introduzione **4**

2 Stato dell'arte **6**

2.1 MPEG-DASH 6

2.1.1 Panoramica 6

2.1.2 Storia 7

2.1.3 MANIFEST: Media Presentation Description (MPD) 9

2.1.4 Un semplice esempio di streaming auto-adattante 11

2.2 TVWS 12

3 Motivazioni **14**

4 Proposta **16**

4.1 Implementazione 17

4.1.1 Classi 20

4.2 Gli Algoritmi 23

4.2.1 RANDOM 23

4.2.2 FIXED 24

4.2.3 CABA 24

4.2.4 StepByStep 26

5 Valutazione **28**

5.1 Contesto 28

5.2 Qualità media 29
5.2.1 Primo Scenario 30
5.2.2 Secondo Scenario 30
5.2.3 Terzo Scenario 31
5.3 Efficienza 32
5.3.1 Algoritmi a confronto 33
5.3.2 CABA FIXED 35
5.4 Lunghezza dei segmenti 38

6 Conclusioni **42**

Ringraziamenti **44**

Bibliografia **44**

Bibliografia **45**

Elenco delle figure **48**

Capitolo 1

Introduzione

Nel 2014 i contenuti video erano responsabili del 64% del traffico globale di Internet. Come riporta un nuovo report della Cisco, i contenuti video nel 2019 costituiranno i quattro quinti del traffico Internet[7]. Inoltre secondo un altro report, questa volta della Ericsson[16], il traffico video mobile aumenterà del 55% all'anno fino al 2020. Lo streaming auto-adattante su HTTP ricopre un ruolo importante in questo scenario, in quanto è la tecnica di streaming maggiormente diffusa: i maggiori fornitori di contenuti video online come Netflix, YouTube e Twitch utilizzano diverse implementazioni di questa tecnica, obbligando i client ad implementare un protocollo diverso per ogni fornitore di contenuti video. Cosi' *MPEG* decise di creare uno standard per uniformare lo streaming auto-adattante su HTTP: *MPEG-DASH*.

Con il seguente lavoro si vuole presentare il contesto, la progettazione, l'implementazione e la valutazione di un middleware multi-interfaccia che permetta la gestione di uno streaming auto-adattante secondo lo standard MPEG-DASH. L'applicazione si propone come un proxy che consente di sfruttare più interfacce connesse ad Internet in modo da permettere una migliore fruizione del contenuto video; usando un'interfaccia per evadere le richieste del client e le restanti verrano usate per scaricare dei segmenti video successivi in modo da creare una sorta di buffer locale nel proxy (tale tecnica è nota come prefetching). Nella progettazione si sono studiati diversi algoritmi per la scelta di prefetching. Per l'implementazione è stata scelto il linguaggio di programazzione ad oggetti C++. Con la valutazione si vuole dimostrare l'efficacia e l'efficienza delle tecniche studiate e

del proxy.

Nelle pagine che seguono si richiameranno, pertanto, gli elementi dello streaming auto-adattante su HTTP: in particolare si parlerà meglio dello strandard *MPEG-DASH*; si proseguirà quindi, ad introdurre le principali caratteristiche delle reti wireless *TVWS* usate per la valutazione; successivamente, ci si soffermerà sull'implementazione del proxy e dei diversi algoritmi di prefetching.Infine verranno presentate le valutazioni sul lavoro svolto e concluderemo lo studio con le potenzialità dell'applicazione e le possibili future espansioni.

Capitolo 2

Stato dell'arte

2.1 MPEG-DASH

Lo streaming multimedia è entrato a far parte della nostra vita di tutti i giorni, diventando uno delle maggiori parti del traffico Internet. Tuttavia non vengono sfruttate tutte le sue potenzialità: questo è dovuto in gran parte al fatto che ogni piattaforma commerciale usa i propri protocolli e formati del manifest e dei contenuti, annullando una qualunque interoperabilità tra i dispositivi ed i server dei vari fornitori. La tecnologia *DASH* (Dynamic Adaptive Streaming over HTTP), conosciuta anche come MPEG-DASH, è la prima tecnica che consente lo streaming video auto-adattante attraverso i tradizionali web server HTTP a diventare uno standard internazionale. Usando come protocollo di trasporto l'HTTP, MPEG-DASH permette a dispositivi di tutti i tipi (come smart TV, computer fissi, smartphone, tablet...) di effettuare streaming video da Internet, adattandosi alle variazioni delle condizioni di rete, fornendo *playback* di alta qualità con minime interruzioni.

2.1.1 Panoramica

Lo streaming auto-adattante(*adaptive bitrare streaming*) è una tecnica usata per lo streaming di contenuti multimediali. La larghezza di banda e la capacità di calcolo della CPU dell'utente vengono rilevate in real-time e la qualità dello stream video viene

aggiustata di conseguenza, ottenendo una diminuzione del buffering, tempo di avvio accelerato e ottimizzando la qualità della visione per gli utenti di tutti i tipi di connessioni.

Nello specifico, nella tecnica MPEG-DASH, il video viene diviso in segmenti di breve durata che saranno inviati al client con dei pacchetti HTTP. Ogni segmento viene reso disponibile in diverse qualità, *bitrates*, *codec*, codifiche audio o lingue[22]: in questo modo, durante lo streaming, il client MPEG-DASH scaricherà tra i vari segmenti alternativi quello più adatto per l'utente. Le informazioni necessarie (i segmenti disponibili e le loro caratteristiche) vengono comunicate al client tramite un manifest: un file XML. Ad esempio, il client può scegliere la qualità migliore a seconda dello stato corrente della rete che permetta di continuare la riproduzione video senza interruzioni o tempi di buffering. Se durante lo streaming dovesse diminuire il *bandwidth* della connessione, seguendo la tecnica MPEG-DASH verrebbero scaricati automaticamente i pacchetti di qualità minore, in modo che il video continui ad essere visualizzato senza interruzioni. Più in dettaglio, all'avvio, il client richiede i segmenti dello stream con bit rate minore. Se la velocità di scaricamento supera il bit rate del segmento scaricato, richiederà i segmenti con bit rate maggiore. Se, in seguito, dovesse accadere che la velocità di scaricamento diventi minore del bit rate del segmento, ciò implicherebbe una deteriorazione del *throughput* della rete e il client richiederebbe quindi i segmenti con bit rate minore.

2.1.2 Storia

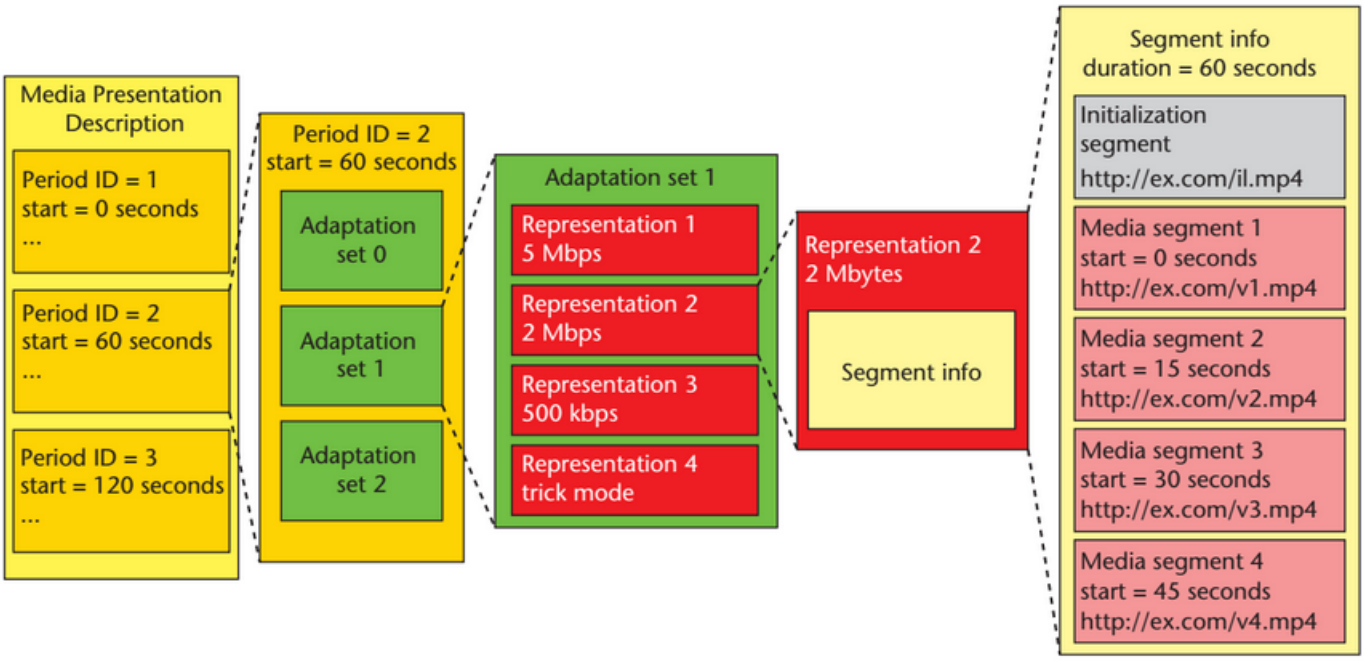
Al giorno d'oggi le tecnologie di streaming auto-adattante si basano quasi esclusivamente su HTTP. Nel passato invece, la maggior parte delle tecnologie di streaming si basavano su protocolli come RTP.

L'uso dello streaming internet è iniziato negli anni 90, con lo scopo di trasmettere tempestivamente grandi quantità di dati. Un gruppo di ricercatori chiamato Audio-Video Transport Working Group creò il protocollo per lo streaming chiamato RTP (Real-Time Transport Protocol) [22]. RTP viene usato sul protocollo UDP e prevede l'uso di sessioni di streaming per permettere il trasferimento di audio e video con un basso ritardo, mentre un altro protocollo, l'RTCP(Real-Time Transport Control Protocol) si occupa di gestire la qualità del servizio (qualità audio, qualità video). Con l'aumento degli utenti di internet RTP ha iniziato a mostrare degli svantaggi come il rendere difficile lo sviluppo

di server che permettessero lo streaming ad un grosso numero di utenti, in quanto RTP prevede una sessione dedica per ogni utente su di esso. Inoltre con il diffondersi del protocollo TCP e lo sviluppo dell'infrastruttura internet verso il support al protocollo HTTP, RTP è andato in disuso.

Si è cosi deciso di usare lo streaming attraverso HTTP che presenta numerosi benefici[22]. Prima di tutto, lo streaming HTTP ha un approccio *client-driven*: tutta la logica per l'adattamento risiede nel client, riducendo il bisogno di avere connessioni persistenti tra client e server ed evitando di dover mantenere informazioni sullo stato della sessione sul server. Questo permette di non imporre al server alcuni costi (in termini di risorse) aggiuntivi qualora il numero di client dovesse aumentare. Per di più, pacchetti non hanno problemi ad attraversare firewall (già configurati al supporto di HTTP). Per questi motivi lo streaming HTTP diventò uno degli approcci più usati e nacquero delle soluzioni proprietarie quali Adobe Systems HTTP Dynamic Streaming, Apple HTTP Live Streaming (HLS) e Microsoft Smooth Streaming. Ognuna di queste soluzioni ha un modo diverso per descrivere la struttura del video e una diversa struttura dei segmenti, cosi ogni client deve implementare un supporto diverso per ogni protocollo proprietario. Uno standard per lo streaming HTTP permetterebbe ad un client di richiedere lo streaming ad un qualunque server, permettendo cosi' l'interoperabilità tra i server ed i client di fornitori diversi. MPEG (*Moving Picture Experts Group*) nel 2009 iniziò quindi a sviluppare uno standard per lo streaming HTTP e due anni dopo pubblicò il primo strandard internazionale: MPEG-DASH.

Figura 2.1: Struttura Manifest[22]



2.1.3 MANIFEST: Media Presentation Description (MPD)

MPEG-DASH permette lo streaming auto-adattante con HTTP utilizzando dei normali web server dove risiede il contenuto diviso in due parti principali: *Media Presentation Description (MPD)* e i segmenti.

MPD è un documento XML che descrive tutte le informazioni dei segmenti, le loro caratteristiche e le informazioni per sceglie quale segmento usare[22]. Il primo elemento del manifest cioè il "Periods" descrive una parte del contenuto come il tempo di inizio e la durata: diversi periodi possono essere usati per dividere il contenuto in capitoli oppure per distinguere la pubblicità dal contenuto. Il Periods contiene il tag "Adaption Set", questo tag descrive i diversi stream disponibili, il caso piu comune è quello in cui si ha un Adaption Set per il video e piu Adaption Set per l'audio (uno per ogni lingua in cui l'audio è disponibile). L'Adaption Set a sua volta è composto dei tag "Representations", questi permettono di avere lo stesso contenuto codificato in diversi modi. Representations descrive le diverse qualità e risoluzioni video in cui è presente il contenuto e inoltre possibile specificare diversi codec in cui è disponibile il contenuto in

modo che i client possano scegliere quelli più adatti alle loro esigenze (per esempio un dispositivo che è in modalità "power save" puo' decidere di cambiare codec e selezionare il migliore per la sua modalità). Infine i Media Segments danno informazioni ai client DASH su quale sia il file reale da visualizzare, la posizione del file è descritta dal "BaseURL" per un singolo segmento, una lista di segmenti (SegmentList) o un template sulla loro struttura (SegmentTemplate). L'inizio dei segmenti e la loro durata viene descritta dal SegmentTimeline e possono essere in più file oppure in un unico file che viene letto a seconda dell'intervallo di byte desiderato.

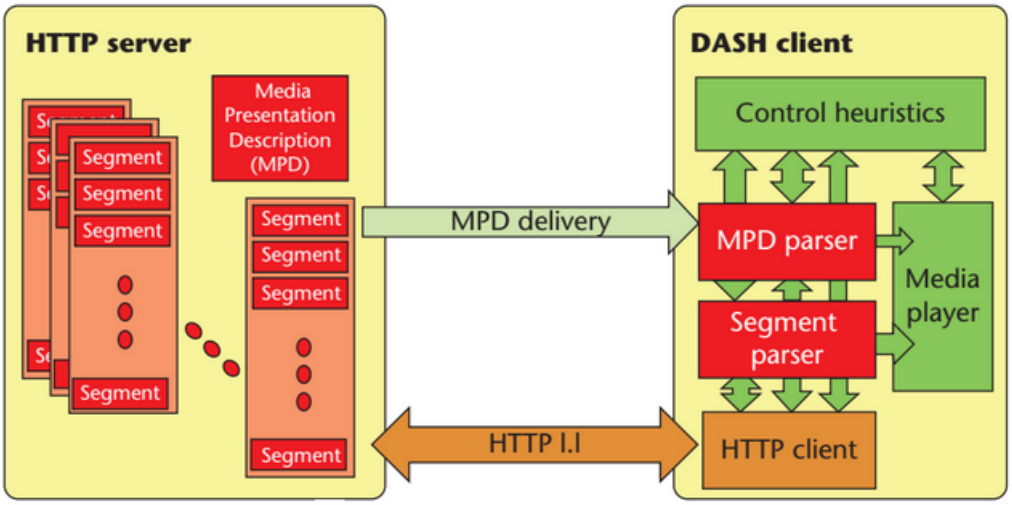
Figura 2.2: Manifest MPEG-DASH


```
<MPD minBufferTime="PT1.500000S" type="static" mediaPresentationDuration="PT0H9M55.46S" profiles="urn:mpeg:dash:profile:isoff-live:2011">
  <ProgramInformation moreInformationURL="http://gpac.sourceforge.net">
    <Title>
      dashed/BigBuckBunny_1s_simple_2014_05_09.mpd generated by GPAC
    </Title>
  </ProgramInformation>
  <Period duration="PT0H9M55.46S">
    <AdaptationSet segmentAlignment="true" group="1" maxWidth="480" maxHeight="360" maxFrameRate="24" par="4:3">
      <SegmentTemplate timescale="96" media="bunny_$Bandwidth$bps/BigBuckBunny_1s$Number$.m4s" startNumber="1" duration="96"
      initialization="bunny_$Bandwidth$bps/BigBuckBunny_1s_init.mp4"/>
      <Representation id="320x240 47.0kbps" mimeType="video/mp4" codecs="avc1.42c00d" width="320" height="240" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="46980"/>
      <Representation id="320x240 92.0kbps" mimeType="video/mp4" codecs="avc1.42c00d" width="320" height="240" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="91917"/>
      <Representation id="320x240 135.0kbps" mimeType="video/mp4" codecs="avc1.42c00d" width="320" height="240" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="135410"/>
      <Representation id="480x360 182.0kbps" mimeType="video/mp4" codecs="avc1.42c015" width="480" height="360" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="182366"/>
      <Representation id="480x360 226.0kbps" mimeType="video/mp4" codecs="avc1.42c015" width="480" height="360" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="226106"/>
      <Representation id="480x360 270.0kbps" mimeType="video/mp4" codecs="avc1.42c015" width="480" height="360" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="270316"/>
      <Representation id="480x360 353.0kbps" mimeType="video/mp4" codecs="avc1.42c015" width="480" height="360" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="352546"/>
      <Representation id="480x360 425.0kbps" mimeType="video/mp4" codecs="avc1.42c015" width="480" height="360" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="424520"/>
      <Representation id="854x480 538.0kbps" mimeType="video/mp4" codecs="avc1.42c01e" width="854" height="480" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="537825"/>
      <Representation id="854x480 621.0kbps" mimeType="video/mp4" codecs="avc1.42c01e" width="854" height="480" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="620705"/>
      <Representation id="1280x720 808.0kbps" mimeType="video/mp4" codecs="avc1.42c01f" width="1280" height="720" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="808057"/>
      <Representation id="1280x720 1.1Mbps" mimeType="video/mp4" codecs="avc1.42c01f" width="1280" height="720" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="1071529"/>
      <Representation id="1280x720 1.3Mbps" mimeType="video/mp4" codecs="avc1.42c01f" width="1280" height="720" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="1312787"/>
      <Representation id="1280x720 1.7Mbps" mimeType="video/mp4" codecs="avc1.42c01f" width="1280" height="720" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="1662809"/>
      <Representation id="1920x1080 2.2Mbps" mimeType="video/mp4" codecs="avc1.42c032" width="1920" height="1080" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="2234145"/>
      <Representation id="1920x1080 2.6Mbps" mimeType="video/mp4" codecs="avc1.42c032" width="1920" height="1080" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="2617284"/>
      <Representation id="1920x1080 3.3Mbps" mimeType="video/mp4" codecs="avc1.42c032" width="1920" height="1080" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="3305118"/>
      <Representation id="1920x1080 3.8Mbps" mimeType="video/mp4" codecs="avc1.42c032" width="1920" height="1080" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="3841983"/>
      <Representation id="1920x1080 4.2Mbps" mimeType="video/mp4" codecs="avc1.42c032" width="1920" height="1080" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="4242923"/>
      <Representation id="1920x1080 4.7Mbps" mimeType="video/mp4" codecs="avc1.42c032" width="1920" height="1080" frameRate="24" sar="1:1" startWithSAP="1" bandwidth="4726737"/>
    </AdaptationSet>
  </Period>
</MPD>
```

2.1.4 Un semplice esempio di streaming auto-adattante

Supponiamo che un dispositivo inizi lo streaming del contenuto desiderato richiedendo i segmenti del video alla qualità disponibile più alta. Dopo lo streaming dei primi segmenti, e monitorando il bandwidth effettivo della rete, il dispositivo si rende conto che il bandwidth effettivamente disponibile è minore del bitrate corrispondente alla qualità richiesta. Di conseguenza, appena gli è possibile, cambia la qualità del video ad un bitrate minore richiedendo i segmenti dela traccia di qualità media. Il dispositivo continua a monitorare il bandwidth effettivo e realizza che è diminuito ancora, scendendo sotto il bitrate corrente: per mantenere la continuità del playback, richiede i segmenti di bitrate ancora inferiore. Continua lo streaming del contenuto in questa qualità finché il bandwidth non aumenta: a questo punto cambia di nuovo la qualità del video, richiedendo segmenti con bitrate maggiore. Questo esempio è uno dei casi d'uso più semplici dello streaming dinamico di contenuti multimediali. Altre applicazioni più avanzate sono, tra le altre, lo streaming di contenuti multimediali 3D, streaming di video con sottotitoli e didascalie, inserzioni di pubblicità dinamiche, e streaming live a bassa latenza.

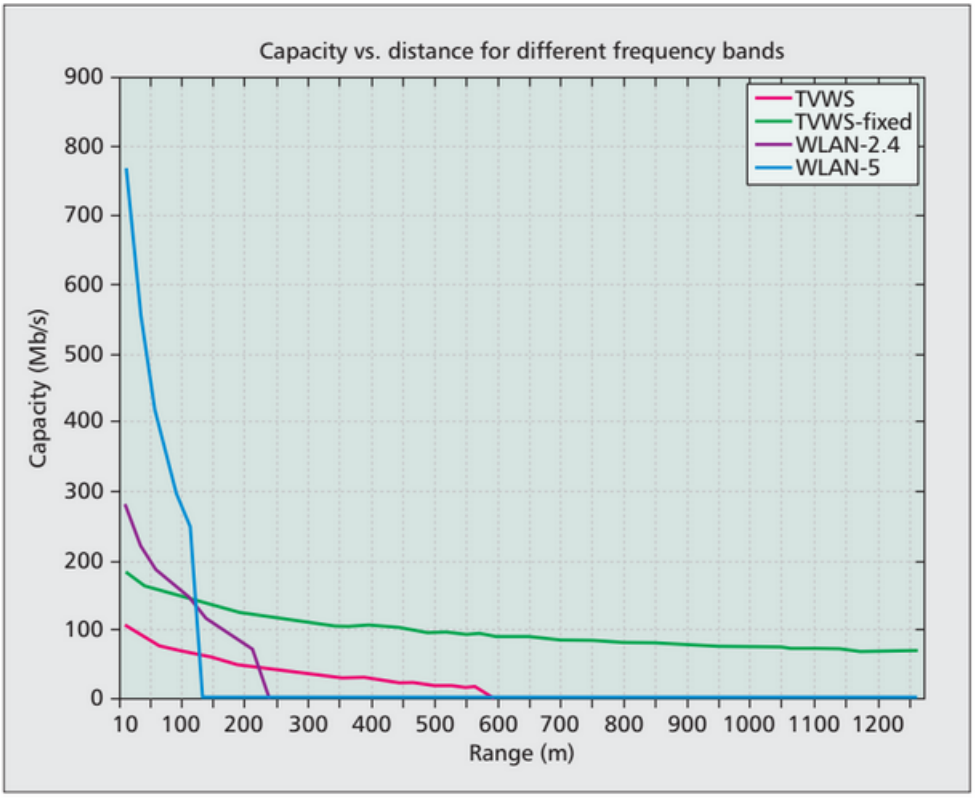
Figura 2.3: MPEG-DASH struttura[22]



2.2 TVWS

Negli ultimi anni, la crescita esponenziale di traffico dati wireless ha portato alla luce il problema della scarsità di spettro radiofrequenziale. Di fatto, al giorno d'oggi, la maggiorparte dello spettro è già allocato in maniera statica o occupato da qualche soggetto licenziatario per vari servizi, come per esempio il digitale terrestre. Per questo motivo le bande inutilizzate sono rare. Una delle soluzioni possibili per contrastare questa carenza sono le reti radio cognitive che riutilizzano alcune parti dello spettro senza causare interferenze dannose al soggetto licenziatario, aumentando la disponibilità dello spettro. Un caso particolarmente interessante è quello dell'utilizzo delle reti radio cognitive sui cosiddetti TV White Spaces (TVWS)[2]. I TVWS sono delle porzioni di banda tra un frequenza assegnata e l'altra del broadcasting televisivo che non vengono utilizzate per evitare problemi di interferenza. Si trovano nelle bande VHF e UHF, e presentano caratteristiche che le rendono ottimali per le comunicazioni wireless. Innanzitutto, le tecnologie TVWS hanno una zona di copertura molto più ampia di quella di un router Wi-Fi tradizionale (all'incirca dieci chilometri di diametro invece di cento metri[2]). Oltre a ciò, possiedono la capacità di penetrare ostacoli quali alberi, edifici e terreni irregolari. Mentre le connessioni a microonde richiedono line-of-sight (LOS) tra i due punti connessi, rendendole una soluzione costosa e a volte irrealizzabile in aree con terreno aspro e forestato, le tecnologie TVWS permettono di penetrare ostacoli e coprire terreno irregolare senza necessitare infrastrutture aggiuntive grazie all'utilizzo delle frequenze UHV. Questo si rivela un vantaggio anche nelle comunicazioni indoor, come nel caso di abitazioni, permettendo una migliore propagazione attraverso i muri.

Figura 2.4: Confronto tra TWVS e Wi-Fi[2]



Capitolo 3

Motivazioni

Come già indicato nell'introduzione, i contenuti video ricoprono attualmente il 64% del traffico Internet mondiale; inoltre, diversi studi condotti dalla Cisco[7]e dalla Ericsson[16] prevedono un aumento esponenziale del traffico video mobile, dovuto in gran parte ai servizi di streaming video e la prevalenza crescente di contenuti video online. Per di più entro il 2020, il 90% della popolazione mondiale sarà coperta da reti mobili. I servizi di streaming, diventando sempre di più la preferenza degli utenti, avranno un ruolo crescente nel futuro di Internet, in particolare tecniche di streaming auto-adattante come MPEG-DASH che permettono di migliorare la qualità del servizio fornito all'utente.

L'utilizzo di internet sui device mobile avviene in due principali modalità:

- Un utilizzo mobile dove l'utente è la maggior parte del tempo in movimento (come ad esempio lunghi viaggi in auto)
- Un utilizzo nomade in cui l'utente effettua dei piccoli spostamenti (per esempio da casa al luogo di lavoro e viceversa) per poi fermarsi in un luogo per più tempo.

Si è visto le reti mobili vengono usate per la maggior parte, in modalità nomade, cosi l'utilizzo outdoor rispetto a quello indoor e di gran lunga inferiore. Inoltre uno studio effettuato a Londra dimostra come gli utenti mobili preferiscano usare sempre di più il 4G piuttosto che il Wi-Fi: in effetti il 65% dell'utilizzo di dispositivi mobili avviene indoor.

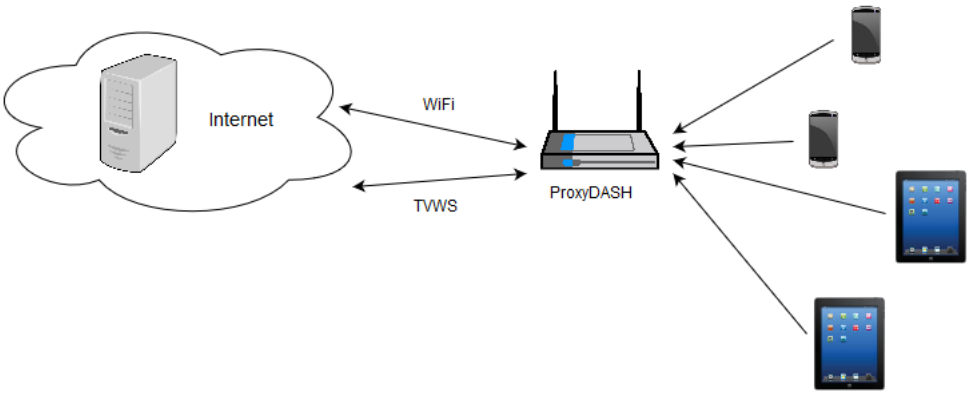
Questo comporta delle problematiche come il cosiddetto shadow fading: ostacoli quali muri e edifici circostanti possono influenzare negativamente la propagazione delle onde, attenuando cosi' il segnale[20]. Tuttavia ci sono scenari futuri possibili che prevedono nuove tecnologie per l'accesso a Internet wireless. Uno degli scenari di interesse maggiore è quello che prevede l'utilizzo dei TVWS, di cui abbiamo parlato nel primo capitolo, come rete di supporto per il 4G. Qui nasce l'idea di creare un sistema middleware per lo streaming auto-adattante che permetta di sfruttare diverse interfaccie. In questo modo, si riesce a migliorare la qualità dello streaming usando tutte le interfacce di rete disponibili. Ad esempio, un utente che si trova in un'area urbana, in cui lo shadowing è particolarmente elevato, può usare il proxy per fornire un supporto alla rete 4G con il TVWS se disponibile, evitando cosi' il peggioramento della connettività. In aree rurali invece, dove le connessioni ADSL potrebbero non essere sufficientemente performanti, si potrebbe sopperire a questa mancanza sfruttando oltre all'ADSL la rete 4G per migliorare l'esperienza dell'utente nello streaming.

Capitolo 4

Proposta

Il progetto consiste in un middleware multi interfaccia che permetta di migliorare la fluidità e la qualita dello streaming con lo standard MPEG-DASH. Il middleware si occuperà della gestione delle richieste HTTP che riguardano uno streaming MPEG-DASH mentre le altre richieste verrano lasciate passare in maniera trasparente. Per ogni nuova richiesta di streaming il proxy si occupera di gestirla.

Figura 4.1: Infrastruttura



4.1 Implementazione

Il proxy è stato implementato usando il linguaggio C++ e i socket per rendere possibile la comunicazione con il client e con il server.

Il proxy resterà in ascolto su di una porta prefissata su la quale il client farà le sue richieste. Alla ricezione di una richiesta da parte del client il proxy analizza l'header HTTP della richiesta, se la richiesta è di tipo MPEG-DASH allora proverà a gestirla in modo appropriato altrimenti lascierà passare la richiesta in modo trasparente cioè si occupera di mettere in comunicazione il client con il server e far passare i dati richiesti senza che esso ne apporti alcuna modifica o interferenza.

Nel caso in cui la richiesta riguardi uno streaming MPEG-DASH il proxy è stato studiato per prevedere delle "sessioni di streaming": ogni differente richiesta di streaming avrà un thread con le informazioni necessarie per gestirla. Quindi per ogni richiesta MPEG-DASH il proxy cerca tra le sessioni di streaming già avviate se ne esiste una per quella richiesta, se la trova passa la gestione al thread che se ne occupa altrimenti controlla la richiesta appena ricevuta, se la richiesta riguarda un manifest MPEG-DASH allora avvierà un nuovo thread per gestirla in caso contrario sarà lasciata passare in modo trasparente.

Al primo avvio il thread gestirà quindi la richiesta del manifest, aprirà la connessione con il server e iniziera il trasferimento del manifest dal server al client. Mentre il manifest viene reinviato al client il thread lo salva in una variabile temporanea, che una volta finito il download e il reinvio al client servirà ad analizzare l'xml e salvare le informazioni necessarie per la corretta gestione dello streaming. La fase di *parser* del manifest viene fatta usando una libreria di supporto chiamata "pugixml" che permette di parserare l'xml e creare delle query XPath. Si è scelto di usare "pugixml" rispetto alle altre tante soluzioni presenti nel campo del *parsering* XML in C++ perchè presenta il vantaggio di essere particolarmente performante nella fase di *parser* e creazione del "DOM Tree". Inoltre, permette la lettura di XML anche da *buffer* che è il caso in cui si presenta l'XML nel thread. A questo punto il nuovo processo prenderà le informazioni necessarie alla gestione corretta dello streaming dal manifest e le memorizzerà in modo da avere tutte le informazioni base sulla struttura del video. In particolare le informazioni neccessarie al processo sono:

- Lista delle qualità in cui è disponibile il contenuto;
- Numero di segmenti che compongono il video;
- Durata media di ogni segmento;
- Path che descriva la posizione dei segmenti nelle diverse qualità;

Una volta acquisite le informazioni riguardo la struttura del video il processo setterà il suo stato su ready in modo da poter iniziare la fase di streaming e prefetching. Da questo momento in poi il processo rimmarrà in attesa di essere risvegliato da una nuova richiesta del client. Una volta arrivata una nuova richiesta il processo principale si occupera di risvegliare il thread usando un meccanismo di signal il tutto correlato da un meccanismo di mutua esclusione che permette di evitare deadlock. In particolare le fasi principali del ciclo del thread sono:

1. Aspetta di essere risvegliato da una nuova richiesta.
2. Controlla lo stato dei figli precedentementi creati in modo da rilevare eventuali interfaccie libere e non più in uso;
3. Sceglie l'interfaccia migliore usando il througput medio precendentemente registrato per ogni interfaccia, l'interfaccia migliore verra usata per evadere la richiesta del client mentre le altre interfaccie libere verrano usate per la fase di prefetching;
4. Aggiorna la lista una lista contente tutti i segmenti in modo da aggiornare lo stato del segmento appena richiesto e quello degli eventuali segmenti finiti con il meccanismo di prefetching;
5. Crea un nuovo processo usando una fork che si occupa di gestire la richiesta del client;
6. Per ogni interfaccia libera sceglie con un algoritmo di prefetching il numero di segmento migliore e crea, usando una fork, nuovo processo che si occupa di gestire il download del segmento scelto. Salva quindi le informazioni riguardo il processo appena creato e aggiorna il segmento scelto nella lista dei segmenti settando il suo stato su "downloading"

7. Torna in attesa di una nuova richiesta;

Il proxy, quindi, per effettuare la gestione dello streaming farà uso oltre che di un thread principicale, anche di due tipologie diverse di processi creati tramite l'uso di *fork*.

La prima tipologia di processi è quella che si occupa di gestire la richiesta del client. Per prima cosa viene controllato nella lista dei segmenti lo stato del segmento richiesto che può avere tre diversi stati: non scaricato, in scaricamento, già scaricato.

Nel caso lo stato del segmento sia *non scaricato* il processo crea un socket impostando l'IP letto dalla richiesta, lo lega all'interfaccia precedentemente scelta e invia una copia della richiesta ricevuta dal client al server. Quando il server risponde iniziando l'invio dei pacchetti il processo inizierà a ricevere i pacchetti e li invierà al client, durante questa fase la struttura dei pacchetti non viene modifica e vengono raccolti i dati utili ai fini statistici. Una volta terminato il trasferimento il processo aggiorna il file contentenete le statistiche aggiungendo quelle del trasferimento appena eseguito e termina la sua esecuzione.

Nel caso lo stato del segmento sia *in scaricamento* il thread provvederà a chiudere il processo che lo sta scaricando ed eliminare il file associato al segmento; dunque continuerà la sua esecuzione come nel caso precedente. Si è scelto di chiudere il processo invece che introdurre un meccanismo di attesa in quanto i nuovi processi vengono creati mediante l'uso di *fork* che rendono problematica una efficiente sincronizzazzione tra il processo padre e quello figlio; inoltre in quanto si sta effettuando streaming video questa soluzione e preferibile per evitare eventuali blocchi su altre interfaccie più lente e meno efficienti.

Mentre nell'ultimo caso, cioè quello in cui lo stato del segmento sia *scaricato*, il processo non si deve preoccupare di instaurare una connessione con il server ma dovrà leggere il contenuto del file associato al segmento e inviarlo al client. Il file conterrà l'header di risposta HTTP ricevuta dal server come prima parte, mentre il resto del file sarà il contenuto del segmento. Una volta terminato l'invio del contenuto del file anche in questo caso verranno aggiornati il file contenten le statistiche e il processo potrà terminare la sua esecuzione, il file non verrà eliminato in questa fase ma solo in un secondo momento dal thread principale per evitare che per qualche problema nella connessione il client lo possa richiedere.

La seconda tipologia di processo che si crea è qeulla dei processi che si occupano del download del segmento in prefetching per prima cosa crea un copia locale dell'header

HTTP e del corpo dell'ultima richiesta GET effettua dal client, lo modifica aggiornando il numero di pacchetto da modificare e,nel caso sia richiesto dall'algoritmo di prefetching o dall'utente, ne modifica anche la qualità. A questo punto anche esso puo aprire un nuovo socket associandolo con l'interfaccia scelta e iniziare la comunicazione con il server solo che in questo caso il contenuto ricevuto sarà salvato su un file locale in attesa di dover essere utilizzato. Anche questo processo raccoglie le informazioni necessarie ai fini statistici e alla fine della ricezione del contenuto aggiorna il file delle statistiche.

Ognuno di questi processi inoltre si occupa anche di aggiornare la velocità dell'interfaccia appena usata in modo che il thread possa scegliere sempre l'interfaccia migliore da dover usare per gestire la richiesta del client e cosi' non provocare dei rallentamenti sullo streaming. Per questo viene usato una classe statica con un area di memoria condivisa in modo che tutti i processi possano aggiornare lo stato delle interfaccie, inoltre la classe possiede un meccanismo di semafori che permettedi gestire la *mutua esclusione* tra processi.

4.1.1 Classi

Le classi usate nel progetto sono:

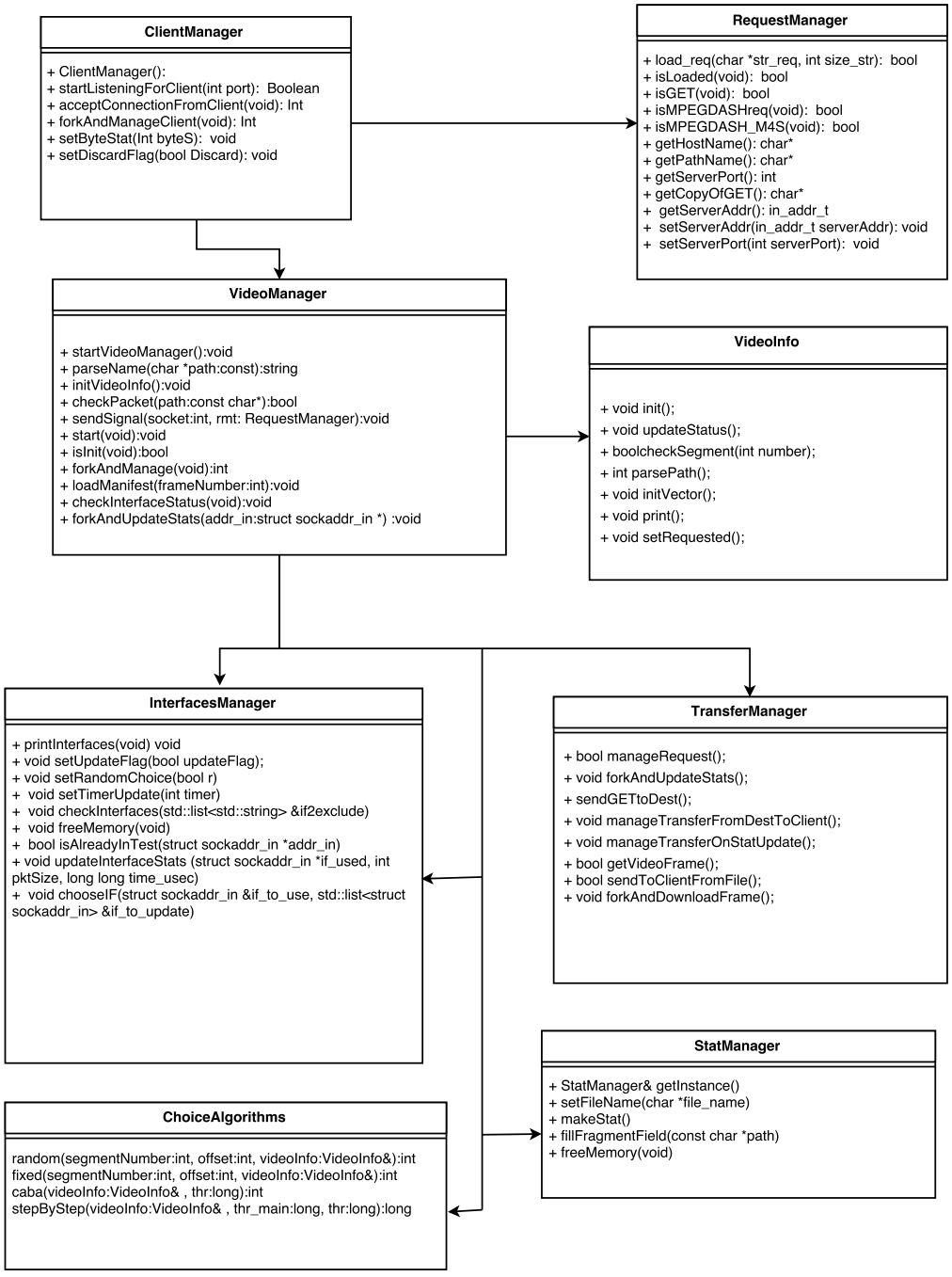
- VideoManager: classe main del thread che si occupa della gestione base dello streaming. In particolare accetta le richieste del client, le evade nel modo corretto e gestisce le interfacce libere per il prefetching.

- VideoInfo: classe che contiene tutte le informazioni base dello streaming in corso. La sua funzione principale è quella di tener traccia dello stato dei segmenti del video e sincronizzarli con i segmenti in stato di download attraverso il meccanismo di prefetching. Questa classe comprende inoltre i dati presi dal manifest MPEG-DASH come un array delle qualità in cui è disponibile il contenute e il path che permette di cambiare numero di segmento e qualità.

- TransferManager: classe con metodi statici che servono per gestire la connessione e il trasferimento delle richieste al server. In questa classe vengono implementate tutte le funzionalità per la comunicazione con il server, il download su di un file temponae e la lettura e l'invio di del contenuto di un file al server.

- ChoiceAlgorithms: classe con metodi statici che contiene l'implementazione di tutti i vari algortmi di prefetching esposti nella sezione successiva.
- StatManager: classe che si occupa di creare i file di log contente le informazioni delle richieste di ogni singolo segmento, sia che quelle effetuate dal client che quelle effettuate per il prefetching. Viene crato un file di log diverso per ogni interfaccia in modo da impedire race conditions di diversi processi sullo stesso file.
- InterfaceManager: classe che si occupa di gestire lo stato dell interfaccie. Essa presenta in un area di memoria condivisa una lista delle interfaccie contente, per ogni interfaccia, il nome, lo stato (cioè un flag che indica se qualche altro processo sta usando l'interfaccia) e una lista contentente le informazione degli ultimi pacchetti TCP ricevuti in modo da poter calcolare lo stato della rete.

Figura 4.2: Diagramma delle classi



4.2 Gli Algoritmi

Una parte molto importante riguarda la scelta del segmento da scaricare in prefetching, ci sono più algortmi possibili per effettuare il prefecthing, ognuno con i suoi punti di forza e le sue debolezze. Come vedremo nelle valutazioni non esiste un algortimo migliore rispetto ad un altro ma ognuno ha dimostrato una sua forza in differenti scenari.

Per evitare problemi nella scelta dello stesso segmento per ogni nuova fase di prefetching ogni algoritmo prevede, al termine della scelta del numero di segmento da scaricare, un controllo sullo stato del segmento scelto. Se lo statto del segmento scelto dovesse risultare come gia scaricato allora si sceglierà il segmento e si ripeterà il controllo fino a quando non si troverà un segmento non ancora scaricato.

Per tutti gli algoritmi è previsto un ciclo che permette di controllare lo stato del segmento scelto in modo che se il segmento scelto sia già stato scaricato non venga riscelto per il prefetching.

4.2.1 RANDOM

Random rappresenta uno degli algoritmi più intuitivi e più facile da progettare.

Esso consiste nella scelta di un segmento casuale compreso in un intervallo di valori che va dal numero dell'ultimo segmento scelto dal client ad un valore massimo settabile nelle impostazioni del proxy.

Algorithm 4.1 Random

```
1 int ChoiceAlgorithms::random(int segmentNumber, int offset, VideoInfo&
    videoInfo){
2     int to_ret = 0;
3     do{
4         to_ret = segmentNumber + ( std::rand() % ( offset + 1 ) );
5     }while(videoInfo.checkSegment(to_ret));
6     return to_ret;
7 }
```

4.2.2 FIXED

L'algoritmo CABA Fixed prevede la scelta del segmento da scaricare in base ad un offset prefissato, sia l l'ultimo segmento scelto dal client e o l'offeset scelto allora il numero di segmento da scaricare, cioè n, sarà uguale a:

$n \leftarrow l + o$

Come si puo facilmente intuire il corretto funzionamento di questo algoritmo dipende dall'offset scelto, infatti l'offset ottimale potra cambiare a seconda dello scenario in cui ci si trova. Non sempre è possibile conoscere a priori il giusto offset per lo scenario in cui ci trova, inoltre potrebbe non rimanere fisso per tutta la durata dello streaming ma evolversi con lo scenario.

Algorithm 4.2 Fixed

```
1 int ChoiceAlgorithms::fixed(int segmentNumber, int offset, VideoInfo&
      videoInfo){
2     int to_ret = 0;
3     to_ret =    segmentNumber + offset;
4     while(videoInfo.checkSegment(to_ret)){
5         to_ret++;
6     }
7     return to_ret;
8 }
```

4.2.3 CABA

Per ovviare ai problemi presentati dall'alogoritmo precendete (CABA Fixed) si è scelto di implementare un soluzione che sia ingrando di adattarsi alla scenario e scegliere ogni volta l'offset migliore.

CABA punta a scegliere l'offset migliore tenendo conto sia delle condizioni delle rete che delle dimensioni e della durata dei segmenti. Infatti calcola il numero di segmento da scaricare utilizzando l'ultimo troughput rilevato per l'interfaccia e la dimensione del segmento nell'ultima qualità richiesta dal client. In particolare, sia $C = \{I_1, \ldots, I_N\}$ l'insieme delle connessioni disponibili. Dalle precedenti misurazioni abbiamo definito $\Delta = \{\delta_1, \ldots, \delta_N\}$ come l'insieme delle velocità medie per ogni connessione. Dal manifest

conosciamo la dimensione media del segmento S e la lunghezza L . Supponiamo che il client richiede il segmento di indice j al momento T e come detto precedentemente selezioniamo la migliore connessione disponibile per scaricare il segmento, più precisamente $i = index(max(\Delta))$. Per cui il meccanisco di prefetching deve scegliere nell'insieme $C' = C - I_i$ quale segmento scaricare su ogni elemento dell'insieme.

Per cui

$$\forall I_i^c \epsilon C' : E_i = T + \left(\frac{S}{\delta_i}\right)$$

Possiamo cosi calcolare il segmento da scaricare prima che il client finisca come

$$F = j + \limsup\left(\frac{\limsup(E_i)}{L}\right)$$

Nonostante CABA cerchi di risolvere i problemi presentati dall'algoritmo precendete anche esso presenta degli svantaggi che possono essere dovuti ad una mal corretta gestione da parte del client. Questo discorso verrà approfondito in dettaglio nella sezione Valutazioni.

Algorithm 4.3 CABA

```
1  int ChoiceAlgorithms::caba( VideoInfo& videoInfo, long thr){
2      //if Thr NaN return last + 1
3      if(thr != thr)
4          return videoInfo.getLastRequest() + 1;
5
6      long sizeSegment = ((videoInfo.getLastReques_bps() * videoInfo.
           getSegmentDuration())/ 8000) ;
7      long expectedTime = sizeSegment/thr;
8      int segmentNumber = videoInfo.getLastRequest() + ceil(expectedTime/
           videoInfo.getSegmentDuration());
9      if(segmentNumber > videoInfo.getSegmentNumber())
10         return 0;
11     return segmentNumber + 2;
12 }
```

4.2.4 StepByStep

L'algortimo StepByStep a differenza di quelli precedentemente analizzati si propone di ottenere una migliore fluidità delle streaming a discapito della qualità del video. L'idea dell'algortimo è quella di cercare di rendere la visualizzazzione del video il più fluida possibile andando ad incentra la scelta non più sul numero di segmento da scaricare ma bensi sulla qualità a cui scaricarlo.

Infatti l'algoritmo come numero di segmento da scaricare sceglie sempre il segmento successivo a quello richiesto dal client mentre incentra la ricerca su quale sia la qualità migliore possibile in cui scaricarlo. Con qualità migliore possibile si intende la migliore qualità che abbia un tempo stimato di download minore del tempo stimato di ricezione della prossima richiesta del client. Come nell'algoritmo precedente, sia $C = \{I_1, \ldots, I_N\}$ l'insieme delle connessioni disponibili. Dalle precedenti misurazioni abbiamo definito $\Delta = \{\delta_1, \ldots, \delta_N\}$ come l'insieme delle velocità medie per ogni connessione. Dal manifest conosciamo l'insieme contente i *bitrate* $S = \{s_1, \ldots, s_m\}$ e la lunghezza L . Supponiamo che il client richiede il segmento di indice j al momento T e come detto precedentemente selezioniamo la migliore connessione disponibile per scaricare il segmento, più precisamente $i = index(max(\Delta))$. Per cui l'algoritmo *StepByStep* deve calcolare il tempo medio di scaricamento di ogni *bitrate* e successivamente scegliere il migliore.

Per cui

$$E_i^k = T + \left(\frac{S_k}{\delta}\right)$$

Possiamo cosi calcolare la miglior qualità da scaricare prima che il client finisca come

$$\max(k) \; t.c. \; E_i^k \leq L$$

Solo nel caso in cui non venga trovato nessuna qualità che finisca nel tempo richiesto allora l'algoritmo passa la ricerca al segmento successivo aumentando la stima del tempo di ricezione delle prossima richiesta da parte del client. StepByStep potrebbe risulta un ottima scelta nel caso di scenari particolari e un modo completamente diverso di approcciarsi al problema del prefetching spostando il problema della ricerca di quale segmento scarica a quale sia la qualità migliore da poter scaricare.

Algorithm 4.4 StepByStep

```
1  long ChoiceAlgorithms::stepByStep( VideoInfo& videoInfo, long thr_main,
       long thr){
2      long bestQual = 0;
3      if(thr != thr)
4          return 0;
5      long sizeSegment = ((videoInfo.getLastReques_bps() * videoInfo.
           getSegmentDuration())/ 8000);
6      long expectedTime = sizeSegment/thr_main;
7      long expectedTimeLocal = 0;
8      for (std::vector<long>::iterator it=videoInfo.qualityArray.begin(); it
           != videoInfo.qualityArray.end(); ++it){
9          long sizeSegmentLocal = (((*it) * videoInfo.getSegmentDuration())/
               8000) ;
10         expectedTimeLocal = sizeSegmentLocal/thr;
11         if(expectedTimeLocal < expectedTime){
12             bestQual = *it;
13         } else
14             break;
15         }
16     return bestQual;
17 }
```

Capitolo 5

Valutazione

Come già detto, nessuno dei diversi algoritmi presenta una soluzione migliore rispetto agli altri in modo assoluto, anzi, come si cerca di dimostrare in questa sezione, questa scelta non è banale e strettamente dipendente dallo scenario in cui ci si trova. Procederemo quindi con la valutazione di due fattori principali: la qualità media e l'efficienza .

5.1 Contesto

La valutazione dei vari algoritmi e tutti i test ad essa annessi sono stati svolti in un ambiente di simulazione usando dati raccolti in scenari reali.

In particolare la maggior parte delle simulazioni è avvenuta basandosi sui dati raccolti nella Queen Mary University in Inghilterra. Oggetto delle simulazioni sono stati due diversi tipi di tecnologie wireless: il WiFi tradizionale e il TVWS. I principali scenari utilizzati nella simulazione sono tre:

- Nel primo scenario le rilevazioni sono state effettuate ad una distanza di circa 10 metri dall'access point e le velocità rilevate sono state di 750KBs per il TVWS e di 1.5 MBps per il WiFi
- Nel secondo scenario le rilevazioni sono state effettuate ad una distanza di circa 14 metri dall'access point e le velocità rilevate sono state di 450KBs per il TVWS e di 1.8 MBps per il WiFi

- Nel terzo scenario le rilevazioni sono state effettuate ad una distanza di circa 30 metri dall'access point e le velocità rilevate sono state di 450KBs sia per il TVWS che per il WiFi

I dati che andremmo analizzare di seguito sono stati creati effettuando, mediamente, quindici simulazioni per ogni scenario ognuna dalla durata di trecento secondi (cinque minuti) usando sempre lo stesso client MPEG-DASH. Il video utilizzato per le simulazioni è "Big Buck Bunny" un open movie project avviato dalla Blender Foundation, disponibile in segmenti di diverse lunghezze. Per le simulazioni si è scelto di utilizzare il video diviso in segmenti da un secondo.

Le simulazioni sono state effettuate con l'uso di tc (traffic control), un tool linux che permette, tra le altre funzionalità, anche la possibilità di limitare la velocità delle interfacce.

Si proverà ora a mettere in correlazione i risultati delle simulazioni e dei vari algoritmi con i diversi scenari e dimostrare l'efficienza , i punti di forza ed i punti di debolezza dei vari algoritmi.

5.2 Qualità media

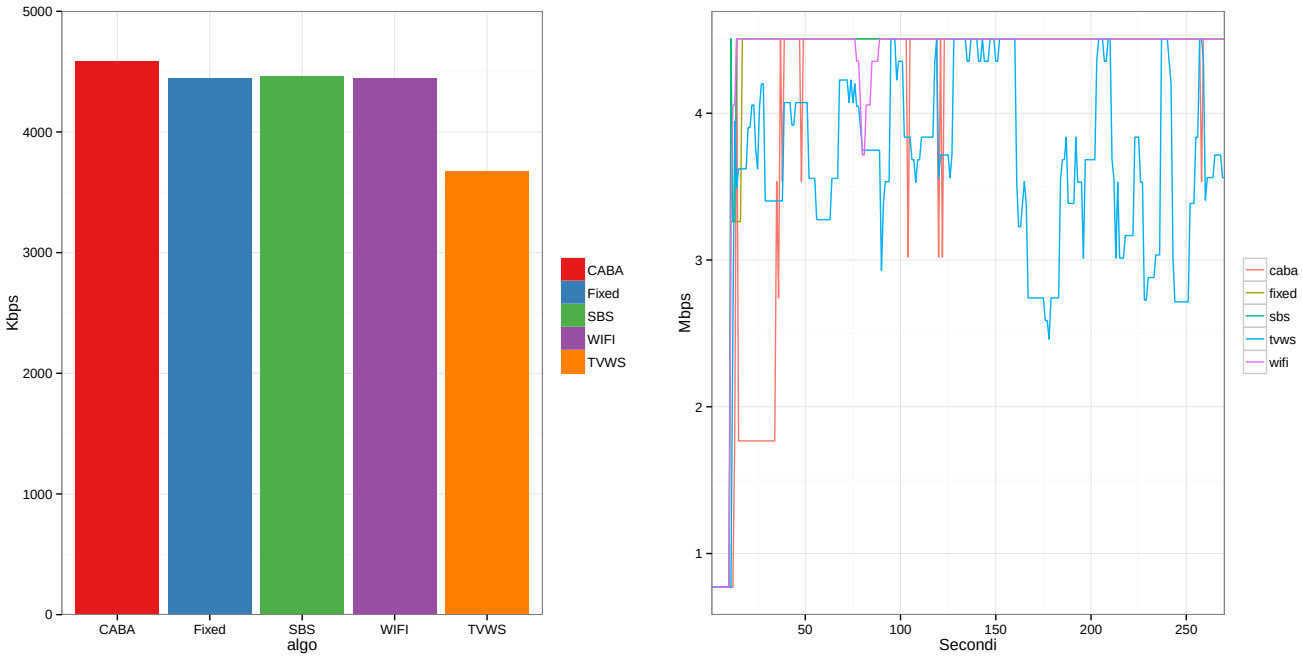
Un fattore importante da tenere in considerazione riguarda sicuramente quello della qualità di visualizzazione del contenuto. Anche se è il client MPEG-DASH a gestire la qualità, con il corretto utilizzo del prefetching il client tende ad aumentare la qualità richiesta in quanto l'invio dei segmenti gia scaricati avviene in un tempo minore rispetto al normale download: questo da l'impressione al client di avere a disposizione una connessione migliore, quindi lo porta ad aumentare la qualità scelta.

In generale l'algoritmo CABA ottiene dei buoni risultati in quanto a qualità media in tutti e tre gli scenari, andando a migliorare la qualità media risultante dall'utilizzo di una sola interfaccia.

5.2.1 Primo Scenario

Nel primo scenario abbiamo il TVWS con una velocità media di 750KBps ed il Wi-Fi con una velocità media di 1.5 MBps. In quanto la velocità di entrambe le connessioni risulta elevata, si viene a creare un'equivalenza della qualità media tra solo Wi-Fi, CABA Fixed e StepByStep mentre come aspettato la qualità rilevata con solo l'interfaccia TVWS risulta più bassa in quanto più lenta. CABA invece sembra apportare un leggero miglioramento della qualità media.

Figura 5.1: Qualità media primo scenario



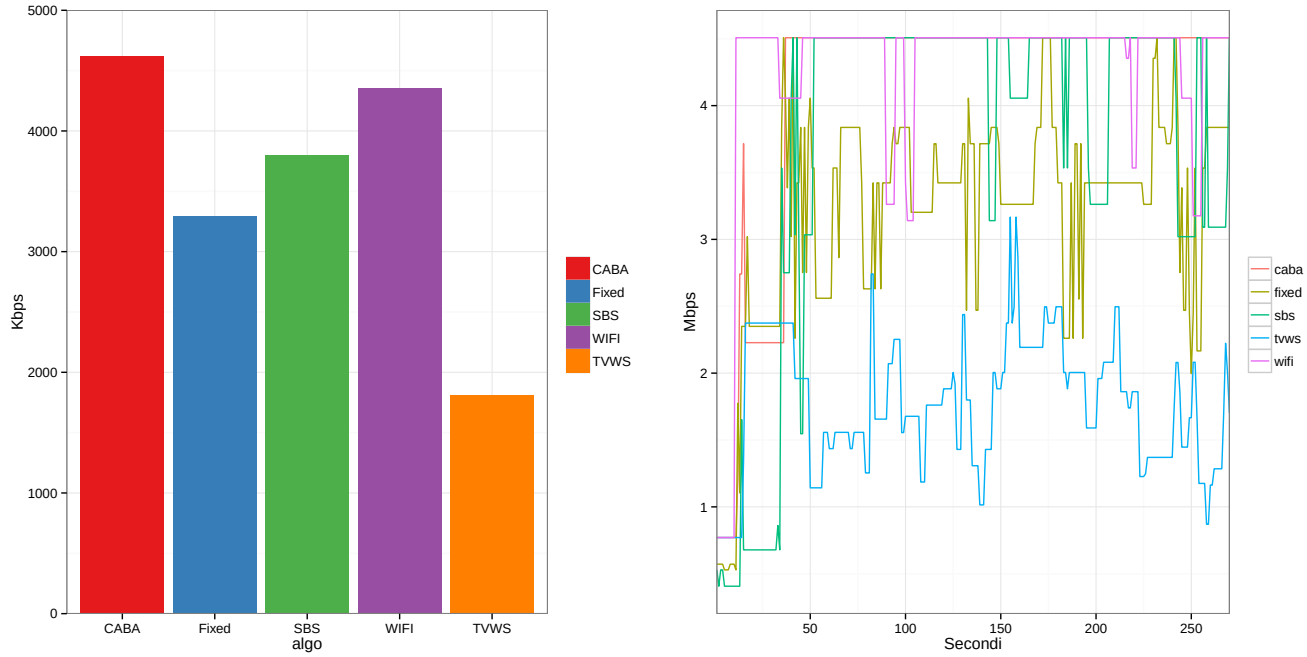
5.2.2 Secondo Scenario

Il secondo scenario vede l'interfaccia Wi-Fi molto più veloce rispetto a quella TVWS, infatti come possiamo vedere la qualità media dell'interfaccia TVWS è inferiore rispetto la qualità media dell'altra interfaccia.

Anche in questo caso la qualità di CABA è superiore, anche se di poco, rispetto a quella della singola interfaccia Wi-Fi. L'algoritmo StepByStep per sua natura tende ad avere una qualità minore per garantire un fluidità nello streaming. Mentre è interessante

notare come la qualità di CABA Fixed sia inferiore rispetto agli altri algoritmi, questo si può ricondurre al problema della scelta del giusto offset: di fatto, l'aver scelto una finestra troppo piccola fa si che il processo di prefetching non funzioni correttamente.

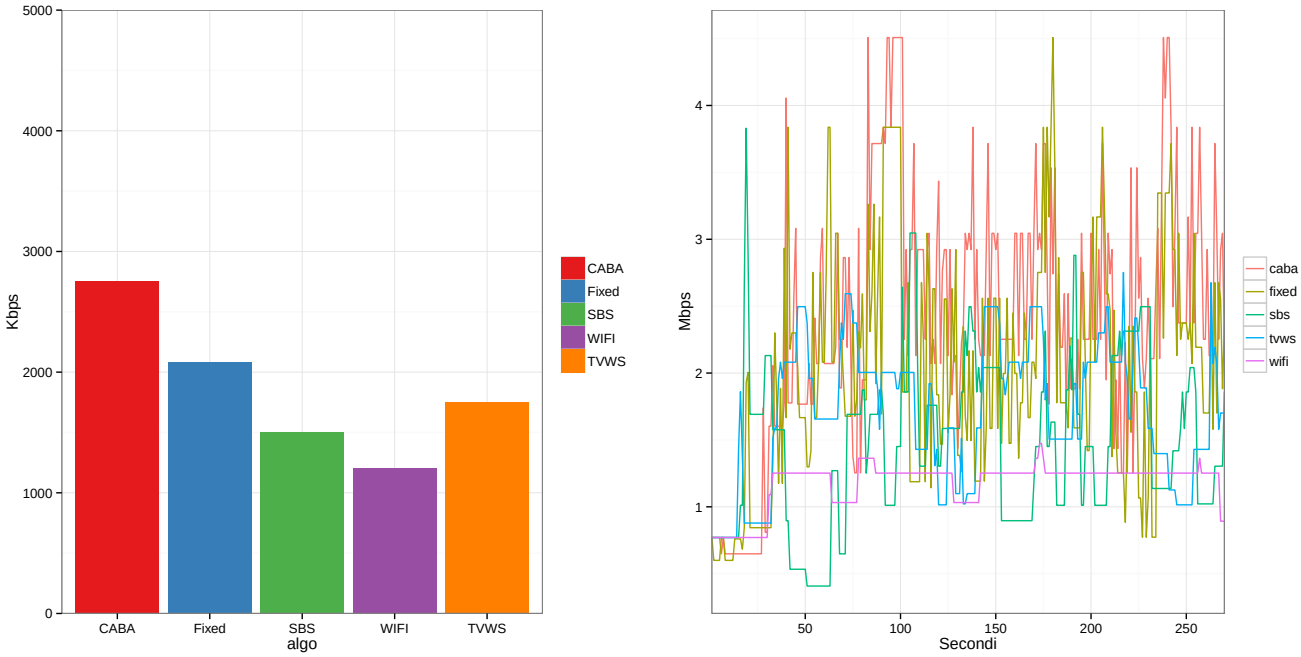
Figura 5.2: Qualità media secondo scenario



5.2.3 Terzo Scenario

Il terzo scenario vede le due connessioni con velocità medie singole molto simili ed entrambe molto basse. Questo scenario mette in risalto il miglioramento di qualità apportato da CABA, in quanto essendo entrambe le interfaccie non molto veloci CABA riesce a combinarle permettendo un aumento di circa il 25% della qualità. Anche in questo caso come è normale che sia StepByStep presenta una qualità media leggermente inferiore. Possiamo inoltre osservare che a differenza dello scenario precedente la qualità media di CABA Fixed sembra essere leggermente migliorata: questo perchè è stata scelta una finestra migliore in questo scenario.

Figura 5.3: Qualità media terzo scenario



5.3 Efficienza

Con il termine efficienza in questa sezione indicheremo la percentuale di segmenti scaricati mediante l'uso di prefetching diviso i segmenti scaricati in prefetching e realmente inviati al client. Ci si potrebbe trovare, infatti, nella situazione in cui il client richieda un segmento che è stato scelto per il prefetching e che il processo che lo stia scaricando non abbia ancora finito, in questo caso si procederebbe con l'annullamento del download in corso. Questo può avvenire per diversi motivi come ad esempio un problema nell'interfaccia scelta oppure un errore di calcolo negli algoritmi di prefetching.

In questa sezione andremmo ad analizzare nei vari scenari quale sia l'efficienza dei diversi algoritmi e la mettermo in confronto con la qualità pecedentemente analizzata. Inoltre sarà particolarmente interessante vedere come varia l'efficienza nell'algoritmo CABA Fixed con l'aumentare della finestra scelta. Infatti per ogni scenario esisterà sempre una finestra sufficientemente ampia che permetterà all'algoritmo di raggiungere un efficienza massima. L'ampiezza di questa finestra è strettamente legata con le caratteristiche

della connessione.

5.3.1 Algoritmi a confronto

Come possiamo vedere dai grafici l'efficienza degli algoritmi CABA sembra rimanere più o meno costante intorno al 95% mentre CABA Fixed sembra avere una efficienza maggiore di CABA. Questo avviene in quanto per i test si è usato una finestra sufficientemente grande e ciò ha permesso all'algoritmo di raggiungere l'efficienza ottimale. I risultati di CABA non sono tuttavia da considerare peggiori rispetto agli altri perchè nell'effettuare le simulazioni è stata utilizzata una versione dell'algoritmo CABA ottimistica che cerca di calcolare il tempo stimato nel download senza grande margine di errore. Bastarebbe aumentare quest'ultimo per migliorare l'efficienza di CABA a discapito però della frequenza di utilizzo del prefetching.

Un discorso diverso va fatto per l'algoritmo StepByStep, che infatti presenta un problema di progettazione nel momento delle simulazioni. Come si può vedere dai grafici infatti l'efficienza di quest'ultimo è minore degli altri due algoritmi ed inoltre si abbassa con il diminuire della velocità dell'interfaccia usata per il prefetching. Il problema è dovuto ad una mancaza di margine di errore nell'algoritmo. Questo fa si che anche una piccola variazione nello stato della rete renda il tempo stimato per il download errato.

Figura 5.4: efficienza primo scenario

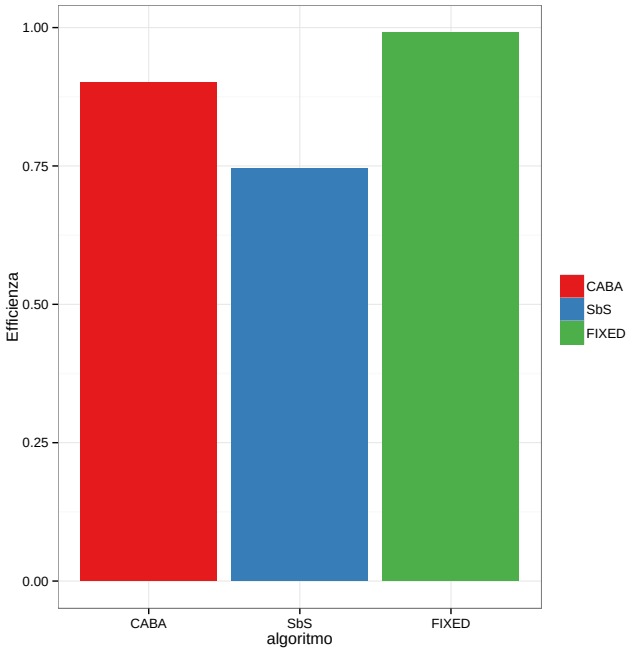


Figura 5.5: efficienza secondo scenario

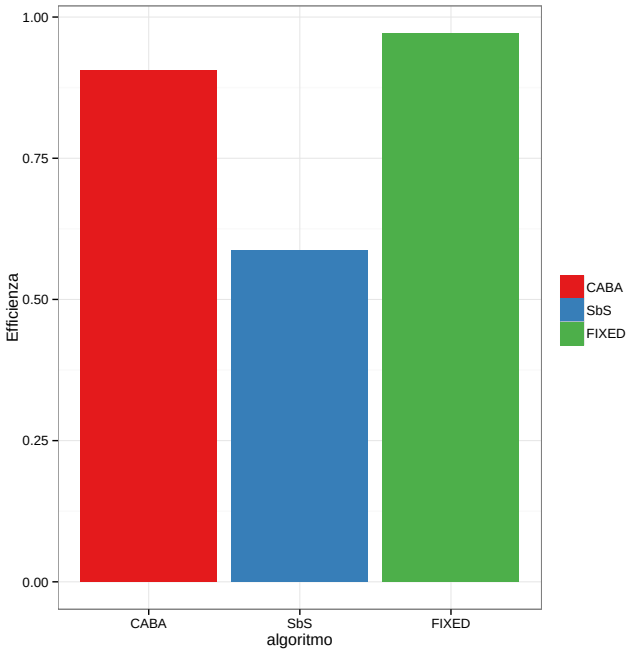
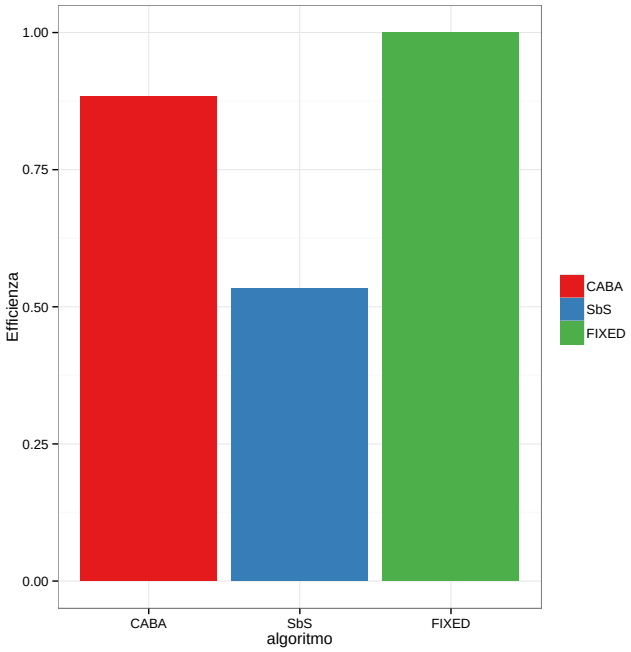


Figura 5.6: efficienza terzo scenario



5.3.2 CABA FIXED

I seguenti grafici mettono in rilievo l'aumentare dell'efficienza con l'aumentare della finestra scelta.

Nel primo scenario come possiamo vedere dai grafici CABA Fixed raggiunge l'efficienza massima con un offset di sei in quanto la seconda connessione è abbastanza veloce da riuscire a finire quasi sempre prima che avvenga la richiesta del segmento da parte del client.

Nel secondo scenario si può notare come al diminuire della velocità della connessione usata per il prefetching aumenti anche la dimensione dell'offset per raggiungere la massima efficienza. Come si può vedere, mentre nel primo scenario l'efficienza dell'algoritmo con un offset di due è intorno al 90%, qui crolla drasticamente abbassandosi al 70% circa. Bisognerà quind aumentare la grandezza dell'offset fino ad 8 per avere l'efficienza massima in questo scenario.

Nel terzo scenario invece visto che la velocità delle due connessioni risulta essere molto simile, l'ampiezza dell'offset ottimale è due. Questo dipende dal fatto che essendo

molto simili le velocità di entrambe le connessioni il client sceglie una qualità che vada bene per entrambe.

Figura 5.7: Efficienza primo scenario

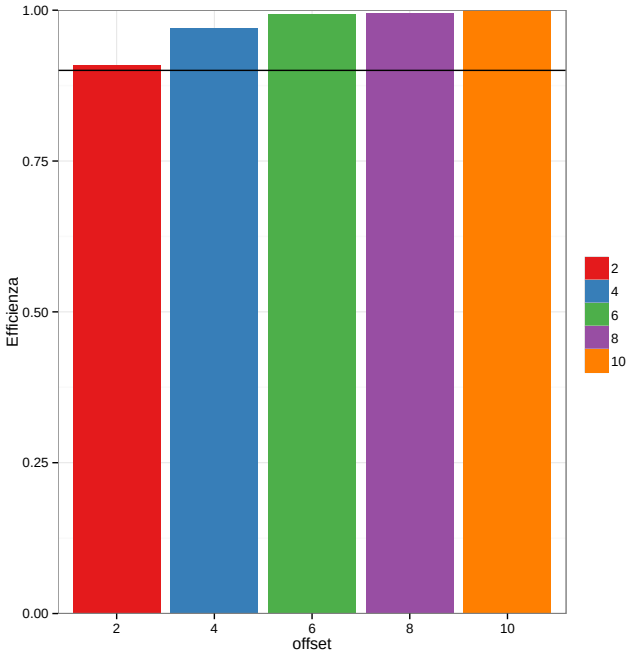


Figura 5.8: efficienza CABA Fixed secondo scenario

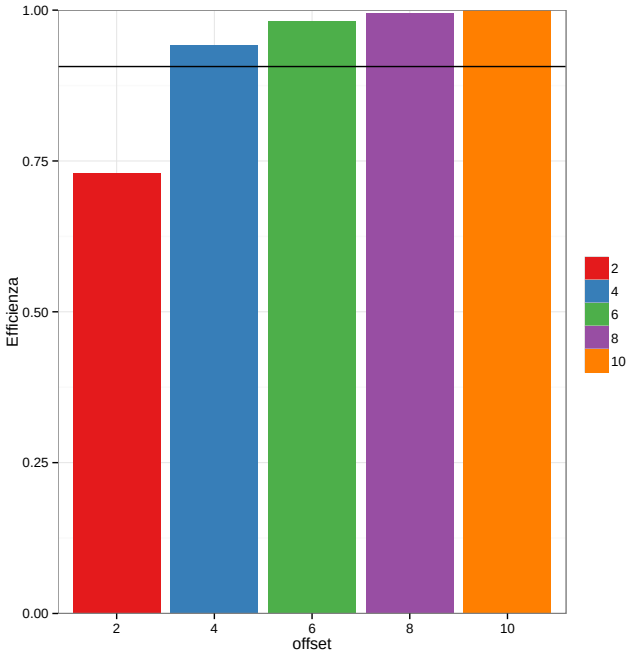
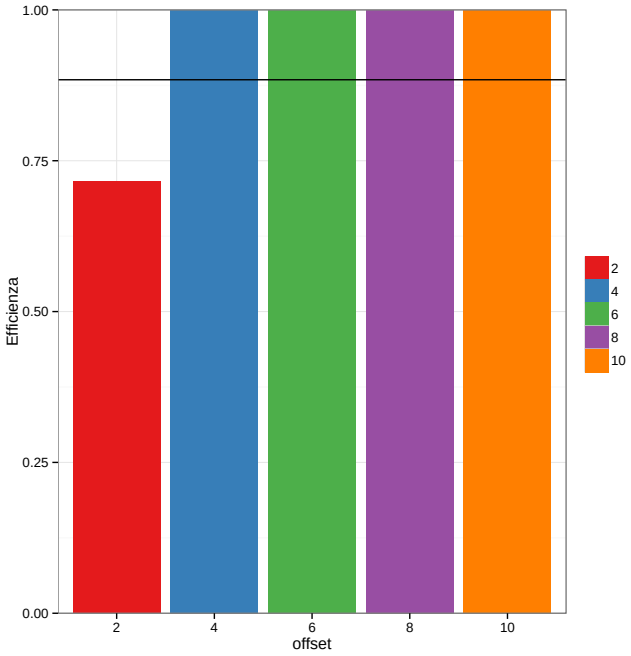


Figura 5.9: efficienza CABA Fixed terzo scenario



5.4 Lunghezza dei segmenti

Un altro dato interessante da studiare riguarda la lunghezza dei segmenti in cui viene diviso il video. Il calcolo della lunghezza ottimale dei segmenti non è banale ed è strettamente correlato con le caratteristiche della rete che si usa. Ad esempio, in reti molto stabili, segmenti più lunghi permettono di sfruttare la qualità della rete al meglio; la maggiore potenza della connessione permette di scaricare dati di dimensioni più grandi con una sola connessione HTTP in modo più efficiente. Al contrario, per reti meno stabili che in quanto tali sono soggette spesso cambiamenti dei segmenti più corti potrebbero essere più adatti . Tuttavia i piu grandi fornitori di servizi di streaming online come Netflix e Apple consigliano l'utilizzo di segmenti dalla durata media di 10 secondi.

Nei seguenti grafici si mostra come varia la qualità media nel tempo in tutti e tre gli scenari usando lunghezze dei segmenti differenti. Il video usato per la valutazione è disponibile in sei diverse lunghezze dei segmenti. In particolare sono previsti:

- segmenti da 1 secondo;
- segmenti da 2 secondi;
- segmenti da 4 secondi;
- segmenti da 6 secondi;
- segmenti da 10 secondi
- segmenti da 15 secondi;

Possiamo notare come nei primi due scenari la qualità tenda ad essere stabile in tutte le diverse lunghezze di segmenti, in contrasto con il terzo scenario dove la qualità media tende ad essere costante solo con segmenti di lunghezza maggiore ai dieci secondi.

Figura 5.10: Segmenti di diversa lunghezza, CABA primo scenario

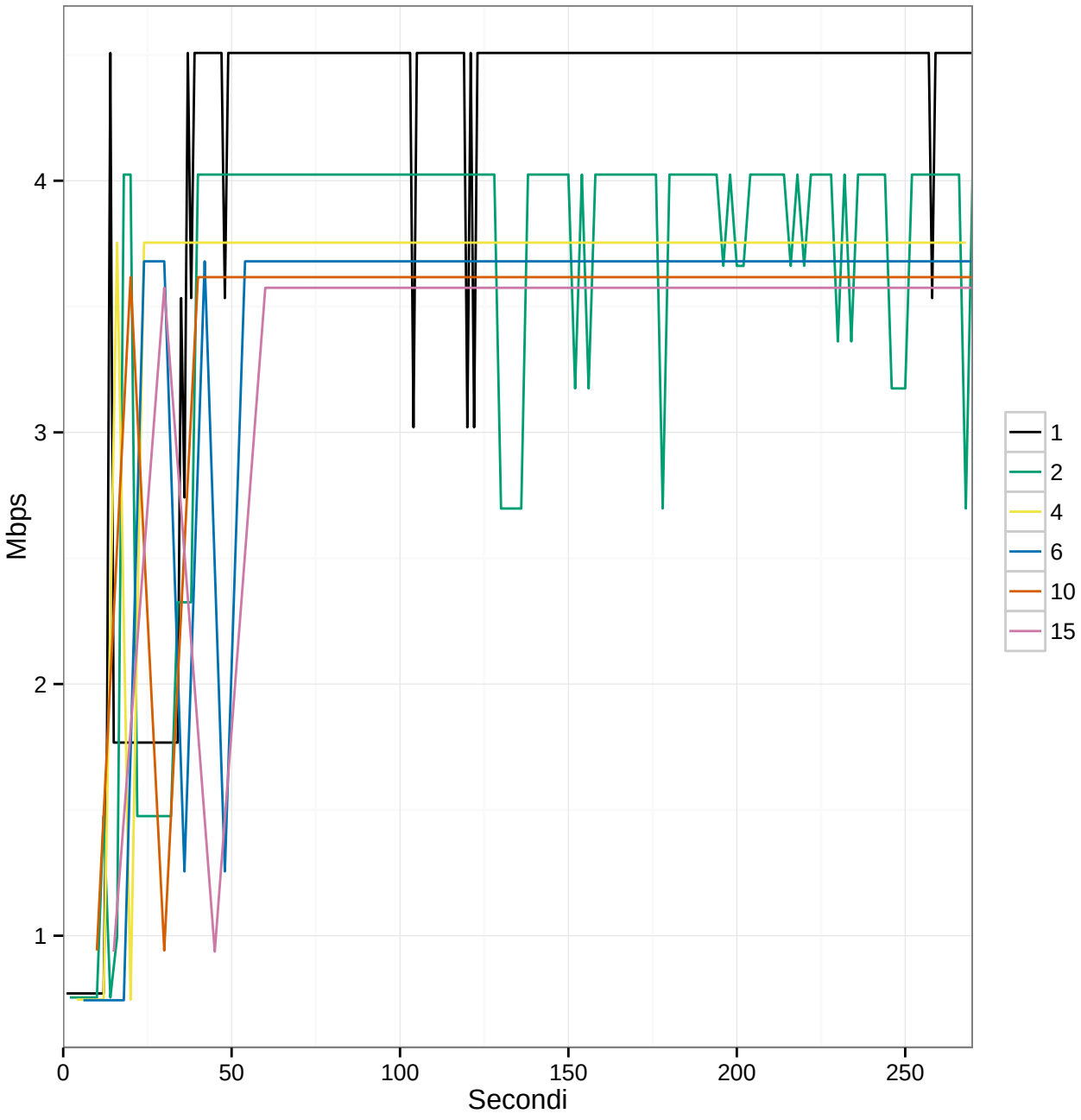


Figura 5.11: Segmenti di diversa lunghezza con algoritmo CABA nel secondo scenario

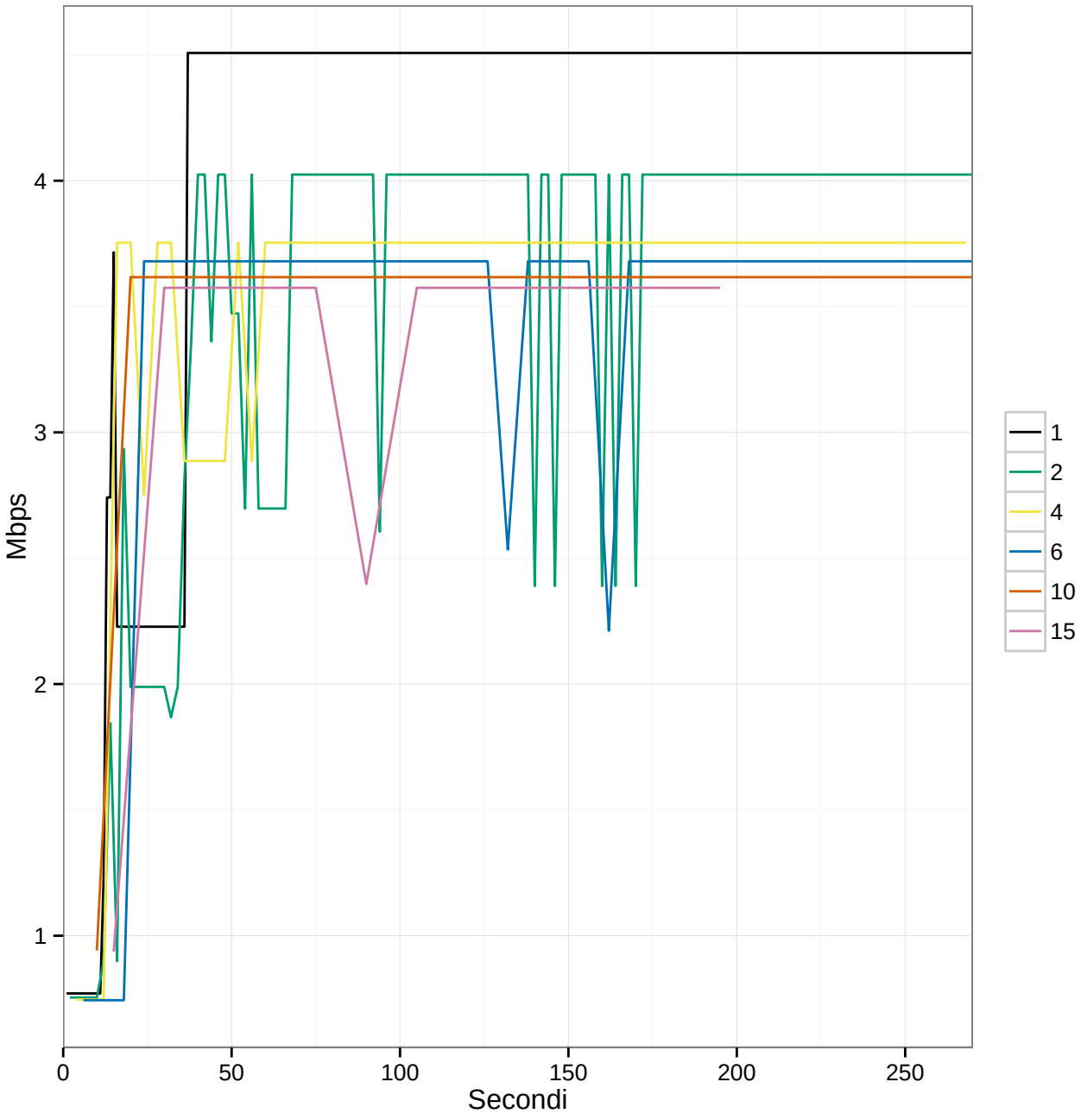
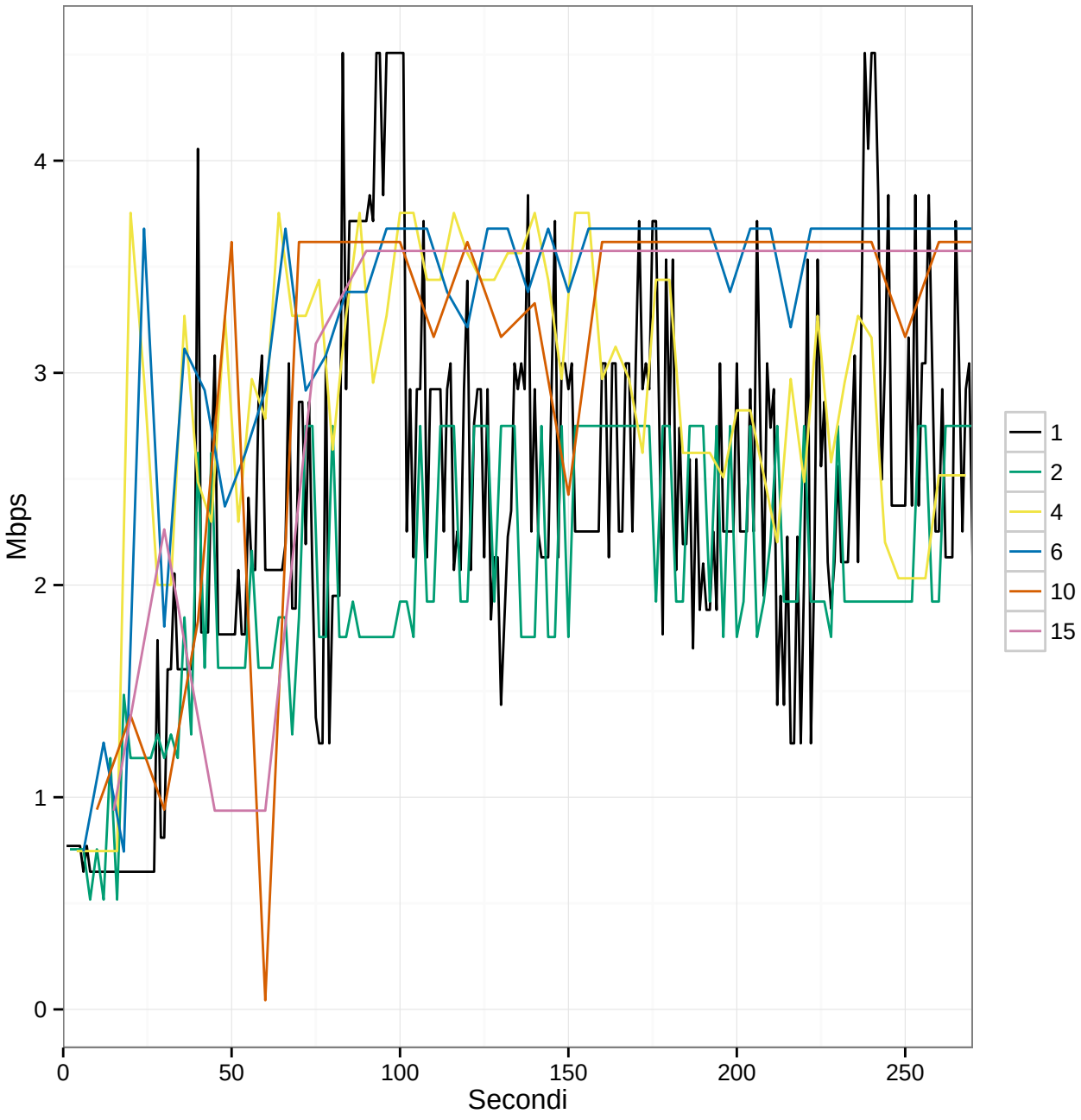


Figura 5.12: Segmenti di diversa lunghezza, CABA terzo scenario



Capitolo 6

Conclusioni

In uno scenario globale dove il traffico video mobile di Internet tende ad aumentare sempre di più si trova la necessità cercare di migliore la qualità nella gestione di questo traffico. Questo progetto si propone di creare un mezzo efficiente per migliore lo streaming video sopratutto laddove la rete non gode di ottime prestazioni. L'utilizzo di più interfacce contemporaneamente in un futuro dove si prevedono sempre nuovi metodi addizionali per accedere ad Internet, potrebbe essere un'ottima soluzione per tutte le tipologie di dispositivi. è quindi stato implementato un implementato un sistema middleware che, oltre a permettere lo streaming auto-adattante come altre tecnologie già esistenti (quale MPEG-DASH), presenta una nuova funzionalità aggiuntiva grazie alla quale vengono sfruttate tutte le interfacce di rete presenti contemporaneamente. Questo permette di adattarsi al contesto in cui si trova l'utente in termini di connettività e fornire ad esso una qualità di streaming migliore. In seguito all'implementazione sono stati effetuati diverse simulazioni per valutare il funzionamento del progetto e le prestazioni dei diversi algoritmi implementati in condizioni diverse. Da questi è risultato che la qualità viene effettivamente migliorata in tutti gli scenari fino a ranggiungere un 33%

Un'ulteriore espansione del progetto potrebbe prevedere un sistestama che permettà all'utente di poter decide un livello di sicurezza, cioè valore che gli algoritmi useranno nell'effettuare la scelta. In sostanza, se il livello di sicurezza sarà alto gli algortimi sceglieranno il segmento che quasi sicuramente riusciranno a scaricare, mentre con un livello di sicurezza minore proverrano a fare un scelta più vicina alla realtà del segmento

da scaricare; anche se la possibilità che il downalod non finisca in tempo si alza.

Inoltre, si potrebbe pensare di aggiungere un ulteriore criterio per la scelta dell'interfaccia rete: quello del costo. Questo permetterebbe all'utente di sfruttare al meglio connessioni più veloci ma allo stesso tempo più costose. Ad esempio, anche se la connessione LTE permette una qualità migliore dell'ADSL, essendo la prima a pagamento, l'utente potrebbe preferire di usarla solo fino ad una certa soglia preimpostata (e.g. dieci euro).

Ringraziamenti

Ringrazio il Professor Bononi e il Dottor Bedogni per il loro prezioso sostegno. Ringrazio anche la mia famiglia per avermi dato l'opportunità di sviluppare la mia passione. Ringrazio Margherita e tutti i miei coinquilini, acquisiti o meno, per la pazienza e l'aiuto datomi in questi anni universitari.

Bibliografia

[1] Luca Bedogni, Marco Di Felice, Fabio Malabocchia, and Luciano Bononi. Cognitive Modulation and Coding scheme Adaptation for 802.11n and 802.11af networks. In *Globecom 2014 Workshop - Telecommunications Standards - From Research to Standards (GC14 WS - TCS)*, 2014.

[2] Luca Bedogni, Marco Di Felice, Fabio Malabocchia, and Luciano Bononi. Indoor communication over TV gray spaces based on spectrum measurements. In *2014 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 3218–3223. IEEE, apr 2014.

[3] Luca Bedogni, Angelo Trotta, and Marco Di Felice. On 3-Dimensional Spectrum Sharing for TV White and Gray Space Networks. In *Proceedings of WoWMoM*, 2015.

[4] L Bononi, M Di Felice, A Molinaro, and S Pizzi. A Cross-Layer Architecture for Robust Video Streaming over Multi-Radio Multi-Channel Wireless Mesh Networks. *Mobiwac09: Proceedings of the Seventh Acm International Symposium on Mobility Management and Wireless Access*, pages 75–82, 2009.

[5] Cambridge White Spaces Consortium. Cambridge TV White Spaces Trial A Summary of the Technical Findings. Technical report.

[6] Jim Carlson. Cambridge TV White Space Trials : The Report Cambridge TVWS Trial Policymaker Event. Technical report, 2012.

[7] Cisco. Cisco Visual Networking Index: Forecast and Methodology, 2014, 2019. 2015.

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

[9] Rob Davies and Monisha Ghosh. Field trials of DVB-T sensing for TV White Spaces. In *2011 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, pages 285–296. Philips Research, IEEE, may 2011.

[10] Andrea Detti, Matteo Pomposini, Nicola Blefari-Melazzi, Stefano Salsano, and Andrea Bragagnini. Offloading cellular networks with information-centric networking: The case of video streaming. In *2012 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks, WoWMoM 2012 - Digital Proceedings*, 2012.

[11] R Di Bari, M Bard, J Cosmas, R Nilavalan, K K Loo, H Shirazi, and K Krishnapillai. Field trials and test results of portable DVB-T systems with transmit delay diversity. *2008 IEEE International Symposium on Consumer Electronics*, 2008.

[12] Savio Dimatteo, Pan Hui, Bo Han, and Victor O K Li. Cellular traffic offloading through WiFi networks. In *Proceedings - 8th IEEE International Conference on Mobile Ad-hoc and Sensor Systems, MASS 2011*, pages 192–201, 2011.

[13] Lei Ding, Scott Pudlewski, Tommaso Melodia, Stella Batalama, John D Matyjas, and Michael J Medley. Distributed Spectrum Sharing for Video Streaming in Cognitive Radio Ad Hoc Networks. *Spectrum*, 28:1–13, 2010.

[14] Florin Dobrian, Asad Awan, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and U C Berkeley. Understanding the Impact of Video Quality on User Engagement. *World*, pages 362–373, 2011.

[15] Florin Dobrian, Asad Awan, Dilip Joseph, Aditya Ganjam, Jibin Zhan, Vyas Sel, Ion Stoica, and Hui Zhang. Understanding the Impact of Video Quality on User Engagement. In *Communications of the ACM*, pages 91–99, 2013.

[16] Ericsson. Ericsson Mobility Report: 70 percent of world's population using smartphones by 2020. 2015.

[17] Adriana B Flores, Ryan E Guerra, and Edward W Knightly. IEEE 802.11af: a standard for TV white space spectrum sharing. *IEEE Communications Magazine*, 51(10):92–100, oct 2013.

[18] Gheorghita Ghinea and Johnson P. Thomas. Quality of perception: User quality of service in multimedia presentations. *IEEE Transactions on Multimedia*, 7:786–789, 2005.

[19] Lorenzo Keller, Christina Fragouli, and U C Irvine. MicroCast : Cooperative Video Streaming on Smartphones Categories and Subject Descriptors. *MobiSys 2012*, pages 57–69, 2012.

[20] Fabio Malabocchia, Romeo Corgiolu, Maurizio Martina, Andrea Detti, Bruno Ricci, and Nicola Blefari-Melazzi. Using Information Centric Networking for Mobile Devices Cooperation at the Network Edge. In *2015 IEEE 81st Vehicular Technology Conference (VTC Spring)*, pages 1–6. IEEE, may 2015.

[21] Christopher Müller, Stefan Lederer, and Christian Timmerer. An evaluation of dynamic adaptive streaming over HTTP in vehicular environments. *Proceedings of the 4th Workshop on Mobile Video - MoVid '12*, page 37, 2012.

[22] Iraj Sodagar. The MPEG-DASH Standard for Multimedia Streaming Over the Internet. *IEEE Multimedia*, 18(4):62–67, apr 2011.

[23] Jaap van de Beek, Janne Riihijärvi, Andreas Achtzehn, and Petri Mähönen. TV White Space in Europe. *IEEE Transactions on Mobile Computing*, 11(2):178–188, feb 2012.

Elenco delle figure

2.1 Struttura Manifest[22] 9
2.2 Manifest MPEG-DASH 10
2.3 MPEG-DASH struttura[22] 11
2.4 Confronto tra TWVS e Wi-Fi[2] 13

4.1 Infrastruttura 16
4.2 Diagramma delle classi 22

5.1 Qualità media primo scenario 30
5.2 Qualità media secondo scenario 31
5.3 Qualità media terzo scenario 32
5.4 efficienza primo scenario 34
5.5 efficienza secondo scenario 34
5.6 efficienza terzo scenario 35
5.7 Efficienza primo scenario 36
5.8 efficienza CABA Fixed secondo scenario 37
5.9 efficienza CABA Fixed terzo scenario 37
5.10 Segmenti di diversa lunghezza, CABA primo scenario 39
5.11 Segmenti di diversa lunghezza con algoritmo CABA nel secondo scenario 40
5.12 Segmenti di diversa lunghezza, CABA terzo scenario 41

List of Algorithms

4.1 Random 23
4.2 Fixed 24
4.3 CABA 25
4.4 StepByStep 27