Università degli studi di Pisa

Facoltà di Scienze Matematiche Fisiche e Naturali

Tesi di Laurea Specialistica in

Tecnologie Informatiche

Disegno e implementazione di un framework per la misura qualitativa dei servizi di rete

Relatore

Prof. Luca Deri

Candidato

Daniele Sartiano

Controrelatore

Prof. Antonio Brogi

Anno Accademico 2009/2010

Università degli studi di Pisa

Facoltà di Scienze Matematiche Fisiche e Naturali

Tesi di Laurea Specialistica in

Tecnologie Informatiche

Disegno e implementazione di un framework per la misura qualitativa dei servizi di rete

Relatore

Prof. Luca Deri

Candidato

Daniele Sartiano

Controrelatore

Prof. Antonio Brogi

Anno Accademico 2009/2010

Abstract

Data la molteplicità e l'eterogeneità dei servizi di rete si ha la necessità di nuovi strumenti di monitoraggio. Quelli attualmente disponibili, offrono risultati orientati alle metriche di rete, i quali sono difficilmente interpretabili da parte degli utenti, che invece ragionano in termini di servizi. Questo lavoro di tesi propone dunque una metodologia per l'analisi di alcuni di essi basati su HTTP, per i quali sono state definite alcune metriche applicative interpretabili dagli utenti. In particolare viene proposta una metodologia per:

- Classificare il traffico HTTP in base ai servizi di rete utilizzati (ad esempio posta elettronica, *streaming* video, aggiornamento di software).
- Correlazione e aggregazione delle singole richieste, per ricavare l'albero di navigazione, da cui poter determinare le attività svolte dell'utente.
- Analizzare la qualità dell'esperienza utente relativa al servizio web e a quello di *streaming* offerto da *YouTube*.

Al fine di validare questo lavoro è stato sviluppato un framework, su cui si basano alcune applicazioni per l'analisi del traffico di rete. La sperimentazione, effettuata analizzando pacchetti catturati da un ISP regionale, ha confermato la possibilità di ottenere informazioni ad alto livello, sia sulle attività svolte dagli utenti, che sulla qualità percepita dei servizi di rete studiati. I risultati ottenuti rendono attuabile l'estensione del framework

per lo studio di altri servizi Internet, come ad esempio la posta elettronica, messaggistica istantanea o multiplayer, in cui vi è la necessità di analizzare l'esperienza utente relativa.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

La molteplicità dei servizi offerti su Internet cresce giorno dopo giorno e con essi nasce l'esigenza di strumenti in grado di monitorarli. I sistemi odierni per il monitoraggio di rete offrono dei risultati vicini alla rete, i quali descrivono cosa accade sulla rete a basso livello, di conseguenza lo studio dei servizi di rete ad alto livello diventa un'operazione complessa, in quanto non vengono esposte in modo chiaro e immediato le informazioni relative ad un determinato servizio.

Da una parte gli Internet Service Provider si stanno spostando verso un approccio orientato al servizio, nella quali vengono utilizzate tecniche *top-down*, in cui vengono monitorati i servizi esposti e in caso di un'eventuale problematica, viene esaminata la rete a basso livello per trovare l'esatta causa che ha scaturito il problema.

D'altra parte l'utente ragiona in termini di applicazione/servizio e non di rete, quindi ha la necessità di un'analisi ad alto livello, per poter comprendere cosa accade durante l'utilizzo di un determinato servizio di rete.

Lo scopo di questo lavoro di tesi è quello di proporre una metodologia per il monitoraggio dei servizi di rete, in grado di tracciare, correlare e caratterizzare il traffico di rete, rendendo disponibile un'interpretazione di esso ad alto livello. Inoltre si vuole analizzare la qualità dei servizi di rete,

offrendo delle metriche in grado di determinare la qualità dell'esperienza utente relativa ai servizi monitorati.

L'utilizzatore di questo sistema sarà in grado di avere una panoramica sul traffico di rete, suddiviso per servizi, potendo quindi evidenziare eventuali problematiche relative ad esso.

1.1 Struttura della tesi

In questo paragrafo verrà illustrata brevemente la struttura della tesi, descrivendo i contenuti per ogni capitolo.

Capitolo 1 Nel seguente capitolo verranno spiegate le motivazioni per cui è stato svolto questo lavoro di tesi. Vi sarà un'introduzione al monitoraggio di rete e dei servizi. Saranno descritte la qualità del servizio e la qualità dell'esperienza utente; i servizi analizzati e le relative caratteristiche prese in considerazione, ed infine gli obiettivi di questa tesi.

Capitolo 2 In questo capitolo verrà prima descritto lo stato dell'arte relativo al monitoraggio di rete. Nella seconda parte verrà trattato lo stato dell'arte correlato a questo lavoro di tesi per quanto riguarda il monitoraggio dei servizi di rete.

Capitolo 3 Qui verrà descritta la metodologia proposta, sarà illustrato come vengono estratte le metriche relative alla qualità dei servizi di rete studiati.

Capitolo 4 Capitolo in cui verrà illustrata l'architettura del framework proposto. Verranno analizzate le caratteristiche di ogni elemento presente e le motivazioni delle scelte progettuali.

Capitolo 5 Verrà validato il framework realizzato, mostrando i risultati ottenuti attraverso l'elaborazione di dati reali.

Capitolo 6 In questo capitolo verranno tratte le conclusioni, evidenziando gli obiettivi raggiunti e i possibili sviluppi futuri.

1.2 Motivazioni della tesi

Attualmente la maggior parte degli strumenti disponibili per il monitoraggio di rete, prevedono la cattura di pacchetti e l'analisi di essi a basso livello; di conseguenza, lo studio dei servizi di rete risulta difficoltoso, soprattutto per chi non è un operatore del settore del networking. Si provi, ad esempio, a pensare ad una macchina collegata ad Internet, la quale venga utilizzata per navigare su Internet, scambiare messaggi tramite servizi di *instant messaging*, videochiamare, guardare video in *streaming*. Dal punto di vista della rete, una considerevole quantità di pacchetti si sposta dalla macchina in questione verso Internet e viceversa, con numerosi protocolli differenti; ne consegue che l'analisi di questi pacchetti, per lo studio della qualità di un determinato servizio, richiede un lavoro notevole ed esperienza nel campo del monitoraggio rete. Da queste considerazioni nasce l'idea del framework proposto in questa tesi.

Si vuole quindi fornire una metodologia in grado di prelevare il traffico di rete a basso livello, di caratterizzarlo, di astrarlo e di correlarlo, in modo da ottenere delle informazioni ad alto livello, comprensibili in maniera chiara e immediata. I dati ricavati potranno essere utili a chi vuole effettuare delle analisi di rete, sia per comprendere quale sia effettivamente il traffico generato da una macchina collegata ad Internet, sia per evidenziare eventuali degradi della qualità di un determinato servizio, così da poter operare per risolvere eventuali problematiche.

1.3 Monitoraggio di rete

I sistemi di monitoraggio di rete attuali, offrono un risultato in termini di rete, ovvero si ha un'analisi in cui vengono esposte le comunicazioni di un determinato protocollo, mostrando delle metriche relative ai pacchetti catturati durante il monitoraggio. Riuscire a capire quello che realmente è accaduto durante una sessione di monitoraggio potrebbe essere arduo, in quanto non vi è un'interpretazione dei dati ottenuti. Per poter fornire all'utente informazioni ad alto livello, ci si sta spostando verso un approccio di monitoraggio orientato ai servizi.

1.4 Monitoraggio dei servizi

Lo studio e l'analisi della qualità dei servizi Internet giocano un ruolo fondamentale per poter assicurare una buona esperienza utente, infatti i fornitori di servizi Internet (*Internet Service Provider*) si stanno spostando da un approccio orientato alla rete ad uno orientato al servizio, in modo tale da monitorare qualità e disponibilità dei servizi e, in caso di un'eventuale problematica, esaminare il problema alla radice a livello di rete. A differenza di prodotti attualmente presenti in commercio [1, 2], si vuole proporre una metodologia la quale possa essere estesa per diversi servizi, e quindi progettare un framework che validi quest'ultima. I campi di interesse per quanto riguarda l'analisi dei servizi di rete sono molteplici, come ad esempio il monitoraggio di applicazioni web o il servizio di posta elettronica. Nel capitolo sucessivo vi sarà una descrizione sui possibili campi di applicazione.

1.5 Qualità del Servizio e Qualità dell'Esperienza

Per misurare la qualità di un servizio internet, è necessario distinguere la qualità del servizio dalla qualità percepita dall'utente, ovvero la *Quality*

of Service (*QoS*) dalla *Quality of Experience*(*QoE*). Con la *QoS* si osserva la qualità del servizio dal punto di vista della rete, con metriche come il numerico di pacchetti persi, il *throughput*, la latenza e *jitter*, consegna dei pacchetti *Out-of-order*.

La *QoE*, viene definita in [3] come un concetto che comprende tutti gli elementi della percezione di un utente della rete e le *performance* relative alle loro aspettative. In [4] la *QoE* viene definita come la percezione soggettiva dell'utente finale riguardo la qualità complessiva di un'applicazione o servizio, specificando che la *QoE* include tutto il sistema *End-to-End* e che la qualità complessiva può essere influenzata dal contesto e dalle aspettative dell'utente. In conclusione, la *QoE* può essere intesa come la misura soggettiva dell'esperienza utente [5].

In [6] vengono introdotte le differenze tra *QoS* e *QoE* per quanto riguarda il servizio Web, nell'articolo viene evidenziato che le metriche relative alla *QoS* sono metriche quali la disponibilità e le *performance*, invece le metriche della *QoE* sono ad esempio il tempo di risposta, la percezione della velocità di un *download*. Viene inoltre fatto notare che mentre i parametri della *QoS* sono sotto il controllo del fornitore del servizio, quelli della *QoE* , anche se strettamente in relazione a quelli della *QoS*, possono esere influenzati da elementi soggettivi dell'utente e da un qualsiasi sistema tra il fornitore del servizio e l'utente finale. La *QoE* può essere applicata a qualsiasi tipo di servizio di rete, come ad esempio il web, lo *streaming* o il VoIP.

La metodologia proposta in questa tesi, ha come obiettivo quello di misurare la qualità di servizi di rete, in termini di *QoS* e in termini di *QoE*; mostrando all'utente cosa realmente accade durante l'utilizzo di determinati servizi di rete. Si precisa che la qualità dell'esperienza utente viene misurata solo in forma oggettiva e non soggettiva, in quanto per una valutazione soggettiva di essa, si avrebbe la necessità di *feedback* forniti dall'utente, i quali evidenzierebbero come l'utente ha percepito soggettivamente la qualità

di un determinato servizio.

1.6 Servizi di rete analizzati

I servizi disponibili su Internet aumentano giorno dopo giorno, ognuno con caratteristiche differenti, risulta quindi irrealistico poterli coprire tutti in un unico lavoro di tesi, per questo motivo si è scelto di approfondire solo alcuni di essi, adottando una metodologia la quale potrà essere estesa per poter integrare qualsiasi altro servizio. Nella tabella 4.2 una lista di alcuni dei servizi più utilizzati su Internet [7]:

Rank	Servizio
1	Web
2	Video
3	VPN
4	Email
5	News
6	P2P
7	Games

Tabella 1.1: Servizi Internet più utilizzati

Attualmente lo scambio di contenuti multimediali, quali foto, video e musica, il *download* di *software* e aggiornamenti, contribuiscono a comporre la maggior parte del traffico Internet [7, 8, 9, 10]. Per accedere a questi contenuti, nella maggior parte dei casi, viene utilizzato il protocollo HTTP, infatti esso rappresenta più del 50 % del traffico Internet [7, 11, 8, 9, 10].

In questo tesi si è voluto analizzare il traffico Web e lo *Streaming on demand*, il quale è analogo al *download* di *files*, nei quali il protocollo per il trasporto è HTTP.

Il VoIP non verrà trattato, in quanto i protocolli per il VoIP, come RTP (*Real-time Transport Protocol*), RTCP (*Real-time Transport Control Protocol*), SIP (*Session Initiation Protocol*) e H.323, sono nati con dei meccanismi per la misura della qualità del servizio, i quali permettono di adattarsi dinamicamente alle condizioni di rete. Medesima motivazione va per i servizi di *Streaming live*, i quali utilizzano protocolli, come RTSP (*Real-time Streaming Protocol*), RTP e RTCP, i quali sono stati progettati per poter monitorare e garantire la qualità del servizio.

Un altro servizio, che in questo lavoro di tesi, per mancanza di tempo, non verrà trattato è il servizio di posta elettronica; la ricostruzione delle attività svolte da un utente, per quanto riguarda il servio *email*, potrebbe risultare non banale. In [12] vengono descritte quali informazioni intercettare, in relazione al processo di ricezione e invio di posta elettronica, in ambito di *cyber security*. In [13] viene proposto un metodo per poter individuare messaggi di posta elettronica contenenti virus, i quali si autopropagano, incrementando significativamente il volume del traffico relativo al servizio email; attraverso il monitoraggio del servizio di rete è possibile così individuare un virus con sufficiente anticipo, in modo tale da infettare solo una piccola parte dei client appartenti ad una rete Intranet. In uno sviluppo futuro potrebbe essere interessante studiare anche questo servizio.

Di seguito verranno trattati alcuni lavori collegati a quato lavoro di tesi riguardanti i servizi di rete trattati.

1.6.1 Il servizio Web

La misura della qualità del servizio Web, risulta utile in molti campi, in quanto è il servizio di rete più utilizzato. In questo paragrafo verranno descritti alcuni studi, in cui attraverso il monitoraggio del traffico di rete è possibile dare una misura qualitativa del servizio Web.

In [14] viene discusso come la *QoE* riguardo al servizio Web è stretta-

mente correlata da due principali parametri della *QoS*, che sono la banda e la latenza. La loro è un'analisi sperimentale in cui viene illustrato che la *QoE* di un utente che naviga sul Web attraverso un *Internet Service Provider* è leggermente affetta dalla latenza di rete e altamente affetta dalla banda di rete. Da questo si deduce la stretta correlazione tra *Quality of Service* e *Quality of Experience*

In [15] viene introdotto uno studio relativo alla *QoE* di un sistema per l'apprendimento automatico, i risultati ottenuti evidenziano che la qualità percepita da un utente finale con un sistema di informazioni on-line è influenzata non solo dalla percezione dell'utente e dalle aspettative in relazione all'interazione con il sistema, ma anche da problemi relativi alla rete, come il tipo di connessione, il carico della rete e caratteristiche del dispositivo utilizzato dall'utente. In conclusione viene mostrato che utilizzando un approccio relativo al miglioramento della *QoE* sono stati ottenuti risultati favorevoli per quanto riguarda il tempo di sessione, il tempo di elaborazione delle informazioni per pagine e il numero di pagine rivisitate. Da questi risultati è chiaro che la qualità di un servizio è fondamentale, ed è per questo motivo che nasce la necessità di fornire una misura qualitativa dei servizi di rete.

Recenti ricerche [16] mostrano che è possibile identificare un utente, in base alle proprie attività su Internet attraverso dei *pattern* applicati ai flussi di rete catturati. Altri studi [17], analizzano dal punto di vista della rete, l'utilizzo dei *Social Network*, riuscendo a identificare le relative sessioni e le azioni degli utenti.

Le informazioni relative alle attività di un utente su Internet, potrebbero essere utilizzate, con l'uso di tecniche di *Data Mining*, per migliorare la qualità dei servizi Internet, come in [18], in cui vengono analizzati ed elaborati dati sull'utilizzo del web, per poter personalizzare in *real-time* la navigazione dell'utente.

Ricostruire le attività svolte da un utente su Internet, può essere utile

per molteplici scopi, soprattuto per quanto riguarda il migliorare l'esperienza utente. Si evidenzia quindi, la necessità di misurare la qualità del servizio Web, in modo tale da poter operare sui suoi punti deboli e quindi di apportare delle migliorie.

1.6.2 Il servizio di *Streaming*

Prima di descrivere i lavori correlati al servizio di *streaming*, è necessario dare una definizione di esso e suddividerlo in due tipologie: *Streaming on demand* e *Streaming Live*.

Il servizio di *streaming* consiste nell'invio continuo di un flusso di dati audio, video o entrambi, i quali vengono trasmessi da una sorgente verso una o più destinazioni in tempo reale. Si possono distinguere due tipologie di streaming:

Streaming on demand: in cui il *client* effettua una richiesta dei contenuti ad un *server*, quest'ultimo li invierà progressivamente, il *client* ricevuta una parte inziale sufficiente dei contenuti, inizierà ad eseguire i contenuti continuando il *download* del file.

Streaming live: dove il flusso audio/video viene inviato in *broadcast*.

Come accennato nel paragrafo 1.6 in questa tesi, non verrà trattato il servizio di *streaming live*, ma verrà trattato lo *streaming on demand*, in cui principalmente viene utilizzato il protocollo HTTP per il trasporto dei dati. Sarà quindi opportuno caratterizzare questa attività, la quale si differenzia dalla semplice navigazione web, per quanto riguarda la durata della comunicazione e di conseguenza della quantità di traffico generato. Per poter ricostruire le azioni svolte dall'utente mentre effettua attività di *streaming*, come ad esempio il cambio di qualità del filmato, sarà necessario identificare gli eventi di rete scaturiti da esse.

Vi sono alcuni studi, come in [19], dove viene misurato e studiato il traffico di rete verso il popolare *YouTube*. Dalle loro analisi risulta che attraverso l'uso di *cache* da parte del client, una distribuzione P2P (*peer-to-peer*), e la *cache* di un proxy, è possibile ridurre significativamente il traffico di rete e ridurre i tempi di accesso per la visione di un video. Simili risultati si hanno in [20], dove viene mostrato che l'uso di strategie di memorizzazione in *cache* può apportare miglioramenti riguardo l'esperienza dell'utente finale.

In questo lavoro di tesi, si vuole misurare la qualità dell'esperienza utente relativa al servizio di *streaming* di video attraverso il web, in modo tale da individuare eventuali insoddisfazioni utente, per poi poterne studiare le cause dal punto di vista della rete.

1.7 Obiettivi del lavoro di tesi

Questa tesi ha come scopo quello di studiare qualitativamente i servizi di rete, da una parte per analizzare l'esperienza utente (*User Experience*) relativa ad un determinato servizio, dall'altra per poter mostrare come evolve il traffico nel tempo, estraendo delle metriche in grado di dare una panoramica ad alto livello sui servizi utilizzati e quindi ricostruire le attività svolte dagli utenti nel tempo.

Per soddisfare questo obiettivo, si vuole proporre una metodologia, la quale verrà validata attraverso la progettazione di un sistema, che preso in ingresso il traffico di rete, possa elaborarlo, in modo tale da produrre un'analisi qualitativa dei servizi utilizzati dagli utenti.

Capitolo 2

Stato dell'Arte

In questo capitolo si analizzeranno per ogni servizio trattato, gli studi e le ricerche correlati a questa tesi. Il capitolo inizia con una breve introduzione al monitoraggio di rete, illustrando le più rilevanti metodologie presenti attualmente, per poi passare al monitoraggio dei servizi, approfondendo lo stato dell'arte relativo.

2.1 Monitoraggio del traffico rete

Il monitoraggio traffico di rete è possibile dividerlo in:

- Monitoraggio attivo: il quale si basa sulla capacità di poter iniettare pacchetti di test nella rete o inviare pacchetti a server e applicazioni, in modo tale da ottenere una misura del servizio analizzato. Questo approccio ha lo svantaggio di aumentare il traffico di rete e di creare del traffico artificiale. Esso viene utilizzato soprattutto per misurare la disponibilità e il tempo di risposta di un servizio.
- Monitoraggio passivo: questa tecnica si basa sulla cattura di pacchetti che transitano sulla rete, in modo tale da poter analizzarli successivamente. Attraverso il monitoraggio passivo non si ha un aumento del traffico di rete e viene misurato il traffico reale.

2.1.1 Tecniche di monitoraggio passivo

In questo paragrafo verranno descritti le metodologie più rilevanti nel campo del monitoraggio di rete, che utilizzano una tecnica passiva.

NetFlow

Uno dei protocolli più comuni, per quanto riguarda l'analisi del traffico attraverso flussi di rete è NetFlow, un protocollo sviluppato da *Cisco System Inc.* [21] per poter monitorare il traffico di rete. In NetFlow si hanno delle sonde, le quali vengono poste in dispositivi di rete, in modo tale da raccogliere informazioni sui flussi ed esportarli, una volta espirati, verso uno o più *collectors*, in Fig. 2.1 un semplice esempio di architettura.

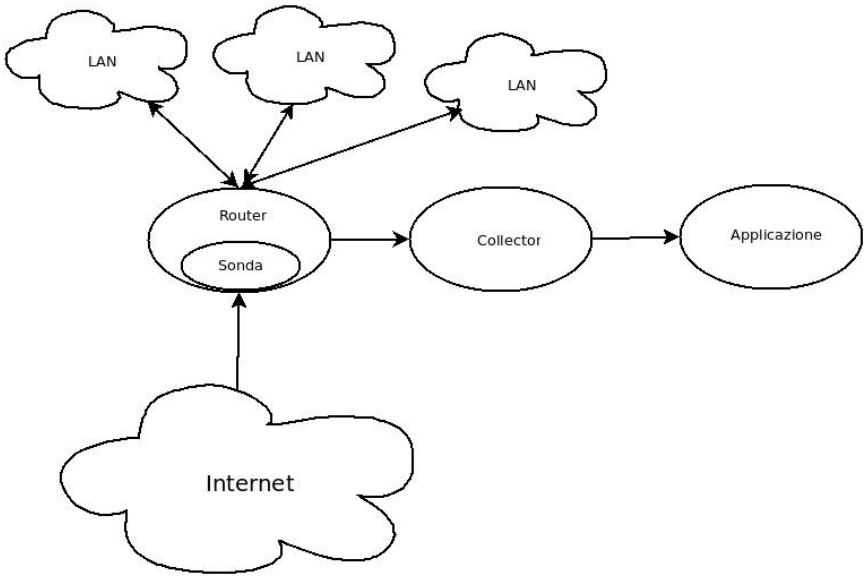


Fig. 2.1: Architettura di NetFlow

Tradizionalmente un flusso è definito da Cisco come l'insieme dei seguenti campi:

- Indirizzo IP sorgente
- Indirizzo IP di destinazione

- Porta sorgente
- Porta di destinazione
- Protocollo al livello 3
- Tipo di servizio (ToS)
- Interfaccia di input

Attraverso questi sette campi, viene definito un flusso. Nelle versioni antecedenti alla nona (v9), un flusso era definito come una sequenza unidirezionale di pacchetti i quali condividono i sette campi definiti sopra. Nel caso in cui, un solo campo sia diverso, allora formerà un flusso diverso. Dalla versione 9 del protocollo, i flussi sono bidirezionali, non vi è un formato specifico per i pacchetti, ma si ha un formato dinamico ed aperto ad estensioni, in Fig. 2.2 vi è un esempio di formato di un pacchetto NetFlow versione 9. Un flusso è terminato, quando vi è una delle seguenti condizioni:

- La comunicazione di rete è terminata, ad esempio quando un pacchetto TCP ha il flag SYN settato.
- É passato il tempo necessario per esportare il flusso.
- Il flusso non è più attivo, ovvero non è stato catturato nessun pacchetto appartenente ad esso nell'arco di tempo prefissato.
- Non vi è più spazio per memorizzare informazioni nella cache.

L'ultima versione di NetFlow è la v9 [22], nella quale vengono introdotti i *template*. Un *template* viene definito attraverso una collezione di campi, i quali possono essere aggiunti senza la necessità di cambiare la struttura di esportazione dei record. I *collector* saranno in grado di interpretare i campi di un template, in quanto i dati che riceveranno conterranno le informazioni sulla struttura dei campi del record del flusso. Per ridurre il volume dei dati esportati, il collector può esportare solo una parte dei campi.

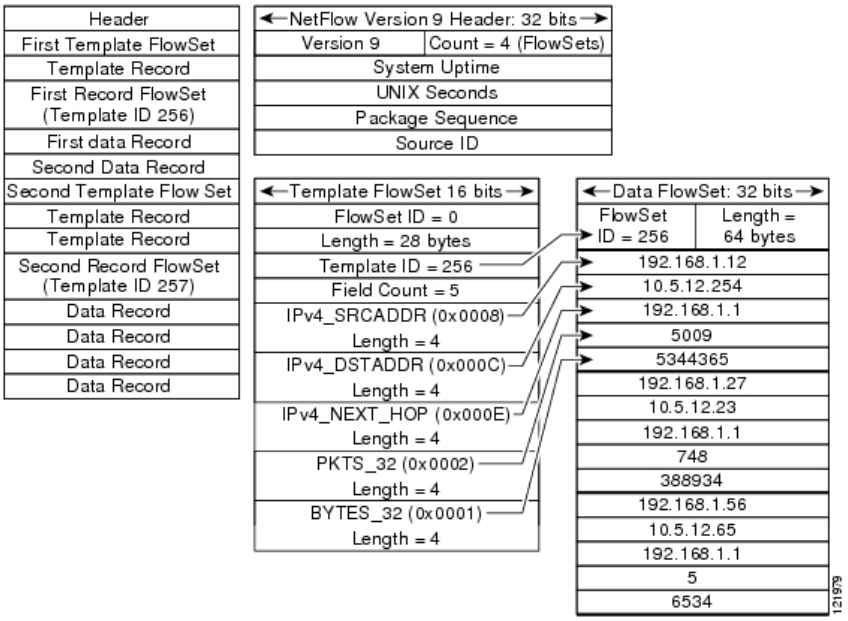


Fig. 2.2: Formato di un pacchetto NetFlow versione 9

IPFIX

La necessità di uno standard per esportare flussi di traffico di rete su IP, dai vari dispositivi di rete, ha dato vita a IPFIX (Internet Protocol Flow Information Export) [23], il quale è un *IETF Working Group*. Si basa su NetFlow versione 9, attraverso esso si ha la possilibità di definire nuovi campi di flusso utilizzando un formato standard. Il trasporto è basato su SCTP (*Stream Control Trasmission Protocol*), ed inolre viene supportato il trasporto su UDP/TCP. Le specifiche del protocollo IPFIX sono in fase di approvazione da parte del IESG (*Internet Engineering Steering Group*).

sFlow

sFlow è una tecnologia per il monitoraggio delle rete basata sui flussi, nella quale viene utilizzata la tecnica del *sampling* [1] per aumentare la scalabilità. L'architettura di *sFlow* comprende una sonda, la quale cattura il traffico e lo invia periodicamente ad un *collector*. Il formato dei pacchetti inviati al *collector* sono del tipo *sFlow Datagrams*, ovvero uno specifico formato specificato in [24].

RMON

RMON (*Remote Monitorin using SNMP*) [25] è uno standard proposto dall'IETF, basato su SNMP per il monitoraggio del traffico di rete. Esso utilizza l'`RMON MIB`, ovvero un'estensione di `MIB-2` [26]. Tramite esso è possibile analizzare tutto il traffico passante per una LAN, filtrare e catturare pacchetti. Il MIB è composto dai seguenti gruppi:

- `Statistics`: contiene delle statistiche sulle interfacce di rete monitorate.
- `History`: memorizza delle statistiche periodicamente, in modo tale da poterle utilizzare successivamente.
- `Alarm`: genera un evento nel momento in cui il valore misurato di una variabile supera una determinata soglia.
- `Host`: contiene delle statistiche associate ad ogni macchina rilevata nella rete.
- `Host Top N`: riporta le macchine sono in cima ad una lista ordinata in base ad una statistica.
- `Matrix`: memorizza delle statistiche relative alla comunicazione tra duo indirizzi.

[1] Con *sampling* si intende la tecnica in cui si analizza 1 pacchetto ogni X pacchetti ricevuti.

- `Filters`: tramite questo gruppo è possibile specificare dei filtri sui pacchetti.
- `Packet Capture`: contiene i pacchetti catturati.
- `Events`: controlla la generazione e la notifica di eventi da un dispositivo.

Un'estensione di RMON è dato da RMON2, il quale definisce altri gruppi, dando enfasi ai livelli di rete più alti, come il livello IP e il livello applicazione.

2.1.2 Metriche Principali

Di seguito verrano elencate alcune delle metriche principali, utilizzate nel monitoraggio di rete.

- *Availability*: identifica la disponibilità di un dispositivo di rete, può essere calcolata come la percentuale di tempo che un dispositivo di rete è disponibile.

- *Response Time*: è il tempo impiegato da un sistema per rispondere a un determinato input. Questa metrica è importante per applicazioni interattive, al contrario non è significativa per applicazioni di tipo *batch*.

- *Throughput*: con questa metrica si misura la quantità di dati che possono essere inviati su un collegamento, in un tempo specificato.

- *Utilization*: determina la percentuale di tempo che una risorsa è in uso, durante un tempo specificato.

- *Latency*: quantità di tempo che un pacchetto impiega per andare da una sorgente ad una destinazione.

- *Jitter*: varianza della latenza nel tempo.

2.2 Monitoraggio dei servizi di rete

Il monitoraggio dei servizi di rete, oltre a dare informazioni relative alle comunicazioni nella rete, deve fornire dettagliate informazioni riguardo ai servizi monitorati.

A differenza del monitoraggio orientato alla rete, nella quale si elaborano delle metriche basate sui pacchetti catturati, il monitoraggio orientato ai servizi, deve fornire delle informazioni dettagliate riguardo al servizio ed eventualmente sulla rete. In [27] vengono descritte le principali operazione da eseguire per poter effettuare il monitoraggio orientato ai servizi, come ad esempio l'ispezione del *payload*, attraverso il quale è possibile decodificare le primitive del servizio, oppure come il filtraggio a livello di servizio. Di seguito verranno descritte le principali operazioni da eseguire per poter identificare un servizio dal punto di vista della rete.

2.2.1 Ispezione del payload

Per poter decodificare opportunamente il servizio, è necessario effettuare l'ispezione del payload. Questa operazione in genere viene implementata da zero in ogni applicazione, come le librerie per la cattura di pacchetti, tra cui la più utilizzata `libpcap` [28], in cui però non vi è questa funzione; `NetBee` [29] dove però i codice sorgente viene rilasciato solo sotto richiesta. D'altra parte vi sono librerie commerciali, in cui non viene rilasciato il codice sorgente. Uno degli strumenti più utilizzati per l'analisi dei protocolli di rete è `Wireshark` [30], il quale però non è disponibile sotto forma di libreria, ma solamente come applicazione.

Una soluzione *open-source* è `PF_RING` [31], un framework modulare per il monitoraggio di rete, esso è in grado di incrementare significativamente le prestazioni della cattura dei pacchetti su hardware *commodity*; tramite questo framework si ha la possibilità di ispezionare i pacchetti e di poter filtrare pacchetti a livello applicazione, questo direttamente a livello *kernel*.

Le regole per il filtraggio sono specificate attraverso dei *plugin*, i quali rappresentano i componenti di analisi di `PF_RING`, essi sono implementati come moduli del *kernel*, i quali possono essere caricati dinamicamente. In questo modo, è possibile definire un *plugin* per poter filtrare i pacchetti per tutti i livelli di rete e così passarli ad un'applicazione in spazio utente.

In questo lavoro di tesi, l'ispezione del *payload* verrà delegato alla sonda per la cattura del traffico, la quale verrà descritta nel paragrafo 4.2.

2.2.2 Classificazione del servizio di rete

Una caratteristica importante, per quanto riguarda il monitoraggio di servizi di rete è la loro classificazione. Risulta non banale riuscire a interpretare la tipologia di un servizio, in quanto il protocollo utilizzato potrebbe non identificarlo. Ad esempio il protocollo HTTP, al giorno d'oggi viene impiegato per molteplici servizi, si pensi ad esempio alla web chat, allo *streaming* sul web, ai servizi di posta elettronica su web, oppure l'utilizzo di tecniche come *HTTP Tunneling*.

Vi sono diversi metodi per la classificazione del traffico di rete, il più semplice è quello mediante l'uso delle *Well Known Ports* [2], il quale risulta inefficace al giorno d'oggi, in quanto si ha una crescita sempre maggiore di nuovi protocolli e di servizi che non utilizzano porte standard. Un'altra metodologia si basa sull'ispezione del *payload*, ad esempio in [32] viene proposto un metodo, in cui si cerca di identificare classi di traffico e non specifici servizi. Un altro approccio si basa sull'analisi delle caratteristiche statistiche dei flussi di traffico [33], come ad esempio le distribuzioni dei tempi di interarrivo dei pacchetti, della loro lunghezza. In [34], viene proposta una metodologia, la quale cerca di trovare delle firme di applicazioni in trac-

[2] Quando viene ricevuto un pacchetto, si analizza l'intestazione per poterne ricavare il protocollo di trasporto e la porta utilizzata, e quindi identificare il servizio utilizzato in base al numero di porta.

ce di flussi, in modo tale da poter localizzare alcune applicazioni, come ad esempio uno specifico web browser, client per il servizio email, o lettore multimediale. I risultati ottenuti potrebbero essere utili ad un'amministratore di rete, il quale potrebbe utilizzarli per la valutare la sicurezza di una rete andando ad identificare per esempio l'utilizzo di software dannoso.

In questo lavoro di tesi, la classificazione del servizio di rete, verrà effettuata mediante un'euristica, la quale verrà spiegata in dettaglio nel paragrafo 3.3.

2.3 Framework per il monitoraggio dei servizi di rete

In letteratura vi sono alcuni framework per il monitoraggio di specifici servizi di rete. Di seguito vi saranno alcuni dei maggiori studi in questo campo.

2.3.1 OneClick: A framework for measuring network quality of experience

OneClick [35] è un framework progettato per studiare la qualità dell'esperienza utente, per quanto riguarda le applicazioni di rete. Esso si propone di analizzare la percezione utente mentre si sta utilizzando un'applicazione di rete. La percezione utente viene valutata attraverso metodologie soggettive e oggettive. Per la valutazione della qualità soggettiva, essi si basano su dei *feedback*, forniti dagli utenti nel momento in cui non sono soddisfatti della qualità del servizio di cui stanno usufruendo. Per valutare la qualità oggettiva dell'applicazione di rete, essi si basano su i metodi PESQ [3] [36] e

[3] *Perceptual Evaluation of Speech Quality*: è una famiglia di standard, i quali comprendono delle metodologie di test, per la valutazione automatica della qualità della voce, come percepità da un utente attraverso un sistema telefonico.

VQM [4] [37]. I loro esperimenti vengono effettuati su macchine equipaggiate con un *key logger* e un software per il monitoraggio del traffico. Durante l'esperimento essi memorizzano le condizioni della rete e i *feedback* forniti dall'utente. Dopo di che vengono analizzati i dati, descrivendo una relazione tra la soddisfazione utente e le condizioni di rete. Viene specificato che la qualità della rete può essere controllata sia passivamente che attivamente. La validazione del framework è stata effettuata attraverso due implementazioni, una per applicazioni di *Instant Messaging* e una per dei giochi online. La loro metodologia presuppone di conoscere a priori il tipo di applicazione utilizzata dagli utenti e di installare un key logger sulla macchina che offrirà il servizio.

2.3.2 HostView: Annotating end-host performance measurements with user feedback

HostView [38] è un framework per misurare l'esperienza utente per quanto riguarda i servizi di rete. In particolare in questo lavoro viene disegnato un framework composto dai seguenti moduli:

- Uno sniffer implementato attraverso la libreria `libpcap`.
- Un modulo che si occupa di associare le applicazioni con le relative *socket* aperte, attraverso l'uso di `gt` [39], uno strumento software in grado di associare il nome di esecuzione di un'applicazione con il flusso di rete relativo, quest'ultimo ricostruito attraverso i pacchetti catturati con `libpcap`.
- Un modulo che misura le prestazioni del sistema, ovvero viene misurato il carico della *CPU* ogni secondo, registrata l'interfaccia di rete attiva cambio e periodicamente vengono memorizzate le prestazioni della rete *wireless*, come la potenza del segnale ricevuto, il livello di rumore, la

[4] *Video Quality Metric*: metodo standard per la misura oggettiva della qualità video.

velocità di trasmissione. Inoltre tenta di trovare l'*autonomous system (AS)* della macchina utente ogni qualvolta che la macchina cambia indirizzo ip.

- Un modulo per recepire i *feedback* degli utenti.

Come si può notare, in questo framework si è scelto di determinare la qualità dell'esperienza utente, attraverso la misura di dati direttamente sulla macchina dell'utilizzatore. Come in *OneClick*, vengono richiesti dei feedback da parte degli utenti, ed inoltre per monitorare le prestazioni della macchina utilizzata, è necessario l'accesso ad essa.

2.3.3 EtE: Passive End-to-End Internet Service Performance Monitoring

Il framework *EtE* [40] si propone di misurare le prestazioni di un sito web. Esso monitora passivamente i pacchetti da un web server per determinare le caratteristiche delle prestazioni del servizio. Per svolgere questo compito, essi hanno sviluppato un metodo euristico composto da due passi e un meccanismo di filtraggio statistico per ricostruire accuratamente le differenti pagine accedute dagli utenti. L'architettura proposta si compone di quattro moduli, i quali sono:

- Network Packet Collector: questo modulo colleziona i pacchetti che transitano sulla rete attraverso *tcpdump* [41] e li memorizza, in modo tale da poter effettuare delle analisi *offline*.

- Ricostruzione Richiesta-Risposta: modulo che ricostruisce tutte le connessioni TCP ed estrae le transazioni HTTP dal *payload*. La ricostruzione di quest'ultime viene effettuata tramite una metodologia proposta in [42].

- Ricostruzione della pagina web: questo modulo ha la responsabilità di raggruppare gli eventuali oggetti all'interno di una pagina web e memorizzarli in un log.

- Analisi delle prestazioni e statistiche: modulo che effettua le misurazioni delle prestazioni di tutti gli accessi da parte dei clienti.

Questo framework dev'essere installato su un server web, come componente software per monitorare le transazioni su un particolare server, oppure può essere posto in un punto della rete in cui è possibile catturare tutte le transazioni HTTP di un web server.

2.3.4 eQoS: Provisioning of client-perceived end-to-end qos guarantees in web servers

eQoS [43] è un lavoro in cui viene proposto un metodo per monitorare e controllare i tempi di risposta, percepiti dal client, di un web server in presenza di un forte traffico. Il framework *eQoS* è composto da quattro componenti: un web server, un controller della *QoS*, un gestore delle risorse e un monitor della *QoS*. Quest'ultimo misura il tempo di risposta di una pagina web, attraverso una metodologia simile a *ksniffer* [44], percepito dal client. Il controller determina la quantità di risorse assegnate a ciascuna classe. Il gestore delle risorse, classifica e gestisce, le richieste effettuate dal cliente e alloca le risorse necessarie per le varie classi. Per garantire la qualità del servizio percepita dal client, essi hanno proposto un controller adattivo *fuzzy* [5] due livelli, per controllare le allocazioni di risorse nel web server.

[5] La logica *fuzzy* è una logica polivalente, in cui sono presenti più valori di verità, rispetto ai classici vero e falso, in essa è possibile attribuire gradi di verità compresi tra 0 e 1.

2.3.5 sMonitor: A non-intrusive client-perceived end-to-end performance monitor for secured Internet services

sMonitor [45] è un framework in grado di monitorare le prestazioni, lato server, di servizi su HTTPS. Esso cattura pacchetti in modo passivo da un sito web, dopo di che effettua un'analisi delle richieste HTTP, per inferire le caratteristiche degli accessi dei clients e misurare il tempo di risposta percepito dal client in tempo reale. Il metodo di analisi delle richieste HTTP, è basato su una loro osservazione, cioè che la prima richiesta HTTP, ovvero quella in cui si riceve l'oggetto base, solitamente rappresentata da un file HTML, di una pagina web, ha dimensione significamente più grande rispetto alle seguenti richieste per ricevere gli altri oggetti presenti nella pagina web. Questo in quanto tra le varie richieste si differenzia il campo `Accept` dell'intestazione della richiesta. In conclusione in *sMonitor* misurano la qualità del servizio web, attraverso la misura del tempo di risposta per ottenere una pagina web da parte del client. La cattura dei pacchetti è stata implementata attraverso la libreria pcap.

2.3.6 Riepilogo

Nella tabella 2.1 vengono riepilogati i framework analizzati, evidenziandone le caratteristiche principali.

	OneClick	**HostView**	**EtE**	**eQoS**	**sMonitor**
Servizio Specifico	Sì (Messaggistica Istantanea e Giochi Online)	No	Sì (Web)	Sì (Web)	Sì (HTTPS)
Accesso Macchina Utente	No	Sì	No	No	No
Richiesta di feedback	Sì	Sì	No	No	No
Monitoraggio	Attivo o Passivo	Passivo (`libpcap`)	Passivo (`tcpdump`)	Passivo (nel kernel)	Passivo (`libpcap`)
HTTPS	No	No	No	No	Sì
Metriche principali	Qualità audio e video in relazione alle condizioni di rete.	Relative al traffico di rete, prestazioni del sistema	Tempo di risposta di pagine web percepito dall'utente, utilizzo cache e numero di *Abort* di una pagina web.	Tempo di risposta di pagine web percipito dall'utente	Tempo di risposta di pagine web percepito dall'utente per servizi su HTTPS

Tabella 2.1: Riepilogo frameworks analizzati

Capitolo 3

Metodologia di Analisi del Traffico di Rete

In questo capitolo, verrà sviluppata la metodologia proposta, la quale prende in esame alcuni servizi che sfruttano il protocollo HTTP per il trasporto dei dati. Viene specificato come è possibile classificare, aggregare e tracciare il traffico su HTTP, come è possibile calcolare delle metriche relative all'esperienza utente riguardo alla navigazione web ed alla visione di filmati sul web.

3.1 Obiettivi della Metodologia proposta

La metodologia proposta ha come obiettivo quello di analizzare dei servizi di rete, rendendo disponibili delle metriche che da una parte evidenzino la qualità del servizio percepita dall'utente, in termini di *Quality of Experience* e *Quality of Service*, dall'altra mostrino il comportamento degli utenti durante la fruizione dei servizi analizzati. I risultati ottenuti da questa metodologia potranno così essere utilizzati dai fornitori dei servizi e dagli operatori di rete, i quali potranno intervenire in caso di degrado della qualità di un servizio, operando tramite un approccio *top-down*.

3.2 I servizi studiati

Com'è già stato discusso nel paragrafo 1.6, i servizi più utilizzati su Internet, usufruiscono del protocollo HTTP per il trasporto dei dati. Quest'ultimo è probabilmente il protocollo di rete più utilizzato su Internet al giorno d'oggi; tradizionalmente ha svolto un ruolo fondamentale per la navigazione web, tuttavia la costante crescita delle applicazioni web e dei servizi su Internet, hanno ampliato il suo utilizzo, impiegandolo per molteplici servizi. Il motivo sostanziale della crescita dei servizi Internet che utilizzano il protocollo HTTP, è dato dal paradigma richiesta/risposta tra *client* e *server*, il quale ha proprietà di essere robusto e sufficientemente flessibile per consentire l'uso di HTTP come protocollo di trasporto per differenti servizi, oltre che per la navigazione web. Dato il considerevole numero di servizi di rete esistenti, risulta impossibile studiarli e analizzarli tutti in un lavoro di tesi, si è scelto quindi di focalizzarsi solo su di alcuni questi servizi, ed in particolare, la navigazione web e la visione di filmati sul web. L'approccio impiegato per lo studio della *Quality of Experience* a livello di servizio di rete, potrà comunque essere attuato per tutti i servizi di rete a livello applicativo esistenti.

3.3 Classificazione del servizio di rete su HTTP

Alcuni dei servizi di rete che utilizzato HTTP, sono ad esempio: leggere, ricevere e inviare posta elettronica, *download* e condivisione di files, attività di *intstant messaging*, *streaming* di contenuti multimediali, il controllo e l'aggiornamento di un sistema operativo. Tutti questi servizi possono utilizzare il protocollo HTTP per il trasporto delle informazioni, da una parte perchè si vuole fornire un'interfaccia web per usufruire del servizio, dall'altra parte l'uso di HTTP garantisce il passaggio delle informazioni attraverso i *firewalls*. L'utilizzo del medesimo protocollo per il trasporto dei dati, da

parte di molteplici servizi di rete, rende necessario la classificazione di questi, sia per poter studiare la qualità percepita dall'utente, che ha usufruito di un determinato servizio, sia per poter dare una visione globale dei servizi utilizzati durante una sessione Internet. Una ricerca che si propone di effettuare la classificazione dei servizi di rete su HTTP, è quella presentata in [46], dove a differenza di questo lavoro di tesi, gli autori classificano il traffico su HTTP, per poter studiare l'evoluzione di Internet, ed in particolare per individuare l'andamento futuro dei modelli di traffico di rete, dei modelli di *social network*, e dei modelli econonomici della prossima generazione di Internet. L'euristica proposta per la classificazione dei servizi di rete, si basa su quella proposta dagli autori dell'articolo, la quale verrà illustrata di seguito.

3.3.1 Euristica per la classificazione dei servizi di rete

La classificazione viene effettuata suddividendo i servizi di rete in diverse classi. Le classi vengono determinate attraverso l'analisi di alcuni campi chiave dell'intestazione dei pacchetti HTTP. Attraverso questi campi, è stata composta un'euristica, la quale permette di determinare con sufficiente precisione la tipologia di servizio utilizzato. Nella tabella 3.1 vengono elencate le principali classi di servizio e i relativi campi da tenere in considerazione per poterle determinare.

Si noti che i servizi appena elencati sono solo una minima parte di quelli esistenti, per la classificazione di ulteriori servizi, sarà necessario aggiungere eventuali regole all'euristica.

3.4 Correlazione richieste HTTP

In questa sezione si vuole illustrare come verrà gestita la correlazione delle richieste HTTP. Verrà illustrato in dettaglio come avvengono richiesta

Classe di servizio	Campi HTTP
Navigazione Web	*Method*, *Request-URI*, *User Agent*, *Content-Type*
Download* di *File	*Request-URI*, *Content-Type*
Servizio Email sul Web	*Host*, *Request-URI*
Servizi multimediali	*Request-URI*, *Content-Type*
Aggiornamento Software	*Content-Type*, *User-Agent*
Servizio di Messaggistica sul Web	*User Agent*

Tabella 3.1: Classi di servizio ed euristica utilizzata

e relativa risposta, per l'accesso ad un contenuto sul web, in particolare verrà esposto come viene effettuato il tracciamento del traffico HTTP di una pagina web e degli oggetti contenuti in essa. Prima di spiegare la metodologia adottata, è utile fornire alcune informazioni riguardanti le connessioni HTTP, in particolare *Persistent Connection* e la tecnica di *HTTP pipelining*, le quali verranno spiegate nei paragrafi successivi.

3.4.1 *Persistent Connection*

In HTTP versione 0.9 e 1.0, una transazione HTTP composta da un ciclo richiesta-risposta, utilizza un'unica connessione TCP. Questo implica che per ogni richiesta-ricezione di un oggetto tramite HTTP, viene aperta una nuova connessione TCP. Con l'avvento di HTTP 1.1, la connessione TCP non è detto che venga chiusa dopo la prima transazione HTTP, ma è probabile che questa venga riutilizzata per la richiesta-ricezione di più oggetti. Questo meccanismo prende il nome *persistent connection*. Attraverso questa modalità si ha il vantaggio di ridurre il numero di connessioni aperte, diminuendo quindi il consumo di *CPU* e l'uso di memoria. Viene ridotto il numero di connessioni TCP aperte e quindi la congestione della rete. Si elimina l'*handshaking* successivo alla prima richiesta, e di conseguenza viene

ridotta la latenza nelle richieste successive alla prima. Gli errori possono essere riportati senza la chiusura della connessione TCP. Inoltre rende possibile l'uso di *HTTP pipelining* 3.4.2. Le odierne versioni dei browser più popolari supportano le connessioni persistenti.

3.4.2 *HTTP pipelining*

La tecnica chiamata *HTTP pipelining* permette di ridurre il tempo di ricezione dei dati, attraverso l'invio di molteplici richieste da parte del client verso un *socket*, prima di ricevere le relative risposte. Le risposte dovranno essere ricevute nello stesso ordine in cui sono state inviate le richieste, in quanto non è specificato un metodo esplicito di associazione tra richiesta e risposta. Nell'immagine 3.1 viene illustrato come viene ridotto il tempo per HTTP 1.1 combinato alla tecnica di *pipelining*.

3.4.3 Richiesta di una pagina web

Una pagina web nella maggior parte dei casi è composta da più oggetti, i quali possono essere di vario tipo, come ad esempio fogli di stile, librerie javascript, immagini e contenuti multimediali. Il *client*, per poter visualizzare correttamente una pagina web, dovrà richiedere al *server* tutti gli oggetti contenuti in essa. Come è stato illustrato in 3.4.1 e 3.4.2 , i *web browser* attuali, per ridurre il tempo di caricamento di una pagina, richiedono i vari oggetti all'interno di essa, sfruttando la connessione persistente e la tecnica di *HTTP pipelining*. Per poter tracciare correttamente il traffico web, è necessario correlare tutte le richieste relative ai singoli oggetti appartenenti alla medesima pagina. Di seguito vi sarà un'esempio di caricamento di una pagina web, da parte di un *browser*, il quale utilizzerà la connessione persistente.

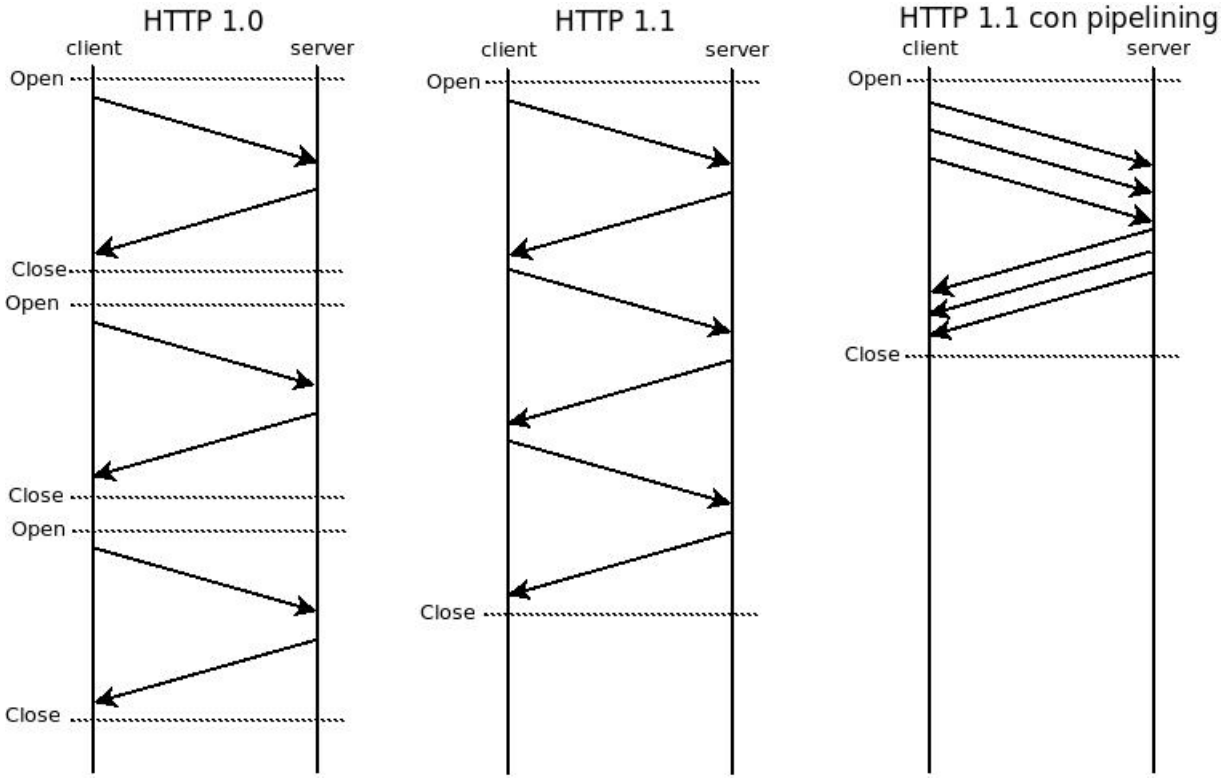


Fig. 3.1: Differenze tra HTTP 1.0, HTTP 1.1 e HTTP 1.1 con *pipelining*

Esempio di caricamento di una pagina web

Si pensi di visualizzare tramite un *web browser* una pagina web, la quale al suo interno è composta da tre immagini. Durante il caricamento della pagina, il browser aprirà più sessioni TCP, nell'esempio che segue si suppone che il browser apra due sessioni TCP. La prima sarà utilizzata per caricare il documento HTML e due immagini in sequenza (*Persistent Connection*). La seconda sessione verrà aperta per caricare la terza immagine; il caricamento di quest'ultima verrà effettuato in parallelo alle due immagini.

Come si può notare dalla Fig. 3.2, prima di poter instaurare la sessione HTTP con il server, verrà effettuata una *query* DNS per stabilire l'indirizzo ip della pagina richiesta[1]. Dopo di che verrà instaurata una sessione TCP con il server, mediante *handshake* a tre vie [47]. A questo punto il client richiederà la pagina al server attraverso una richiesta HTTP. Esaminando l'esempio, inizialmente viene richiesto il documento HTML, denominato `pagina.html`. Il server, ricevuta questa richiesta, preparerà il documento richiesto ed inizierà il traferimento al client. Il *browser*, ricevuto il documento HTML, lo analizzerà individuando gli oggetti contenuti nella pagina web. In questo esempio sfrutterà la connessione TCP aperta per richiedere la prima immagine, ed un'altra per poter richiedere la terza immagine in parallelo alla prima, la seconda immagine verrà richiesta, dopo aver ricevuto la prima immagine, mediante la connessione già instaurata all'inizio. Il numero delle connessioni aperte dipende dal tipo di client e dal numero di oggetti presenti nella pagina HTML.

3.4.4 Parametri fondamentali per la correlazione

Al fine di poter correlare correttamente gli oggetti presenti in una pagina web, è necessario analizzare alcuni parametri chiave delle richieste HTTP:

[1]La query DNS non verrà effettuata nel caso in cui al browser si fornisca direttamente l'indirizzo IP, oppure nel caso in cui l'indirizzo richiesto sia specificato nel file *hosts*.

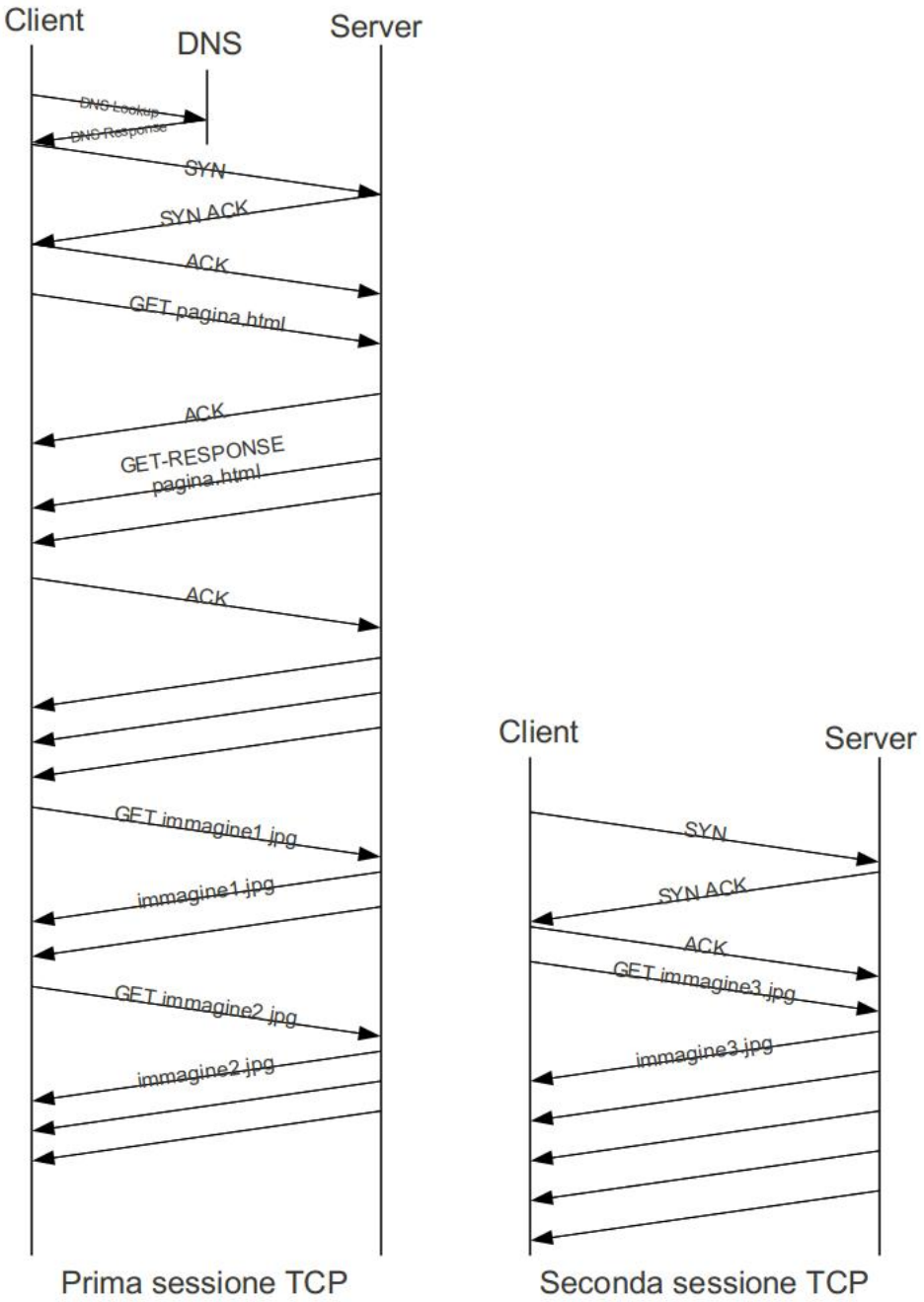


Fig. 3.2: Schema di una richiesta web

- Il tempo in cui è stata effettuata la richiesta.

- Indirizzo ip del client che ha effettuato le richieste.

- Il campo *Host* [48], che indica il server e il numero di porta in cui vi è la risorsa richiesta, nel caso in cui il numero di porta non sia stato specificato, allora è implicito il numero di porta standard per il servizio richiesto (ad esempio 80 per una URL HTTP).

- Il campo *User Agent* [48], il quale specifica il client che inizia la richiesta, questi solitamente sono *browsers*, *editors*, *spiders*, o altri strumenti in grado di effettuare richieste HTTP.

- Il campo *Request-URI* [48], è un *Uniform Resource Identifier* e identifica la risorsa per cui è stata effettuata la richiesta.

- Il campo *Referer* [48], il quale indica l'indirizzo della richiesta da cui è stata ottenuta la *Request-URI* della risorsa che si sta chiedendo.

- Il campo *Content-type* [48] specifica la tipologia di contenuto del body inviata al destinatario, oppure nel caso in cui sia stato utlizzato il metodo HEAD, indica la tipologia di contenuto che sarebbe stata inviata nel caso in cui la richiesta sia stata di tipo GET.

- Campo *Status-Code* [48], presente nell'intestazione della risposta da parte del server, esso è un codice a tre cifre di tipo intero, il quale identifica lo stato della risposta.

- Il campo *Location* [48], il quale è utilizzato per redirigere il mittente in un'altra locazione, piuttosto che quella indicata dal campo *Request-URI* [48] per poter completare la ricezione della risorsa richiesta.

3.4.5 Tipologie di richieste HTTP

Attraverso questi parametri è possibile discriminare alcune tipologie di richieste HTTP, utili per la correlazione del traffico web. Di seguito vengono elencate:

1. La prima richiesta, che sarà quella del documento HTML, avrà un *Content-type* del tipo *text/html*, *text/xml*, oppure *text/plain*. Analizzando questo campo e il campo *Request-URI*, sarà possibile determinare le richieste delle pagine HTML, le quali richiederanno gli oggetti contenuti in essa. In seguito questo tipo di richiesta verrà chiamata richiesta radice.

2. Richiesta in cui il campo *Referer* ha un valore, e questo valore corrisponde al campo *Host* concatencato al campo *Request-URI*.

3. Richiesta in cui il campo *Host* è uguale a quello di una richiesta radice, ma in cui il campo *Referer* non ha valore.

4. Richiesta in cui il campo *Host* è diverso da quello di qualsiasi richiesta radice, ma in cui il campo *Referer* ha un valore.

5. Richiesta in cui il campo *Location* ha un valore e il campo *Status Code* ha un codice della categoria redirezione, ovvero che ha un codice del tipo 3xx.

6. Richiesta in cu il campo *Host* è differente da qualsiasi richiesta radice ed inoltre il campo *Referer* non ha un valore.

3.4.6 Euristica utilizzata

Per poter identificare una prima richiesta, caso 1, sarà necessario analizzare il campo *Content-type*, nel caso in cui verrà richiesto un documento (html, xml, plain...). Le successive richieste sarà possibile correlarle a quella

iniziale usufruendo dei campi appena citati. Nei casi 2, 3 e 4 la correlazione può essere effettuata utilizzando i campi *Host*, *Request-URI* e *Referer*. Nel caso 5 si ha una redirezione verso il valore specificato nel campo *Location*, di conseguenza le richieste successive avranno quella locazione come campo *Host*, quindi è possibile correlare le richieste. Nel caso 6, a differenza degli altri casi, si ha qualche complicazione, in quanto non vi sono informazini esplicite che potrebbero far associare la richiesta con altre, in questo caso sarà necessario cercare di correlare la richiesta attraverso gli indirizzi ip, sorgente e destinazione, e i numeri di porta associati.

3.4.7 Il caso di HTTPS

Nel caso di HTTPS [49], ovvero il protocollo che integra l'interazione del protocollo HTTP attraverso un meccanismo di crittografia di tipo *Transport Layer Security (SSL/TLS)*, la metodologia per la correlazione del traffico, non è applicabile, in quanto l'intestazione dei pacchetti HTTP è cifrata. Una metodologia possibile per poter correlare parzialmente il traffico su HTTPS, è quella di ispezionare il contenuto dei pacchetti, in modo tale da estrarre il campo *Host* e quindi di tracciare parzialmente le richieste effettuate. In questa tesi non verrà trattato questo caso, ma solo prese in cosiderazione le prime richieste effettuate verso un determinato dominio, una futura integrazione potrebbe studiare in maniera più approfondita il caso di HTTPS, come ad esempio viene effettuato dal framework *sMonitor* descritto in 2.3.5.

3.5 Aggregazione del Traffico

Oltre a correlare e quindi tracciare il traffico HTTP, si vuole aggregarlo, proponendo delle metriche sul traffico generato dagli utenti. L'aggregazione viene effettuata analizzando tutte le richieste HTTP, suddividendole prima

in sessioni HTTP e poi aggregando quest'ultime in base al *server* in cui sono state effettuate le richieste. Come risultato si avranno delle metriche che evidenziano di quali servizi di rete ha usufruito l'utente, esponendo delle metriche relativi ad essi, come ad esempio il totale di traffico effettuato verso un determinato *Host*.

3.5.1 Sessione HTTP

La *Persistent Connection* 3.4.1 di HTTP 1.1, permette di effettuare richieste multiple verso un determinato *host* utilizzando la stessa connessione TCP, risulta quindi possibile raggruppare le richieste in un'unica sessione HTTP. Le richieste appartenenti alla stessa sessione HTTP, utilizzeranno la stessa connessione TCP, il quale implica che utilizzeranno le stesse porte, sorgente e destinazione. Risulta quindi possibile discriminare le sessioni HTTP, analizzando:

- Indirizzo IP sorgente.
- Porta sorgente.
- Indirizzo IP destinazione.
- Porta destinazione.

Una sessione HTTP, sarà caratterizzata inoltre da i seguenti parametri:

- *User agent* utilizzato per effettuare le richieste.
- Il campo *Host* verso cui sono state fatte le richieste.
- Il totale dei *bytes* scambiati nelle richieste.
- Il campo *Referer*.
- L'inizio della sessione.
- Il termine della sessione.

3.5.2 Aggregazione delle sessioni HTTP

Dopo aver suddiviso il traffico in sessioni HTTP, vi sarà un'aggregazione di queste, in modo tale da evidenziare verso quali *Host* sono state effettuate richieste e in quale arco temporale. L'aggregazione può essere effettuata prendendo in prima analisi le sessioni con campo *Referer* nullo; a queste verranno poi aggregate tutte quelle aventi aventi campo *Referer* uguale al campo *Host* delle prime. Le sessioni aggregate potranno essere caratterizzate dai seguenti campi:

- *Client*, l'indirizzo ip che ha effettuato la richiesta.
- *User Agent*, il *software* utilizzato per eseguire la richiesta.
- Il dominio verso cui sono state effettuate le richieste.
- Totale *bytes* scambiati.
- Inizio della sessione aggregata.
- Fine della sessione aggregata.

3.5.3 Aggregazione in sessioni globali

Un'ulteriore aggregazione effettuata, è quella in cui vengono aggregate le sessioni descritte in precedenza, in maniera globale. Ovvero dove non si discriminano le sessioni in base al *client* e allo *user agent* utilizzato, ma le richieste vengono aggregate in base al dominio verso cui sono state effettuate. Come risultato si ha una visione globale del traffico in uscita e in entrata, caratterizzando alcune metriche quali ad esempio il tempo totale di navigazione verso un *server*, il totale dei *bytes* scambiati verso esso, il tempo medio di risposta di un determinato server.

3.6 Analisi della qualità del servizio Web

Per quanto riguarda il servizio web, in questo lavoro di tesi, si è voluto effettuare una stima della qualità percepita dall'utente durante la navigazione, calcolando delle metriche che evidenziano eventuali degradi di qualità, ed inoltre che rendano una panoramica del traffico di rete analizzato. Nel paragrafo successivo, verrano spiegate le metriche calcolate.

3.6.1 Tempo totale di caricamento di una pagina web

Una delle metriche più importanti, per quanto riguarda il servizio web, è il tempo totale di caricamento di una pagina web. Per poter calcolare correttamente questa metrica, è necessario identificare tutte le richieste presenti nella pagina web richiesta, e determinare il tempo della prima richiesta e il tempo dell'ultima richiesta. Questo approccio, utilizzato in nel framework *EtE* 2.3.3, era un approccio più che valido quando non vi erano degli script che effettuavano richieste autonome, per il caricamento di oggetti dinamici all'interno della pagina. Come esempio si può pensare all'aggiornamente in tempo reale di un *feed* RSS, il quale effettua delle richieste periodiche verso un *server*, rendendo impossibile ricavare il tempo dell'ultima richiesta, e di conseguenza il tempo totale di caricamento di una pagina. Un possibile metodo per poter determinare questa metrica, è quello di analizzare il contenuto HTML della pagina richiesta, identificando tutti gli oggetti in essa, e quindi calcolare il tempo necessario per ricevere questi oggetti. Un metodo alternativo, potrebbe essere quello di applicare un'euristica che non tiene conto delle richieste successive ad un determinato lasso di tempo. In questo lavoro di tesi si è scelto di non fornire questa metriche, tuttavia l'ultimo approccio menzionato potrebbe essere integrato in una versione futura della metodologia.

Throughput

Con *throughtput* si vuole identificare il numero di bit ricevuti per secondo per una determinata richiesta. Nel caso si verifichino *throughput* bassi, questo potrebbe indicare una basso rendimento di un *server* rispetto ad un altro, rendendo l'esperienza utente insoddisfacente.

$$throughput = \frac{bit}{fine\_richiesta - inizio\_richiesta} \frac{bit}{sec}$$

3.6.2 Tempo di risposta

Attraverso il tempo di risposta, si vuole indicare il tempo necessario che intercorre tra la prima richiesta HTTP e il primo pacchetto in risposta. Tempi di risposta elevati, indicano una bassa qualità del servizio e di conseguenza un'insoddisfazione dell'utente, in quanto dovrà attendere un maggior tempo per usufruire del servizio.

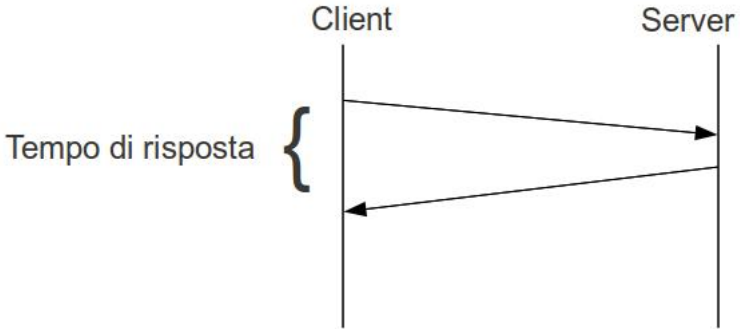


Fig. 3.3: Tempo di risposta di una richiesta web.

3.6.3 *User Impatience*

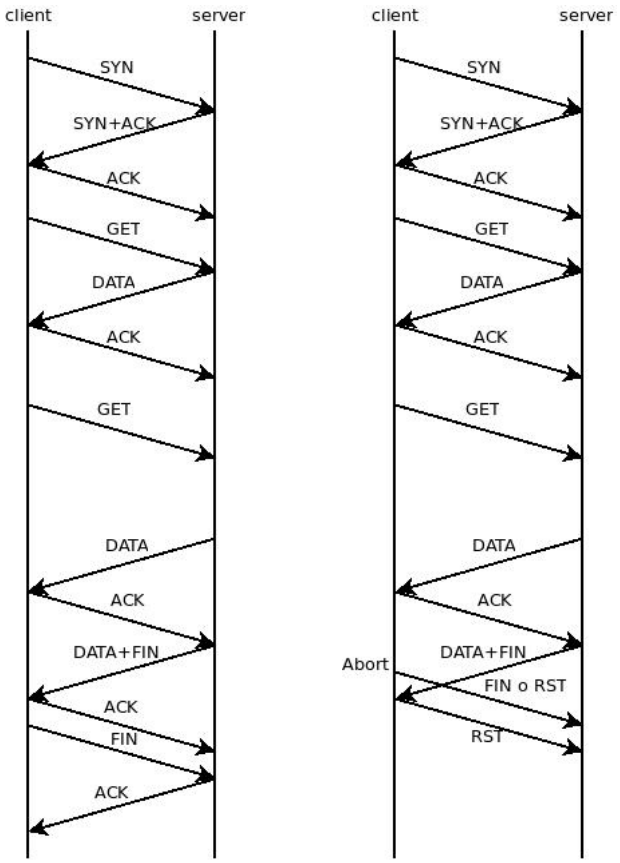
Un'altra importante metrica per misurare la qualità dell'esperienza utente, è il numero di *Abort* effettuati dall'utente. Questo fenomeno in letteratura viene denominato *User Impatience*. Gli utenti che usufruiscono del servizio web, tendono a valutare la propria esperienza, attraverso il tempo necessario

che intercorre tra la richiesta della pagina e la visualizzione di essa. Durante la navigazione web, l'utente risulta essere impaziente, disposto ad aspettare pochi secondi [50] prima che una pagina web venga caricata completamente. Nella maggior parte dei casi, l'utente che non visualizza in tempi brevi la pagina web richiesta, interromperà la richiesta, solitamente per riprovare a richiederla. Quindi l'interruzione del caricamento di una pagina da un'indicazione della qualità dell'esperienza utente avuta durante la navigazione. Per questi motivi, si è voluto tenere conto di questa metrica, evidenziando il numero di pagine web interrotte.

L'interruzione del caricamento di una pagina web, da parte dell'utente, può avvenire attraverso un click sui bottoni STOP o RELOAD del *browser* fintanto che la pagina web viene caricata, oppure cliccando su un link all'interno della pagina, quando questa non è stata ancora completamente ricevuta dal *client*. Nell'immagine 3.4 vi è un esempio che illustra l'interruzione da parte del client di una richiesta HTTP. Come si può notare, a differenza del normale caricamento di una pagina, nel momento in cui il browser effettua l'interruzione della ricezione, invia un pacchetto TCP avente *flag* FIN, oppure RST, settato. L'invio di un pacchetto con FIN o RST settato, dipende dal tipo di sistema operativo e dal tipo di *browser* utilizzato.

Per poter identificare correttamente un'azione di *abort* la quale indica un'insoddisfazione dell'utente, si è voluto prendere in considerazione i flussi che sono stati interrotti da parte del *client* e in cui il tempo di risposta del server è maggiore di una determinata soglia di tempo, la quale indica il tempo massimo per poter ricevere una pagina web, senza un degrado della qualità dell'esperienza utente. Si vuole quindi monitorare degli eventi di interruzione, ovvero quelle transazioni interrotte, prima che il server ha terminato di inviare dati.

Ricapitolando dal punto di vista del browser, un evento di interruzione, può essere generato attraverso l'interazione dell'utente con il browser, tra



(a) Richiesta Web senza *Abort*

(b) Richiesta Web con *Abort*

Fig. 3.4: Esempio di interruzione da parte di un utente durante il trasferimento dati dovuto ad una richiesta web.

cui:

- pressione del tasto $STOP$;
- pressione del tasto $RELOAD$;
- cliccare su di un link prima che la pagina sia completamente caricata;
- chiusura del browser.

Dal punto di vista di una connessione TCP, le operazioni elencate causano la chiusura di tutte le connessioni TCP utilizzate per la ricezione di tutti gli oggetti presenti all'interno della pagina web. Per ottenere queste informazioni, è necessario tenere traccia di tutte le connessioni TCP terminate dal *client*, ovvero quelle che sono terminate a causa di un pacchetto TCP avente *flag* FIN o RST settato inviato dal *client* verso il *server*. Avendo quindi tutte le sessioni HTTP, le quali sono state chiuse dal client, è necessario applicare un'euristica per poter determinare quali di esse possono essere state causa di un'eventuale degrado della qualità dell'esperienza utente. L'euristica proposta è simile a quella in [51], dove viene introdotta una condizione necessaria per determinare un flusso interrotto, chiamata *idoneità*, ovvero le connessioni TCP in cui il server invia dei dati, ma che non inviano un segmento FIN (o RST) e in cui il client invia un pacchetto con FIN (o RST) settato vengono chiamate idonee.

$$idoneo = not(FIN_S \; or \; RST_S) \; and \; DATA_S \; and \; (FIN_c \mid RST_C)$$

Questo criterio da solo non risulta sufficiente per poter distinguere tra le connessioni interrotte e quelle complete, in quanto in alcuni casi, per ottimizzazione interna dei *browser*, il *client* invia dei pacchetti aventi *flag* FIN (o RST), senza essere in presenza di *Abort* da parte dell'utente. Per questo motivo, oltre a questo criterio, è necessario tenere conto di una soglia

di tempo minimo (da ricerche effettuate in quest'ambito [50], viene indicato come tempo medio ragionevole una soglia di 4 secondi), corrispondente al tempo di risposta del *server*, la quale indichi che la risposta sia in ritardo. Quindi l'euristica finale può essere riassunta in questo modo:

$$idoneo = (not(FIN_S \; or \; RST_S) \; and \; DATA_S \; and \; (FIN_c \; or \; RST_C)) \; and \; (tempo\_risposta \; > \; soglia)$$

3.6.4 Altre metriche

Oltre alle metriche descritte, attraverso i dati raccolti, è possibile esporre delle metriche che evidenzino il comportamento degli utenti, in termini di traffico generato durante l'utilizzo dei servizi su HTTP. Di seguito verranno elencate alcune delle metriche:

- Totale dei bytes trasferiti per *user agent*.
- Totale dei bytes trasferiti per *client*.
- Totale dei bytes trasferiti verso un dominio.
- Tempo di risposta medio per sessione HTTP.
- Tempo di risposta medio per sessione aggregata.
- Tempo di risposta medio per dominio.
- Tempo totale di navigazione verso un dominio.
- Tracciamento temporale delle richieste effettuate in una sessione aggregata.

3.7 Servizio di Streaming

Nel paragrafo 1.6.2 si è parlato del servizio di *Streaming* specificando che in questo lavoro di tesi si analizzerà la qualità del servizio di *Streaming on demand* su HTTP. Questo servizio è caratterizzata dalle seguenti operazioni:

- Il *client* effettua la richiesta di visualizzazione di un determinato filmato al *server*.
- Il *server* risponde alla richiesta inviando il filmato richiesto progressivamente.
- Il *client*, ricevuta una parte di video iniziale, visualizza lo visualizza utente, continuando a ricevere i dati del filmato dal *server*, fino al completamento.

Il *client* una volta ricevuti le prime informazioni dal *server*, aspetterà a riprodurre il filmato, in modo tale da avere un *buffer* di dati, il quale è necessario per non interrompere la visualizzazione nel caso in cui ci siano ritardi di rete. Sul web attualmente vi sono molti portali che offrono questo servizio, i quali cercano di effettuare il servizio, offrendo una buona qualità in termini di esperienza utente. Attraverso la metodologia proposta di seguito, si vuole analizzare la qualità dell'esperienza utente per questo servizio. Questo è effettuato misurando il *throughput* effettivo della trasmissione dei dati del video e paragonandolo al *bitrate* richiesto per avere un'esperienza utente ottimale.

3.7.1 Un caso specifico: Youtube

In questo lavoro di tesi si è scelto di studiare il servizio offerto da *YouTube* [52], il quale permette agli utenti di condivedere filmati e di poterli visionare. Si analizzerà il servizio di *Streaming On Demand*, il quale oltre a offrire la possibilità di visualizzare i filmati caricati, permette di poter scegliere differenti

Qualità	Massima Larghezza	Massima Altezza
240p	400 *pixels*	240 *pixels*
360p	640 *pixels*	360 *pixels*
360p	480 *pixels*	360 *pixels*
480p	854 *pixels*	480 *pixels*
720p	1280 *pixels*	720 *pixels*
1080p	1920 *pixels*	1080 *pixels*

Tabella 3.2: Formati dei video su Youtube.

qualità per video. Per qualità si intende la risoluzione con cui è possibile vedere il video; *YouTube* offre la possibità di poter scegliere il video in differenti formati, i quali sono tutti a scansione progressiva [2]. I formati vengono identificati dalla risoluzione verticale in *pixel*, ovvero l'altezza verticale delle linee di scansione, seguiti dalla lettera p che identifica appunto la scansione progressiva. Nella tabella 3.2 vi sono i formati disponibili per la scelta da parte dell'utente [3]. Per garantire una buona esperienza utente, è necessario che per ognuno di questi formati, si abbia *throughput* maggiore o uguale al *bitrate* specificato nella tabella 3.3 [4].

L'utente sarà in grado di cambiare la qualità del video, mediante un

[2]La scansione progressiva indica un sistema per la visualizzazione e la trasmissione di immagini, che si contrappone alla scansione interlacciata. Nella scansione progressiva le linee di scansione che compongono le immagini sono scomposte una dopo l'altra, nella scansione interlacciata l'immagine è suddivisa in due semiquadri, uno contenente le linee pari e l'altro quelle dispari.

[3]la possibilità di scegliere diversi formati per un video, dipende dal formato in cui il video è stato caricato, quindi non è detto che per tutti i video, vi sia la possibilità di poter scegliere un formato di visione, oltre a quello standard

[4]I valori specificati non sono stati forniti da Youtube, ma sono dati approssimativi basati su dati statistici [53, 54, 55, 56].

Qualità	*Bitrate*
240p	0.25 Mbit/s
360p	0.5 Mbit/s
480p	0.8-1 Mbit/s
720p	2 Mbit/s
1080p	3.5-5 Mbit/s

Tabella 3.3: *Bitrate* necessari per una buona qualità dell'esperienza utente.

apposito link fornito da Youtube. Dal punto di vista della rete, questo equivale ad effettuare una richiesta GET, con dei parametri opportuni. Di seguito vi è un esempio di come viene effettuata la richiesta di un video da parte del browser e di quali parametri è necessario tenere conto per quanto riguarda la qualità del video scelto.

Esempio di richiesta di un video

La richiesta di un video viene effettuata attraverso una richiesta HTTP, avente come unico parametro l'identificativo del video che si vuole visualizzare:

```
http://www.youtube.com/watch?v=sZht_CtjNJc
```

In risposta a questa richiesta, viene inviato il codice HTML relativo alla pagina che verrà visualizzata all'utente. All'interno della pagina vi è uno script *javascript* che invierà la richiesta ad un *Content Delivery Network* [5]

[5]Un *Content Delibery Network* o CDN, è cluster di server che si occupano di distribuire contenuti su Internet nel minor tempo possibile. Solitamente i nodi sono distribuiti geograficamente, e la richiesta viene instradata al nodo più vicino geograficamente, oppure a quello meno carico di lavoro.

, il quale risponderà con il flusso audio/video. In figura 3.5 l'infrastruttura di *YouTube*. La richiesta in questione è la seguente:

```
v24.lscache6.c.youtube.com/videoplayback?sparams=id%
2Cexpire%2Cip%2Cipbits%2Citag%2Calgorithm%2Cburst%2Cfactor%
2Coc%3AU0dYTVNNU19FSkNNOF9LR1dH&fexp=900016&algorithm=
throttle-factor&itag=34&ipbits=0&burst=40&sver=
3&signature=0BB2B74E1D663CD32994157B5316B8849E8A879A.
BE5831E89A74CD5424E48D19707050B6ADD743A6&expire=1298325600&key=
yt1&ip=0.0.0.0&factor=1.25&id=b1986dfc2b633497
```

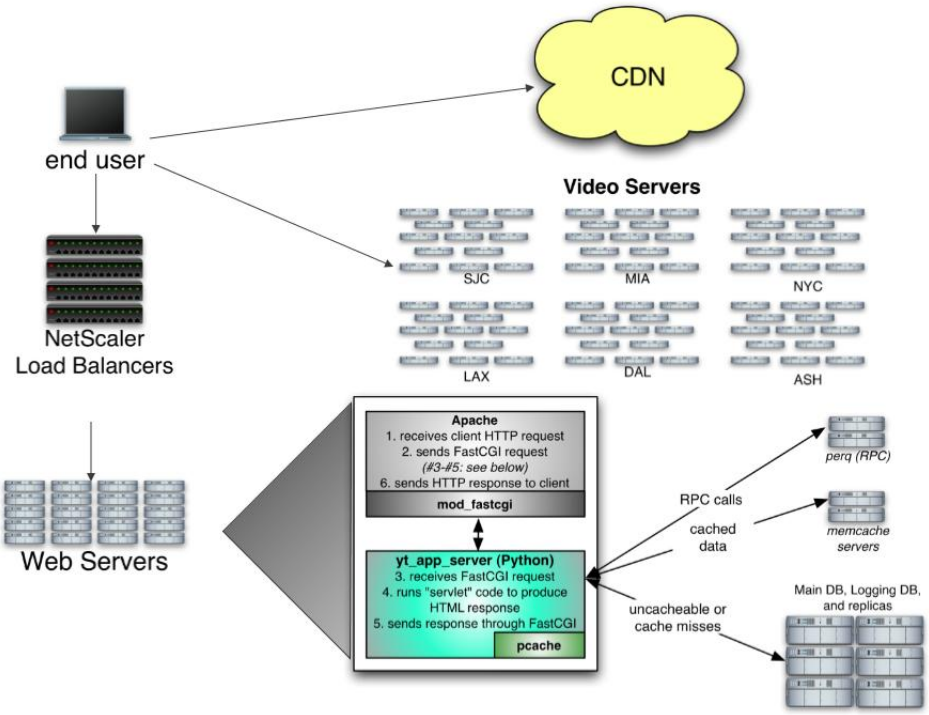


Fig. 3.5: Infrastruttura di Youtube

Di tutti i parametri di questa richiesta GET, è da tenere in considerazione il parametro **itag**, il quale identifica la qualità del video richiesta, che in caso di prima richiesta avrà il valore di default. In questo esempio, il parametro *itag* ha valore 34, il quale equivale a richiedere un formato del video di

Parametro *itag*	Qualità del video
5	240p
18	360p
34	360p
35	480p
22	720p
37	1080p

Tabella 3.4: Valori del parametro *itag* e relativo formato del video.

360p. Nella tabella 3.4 vengono illustrati i vari valori del parametro *itag* e i corrispondeti formati video. Avendo il formato del video, è possibile determinare il *bitrate* richiesto attraverso la tabella 3.3.

Esempio di cambio qualità di un video

Nel caso in cui l'utente cambi la qualità del video, attraverso l'interfaccia del player di Youtube, verrà effettuata dal lato della rete la seguente richiesta

```
v14.lscache3.c.youtube.com/videoplayback?sparams=id%
2Cexpire%2Cip%2Cipbits%2Citag%2Calgorithm%2Cburst%2Cfactor%
2Coc%3AU0dYTVRLVl9FSkNNOF9MRVpF&fexp=900016&algorithm=
throttle-factor&itag=35&ipbits=0&burst=40&sver=
3&signature=9E7501E4911E26E8D84706E2194FAE43861E9798.
90B69B27D64903CDD9CCC177FE8D5AF71340408D&expire=1298408400&key=
yt1&ip=0.0.0.0&factor=1.25&id=b1986dfc2b633497
```

Come si può notare il valore del parametro *itag* è cambiato da 34 a 35, il che significa che la qualità del video è cambiata da *360p* a *480p*, mentre il resto dei parametri risulta invariato.

3.8 Analisi della qualità del servizio di *Streaming*

L'analisi della qualità riguardante il servizio di *Streaming on demand* viene effettuata calcolando il *throuhput* relativo alla ricezione di un video e paragonandolo al *bitrate* richiesto.

Calcolo del *throughput*

Il calcolo del *throughput* viene effettuato, calcolando i *bytes* trasferiti per la ricezione del video e dividendoli per il tempo totale del trasferimento.

$$throughput = \frac{bytes\ totali}{fine\_richiesta - inizio\_richiesta}\ \frac{bytes}{sec}$$

3.8.1 Analisi della qualità

Dopo aver calcolato il *throughput*, è necessario identificare la qualità del video richiesta, attraverso il parametro *itag* nella *request-URI* del video, e quindi associare, attraverso i dati specificati nella tabella 3.4, il *bitrate* richiesto per una visione ottimale. Nel caso in cui il *throughput* sia maggiore o uguale al *bitrate* richiesto, allora il servizio sarà stato servito con una qualità buona, al contrario la qualità del servizio percepita dall'utente sarà insoddisfacente.

3.8.2 Altre metriche

Oltre all'analisi della qualità di un video, avendo i dati necessari è possibile calcolare altre metriche, quali:

- Totale traffico trasferito per il servizio di *Streaming on Demand*.
- Totale del tempo impiegato per il trasferimento dei video.
- *Throughput* effettivo per video.
- *Throughput* medio per *client* e *user agent*.

- Informazioni fornite da *YouTube* relative ad un determinato video.

Capitolo 4

Disegno e implementazione del Framework

In questo capitolo verrà illustrata l'architettura in cui è possibile collocare il framework, dopo di che verrà descritto il framework, il quale adotta la metodologia illustrata nel capitolo precedente per poter estrarre le metriche cictate. Verrano così esposte le scelte progettuali impiegate.

4.1 Architettura del Framework

L'architettura si compone di principalmente di tre componenti:

- `Sensori`: attraverso i sensori, i quali possono essere di tipo *software* o *hardware*, viene prelevato il traffico da analizzare e passato al framework per l'elaborazione delle informazioni.

- `Framework`: il framework si occupa di ricevere i dati forniti dai sensori e di elaborare delle metriche relative alla qualità dei servizi analizzati, queste metriche verranno poi messe a disposizione di client in grado di interpretarle.

- `Client`: il client prenderà i dati elaborati dal framework e li presentarà in qualche forma agli utenti.

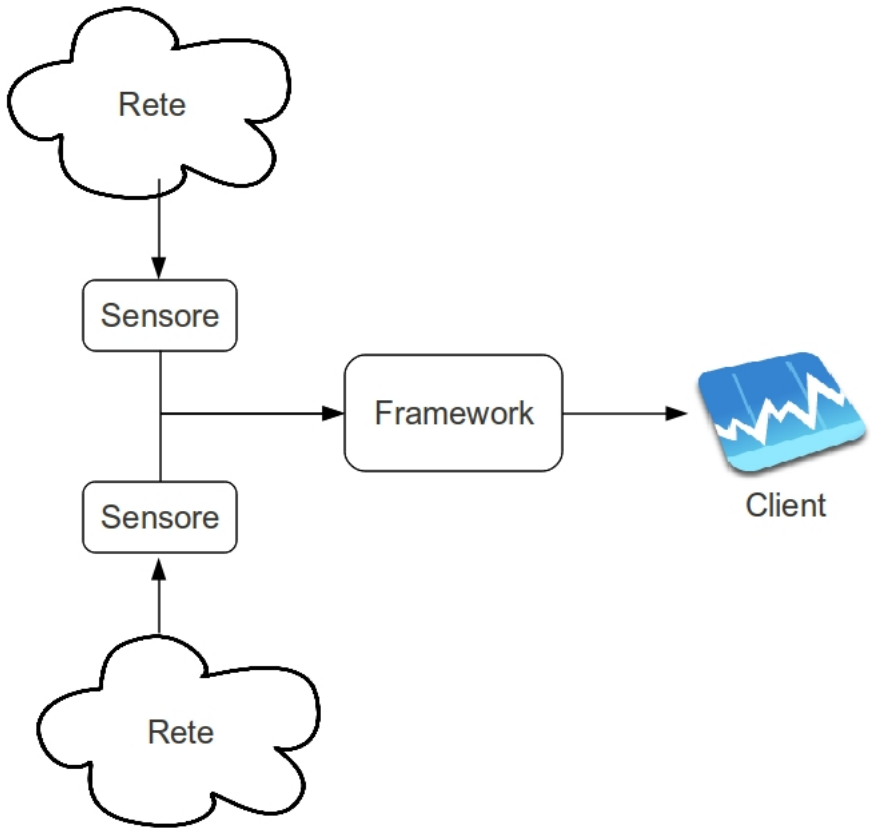


Fig. 4.1: Esempio architettura

L'architettura è stata suddivisa in questi componenti in modo tale da poter utilizzare più sonde distribuite sulla rete, si pensi ad esempio di dover monitorare più segmenti di rete, e di voler aggregare il traffico, in modo tale da poter effettuare delle analisi del traffico generato complessivamente; attraverso un'architettura modulare, questo risulta possibile.

4.2 Sensori

Per poter prelevare i dati da analizzare, sono utili delle sonde in grado di monitorare il traffico e di poterlo replicare al framework. Nel mondo del monitoraggio di rete, le sonde sono dei dispositivi di rete, *hardware* o *software*, in grado di catturare passivamente il traffico che transita attraverso la rete monitorata e di analizzarlo in *real-time*. Attraverso una cattura passiva del traffico di rete, si ha il vantaggio di monitorare il traffico, senza creare traffico artificiale, come avviene per la monitoraggio attivo dei servizi. In questo lavoro di tesi, si è scelto di utilizzare, come sonda *nProbe* [57], un progetto *opensource*, disponibile per i maggiori sistemi operativi (Unix, MacOS X, Solaris, Windows e ambienti embedded). *nProbe* ha la capacità di catturare in modo passivo il traffico sulla rete, e di esportarlo sotto forma di flussi, ed inoltre ha come funzionalità, quella di rendere disponibile un log contenente le richieste HTTP effettuate durante la cattura del traffico di rete.

4.2.1 Caratteristiche di *nProbe*

nProbe è un'applicazione *software* in grado di monitorare il traffico su un segmento *ethernet*, creando dei flussi *NetFlow* e di esportarli verso un collettore. Attualmente *nProbe* supporta gli standard *NetFlow v9/IPFX*, *IPv4* e *IPv6*, da la possibilità di salvare i flussi esportati in un DBMS, come *MySQL* e *SQLite*, è possibile salvare i flussi in formato *FastBit*, supporta nativamente *PF_RING* per la generazione di flussi ad alta velocità. Può essere utilizzato come collettore o proxy per i flussi, come si può notare dalle immagini 4.2, 4.3, 4.4. *nProbe* inoltre fornisce un log contenente tutte le richieste HTTP effettuate durante il monitoraggio di rete; questa caratteristica è stata fondamentale in questo lavoro di tesi, in quanto il log fornito, è l'*input* fornito al framework progettato, il quale si occupa di analizzarlo ed estrarre i dati relativi alla qualità dei servizi studiati.

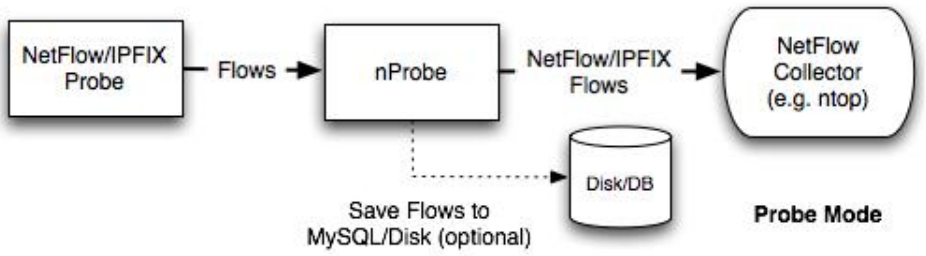


Fig. 4.2: Modalità *probe* (immagine presa da [58])

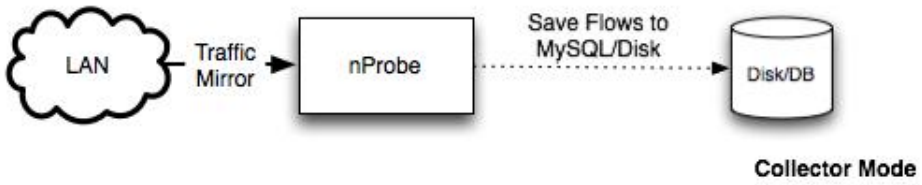


Fig. 4.3: Modalità *collector* (immagine presa da [58])

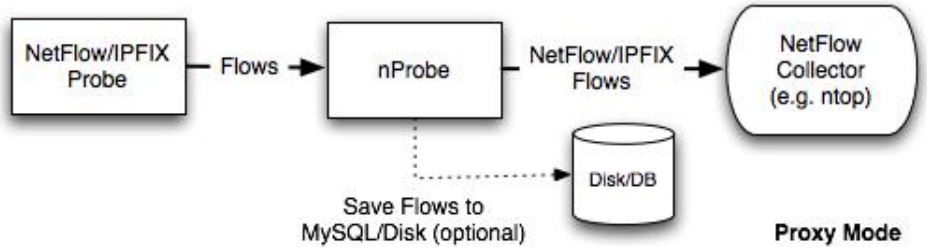


Fig. 4.4: Modalità *proxy* (immagine presa da [58])

4.2.2 Funzionamento di *nProbe*

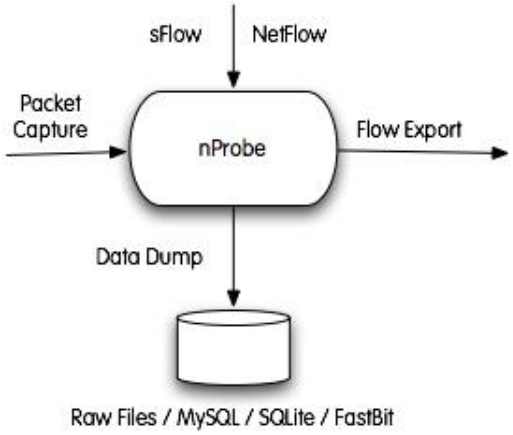


Fig. 4.5: Schema di *nProbe* (immagine presa da [58])

nProbe si suddivide in tre componenti logici, uno per l'aquisizione dei dati, uno per la rappresentazione in flussi dei dati e uno per la conversione dei flussi da interni a esterni. Il componente relativo all'acquisizione dei dati è responsabile della ricezione di essi, i quali possono essere pacchetti catturati su un'interfaccia di rete, attraverso la libreria *libpcap* o *PF_RING*, oppure dei flussi di rete in formato *NetFlow/IPFIX/sFlows* inviati da una sonda esterna a *nProbe*, in quest'ultimo caso *nProbe* funge da collettore. I dati ricevuti da *nProbe* sono letti sequenzialmente da una *socket*, vengono poi decodificati e posti in code per processarli. Ogni istanza di *nProbe* ha un coda per *core* (su una macchina con *dual core* vi saranno due code). I dati vengono inseriti all'interno di una coda determinata in base all'intestazione del pacchetto, in modo tale che i pacchetti che appartengono ad una medesima connessione, saranno sempre inseriti nella stessa coda. Ogni coda viene servita da un *thread* che elabora i dati di quest'ultima. I *threads* che elaborano i dati nella coda, non sono sincronizzati con quelli che li inseriscono in essa, le strutture dati utilizzate sono *lockless*, questo per migliorare significativamente le prestazioni della sonda. Ogni volta che un elemento viene

rimosso da una coda, questo viene utilizzato per aggiornare la rappresentazione interna del flusso. I dati sono memorizzati in una *hash table* statica, dove le collisioni sono gestite attraverso una *linked list*. Un *thread* in *background* visita periodicamente tutti gli elementi della *hash table*, ed esporta quelli che sono espirati (ad esempio un flusso finito, oppure un flusso che non riceve pacchetti dopo un determinato tempo). Gli elementi esportati verranno poi convertiti dalla rappresentazione interna ad un formato selezionato dall'utente (ad esempio *NetFlow v9*). Le informazioni da esportare verranno prima inserite in una coda e poi inviate in uscita.

Plugin

Il *core* di *nProbe* non è responsabile della codifica e la gestione dei pacchetti al di sopra del *layer* 4. Questa scelta è stata fatta per mantenere neutrale il *core* rispetto ai protocolli a livello applicativo. La gestione di questi viene effettuata tramite dei *plugin*, i quali si occupano della decodifica e della gestione di specifici protocolli. Per questo lavoro di tesi è stato utilizzato un plugin per il protocollo HTTP, il quale si occupa di raccogliere i dati specifici del protocollo HTTP e di fornire un log con questi dati.

4.2.3 Formato del log HTTP

Il log fornito da *nProbe*, è composto dai seguenti campi:

Modifiche al plugin HTTP di *nProbe*

Per ottenere i campi citati nel log, è stato necessario modificare il codice sorgente di *nProbe*, in quanto la sonda nella versione originale, non presenta i campi:

- *Location*: questo campo risulta utile in caso di redirezioni, e quindi per poter correlare correttamente una richiesta che viene reindirizzata verso un altro indirizzo.

Nome del campo	Descrizione
Client	Indirizzo ip del client.
Server	Campo *Host* della richiesta HTTP.
Protocol	Il protocollo utilizzato, il quale potrebbe essere HTTP o HTTPS.
Method	Metodo della richiesta HTTP.
URL	Indica la URL a cui è stata effettuata la richiesta.
HTTPReturnCode	Il codice di ritorno HTTP della richiesta.
Location	Campo *location* in caso di redirezione della richiesta.
Referer	Campo *referer* della richiesta HTTP.
UserAgent	Indica la tipologia di client utilizzato.
ContentType	*Mime type* della risposta.
Bytes	Totale dei byte per la richiesta.
BeginTime	Tempo di inizio della richiesta, in formato *Unix Time*[1].
EndTime	Tempo di fine della richiesta, in formato *Unix Time*.
Flow Hash	Risultato di una funzione hash, la quale viene calcolata sommando gli indirizzi ip sorgente e destinazione espressi come interi a 32 bit, ed i numeri della porta sorgente e quella di destinazione.
Cookie	Identificativo del cookie, il quale potrebbe essere utile per distinguere due utenti con medesimo indirizzo ip e medesimo *user agent*.
Terminator	Carattere che indica chi, tra il client e il server, ha chiuso la connessione TCP.
ApplLatency	Tempo che intercorre tra l'invio della prima richiesta e la ricezione del primo pacchetto da parte del client.

Tabella 4.1: Formato del Log HTTP fornito da *nProbe*.

- *Terminator*: attraverso questo campo è possibile determinare chi, tra client e server, ha chiuso la connessione TCP, e di conseguenza è un requisito per poter identificare delle azioni di *Abort* da parte dell'utente.

- *ApplLatency*: campo necessario per calcolare il tempo di risposta di un server, e quindi dell'eventuale degrado della qualità del servizio.

```
#
# Client Server Protocol Method URL HTTPReturnCode Location Referer UserAgent ContentType Bytes BeginTime EndTime Flow Hash Cookie Terminator ApplLatency
#
192.168.1.3 clients1.google.it http GET /complete/search?client=chrome&hl=en-US&q=www.goog 200
Mozilla/5.0 (X11; U; Linux i686; en-US) AppleWebKit/534.16 (KHTML, like Gecko) Ubuntu/10.04 Chromium/10.0.648.133 Chrome/10.0.648.133 Safari/534.16 text/javascript 2704 1300552560.945
1300552562.249 2449379136 43565 U 0.196
192.168.1.3 www.google.it http GET / 200 Mozilla/5.0 (X11; U; Linux i686; en-US) AppleWebKit/534.16 (KHTML, like Gecko) Ubuntu/10.04 Chromium/10.0.648.133 Chrome/10.0.648.133 Safari/534.16
text/html 12110 1300552560.48 1300552563.471 187070323 43565 U 0.346
192.168.1.3 www.google.it http GET
/csi?v=3&s=webhp&action=&srt=430&e=17259,28561,29014,29428&ei=ctuETc3WN8HdtAbp7YXrBQ&expi=17259,28561,29014,29428&imc=1&imn=1&imp=0&rt=xjsls.259,prt.260,xjses.342,xjsee.438,xjs.443,ol.527,iml.260
204 www.google.it Mozilla/5.0 (X11; U; Linux i686; en-US) AppleWebKit/534.16 (KHTML, like Gecko) Ubuntu/10.04 Chromium/10.0.648.133 Chrome/10.0.648.133 Safari/534.16 text/html 2875
1300552563.504 1300552568.379 187070338 43565 U 0.194
192.168.1.3 www.google.it http GET /ig/cp/get?hl=it&gl=it&bundleJs=1 200 www.google.it
Mozilla/5.0 (X11; U; Linux i686; en-US) AppleWebKit/534.16 (KHTML, like Gecko) Ubuntu/10.04 Chromium/10.0.648.133 Chrome/10.0.648.133 Safari/534.16 text/javascript 11950 1300552560.183
1300552568.533 187070323 43565 U 0.405
192.168.1.3 www.google.it http GET /s?hl=it&xhr=t&q=you&cp=3&pf=p&sclient=psy&safe=off&site=&source=hp&aq=&aqi=&aql=&oq=&pbx=1&fp=3b345a9cc4a57a&tch=1&ech=2&psi=ctuETc3WN8HdtAbp7YXrBQ13005525633781
200 www.google.it Mozilla/5.0 (X11; U; Linux i686; en-US) AppleWebKit/534.16 (KHTML, like Gecko) Ubuntu/10.04 Chromium/10.0.648.133 Chrome/10.0.648.133 Safari/534.16 application/json 3189
1300552560.183 1300552568.921 187070323 43565 U 0.384
192.168.1.3 www.google.it http GET /s?hl=it&xhr=t&q=youtu&cp=5&pf=p&sclient=psy&safe=off&site=&source=hp&aq=&aqi=&aql=&oq=&pbx=1&fp=3b345a9cc4a57a&tch=1&ech=4&psi=ctuETc3WN8HdtAbp7YXrBQ13005525633781
200 www.google.it Mozilla/5.0 (X11; U; Linux i686; en-US) AppleWebKit/534.16 (KHTML, like Gecko) Ubuntu/10.04 Chromium/10.0.648.133 Chrome/10.0.648.133 Safari/534.16 application/json 3404
1300552568.701 1300552569.323 187070339 43565 U 0.297
192.168.1.3 www.google.it http GET
/csi?v=3&s=web&action=&ei=eNuETa39KIjh4AaEvOCbCQ&e=17259,28561,29014,29428&cp=true&imp=2&pf=1&pfa=n.3,ttfc.344,ttlc.0,cbt.10&pfm=n.3,ttfc.506,ttlc.0,cbt.100&imn=7&rt=prt.186,pprt.186,ol.186,jsrt.718,iml.227
204 www.google.it Mozilla/5.0 (X11; U; Linux i686; en-US) AppleWebKit/534.16 (KHTML, like Gecko) Ubuntu/10.04 Chromium/10.0.648.133 Chrome/10.0.648.133 Safari/534.16 text/html 2647
1300552568.976 1300552572.238 187070339 43565 U 0.062
192.168.1.3 www.google.it http GET /s?hl=it&xhr=t&q=yo&cp=2&pf=p&sclient=psy&safe=off&site=&source=hp&aq=&aqi=&aql=&oq=&pbx=1&fp=3b345a9cc4a57a&tch=1&ech=1&psi=ctuETc3WN8HdtAbp7YXrBQ13005525633781
200 www.google.it Mozilla/5.0 (X11; U; Linux i686; en-US) AppleWebKit/534.16 (KHTML, like Gecko) Ubuntu/10.04 Chromium/10.0.648.133 Chrome/10.0.648.133 Safari/534.16 application/json
29386 1300552563.755 1300552572.375 187070338 43565 U 0.293
```

Una piccola porzione del log HTTP fornito da *nProbe*.

nProbe viene quindi utilizzato come sonda per catturare il traffico di rete, il quale viene analizzato e decodificato per produrre i files di log HTTP, i quali poi verranno elaborati dal framework, il quale computerà delle metriche indicanti la *Quality of Experience* relativa ai servizi analizzati.

4.3 Componenti del Framework

Il framework, come illustrato in figura 4.6 prende in input le richieste HTTP monitorate e resituisce in output delle informazioni elaborate ed aggregate relative alla qualità dei servizi analizzati. Si è scelto di non operare in *Real-Time*, ma di reperire le informazioni a blocchi, al fine di poter effettuare le analisi offline, infatti vi potrebbero essere delle necessità di operare in orari di meno carico, si pensi ad esempio a dei servizi critici, come quelli esposti da una banca, i quali solitamente effettuano le operazioni di analisi in orari notturni Il framework è stato sviluppato nel linguaggio *Java*, un linguaggio flessibile, orientato agli oggetti e indipendente dalla piattaforma. I dati di ingresso al framework, sono i files di log prodotti da *nProbe*, il framework prende queste informazioni, le elabora e fornisce in uscita i dati relativi al servizio analizzato. Le informazioni prodotte dal framework, verranno salvate internamente in una base di dati e potranno successivamente essere utilizzate da un client software.

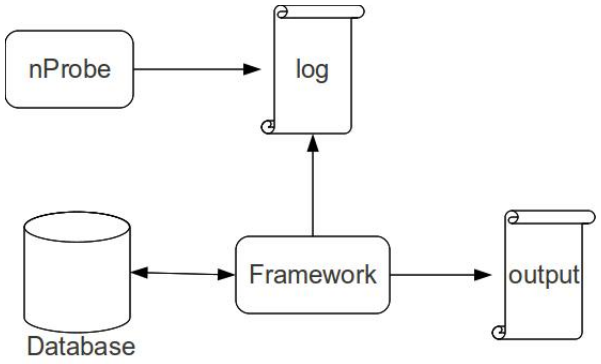


Fig. 4.6: Input e output del framework.

4.3.1 Struttura interna del framework

Il framework è stato disegnato seguendo un approccio modulare in ottica di semplificare lo sviluppo dei singoli componenti e di facilitare l'estensione di esso. Si è voluto quindi suddividerlo in vari gestori, i quali sono:

Main Attraverso esso si ha l'accesso alle informazioni del framework. Il client potrà utilizzare esso per usufruire delle funzionalità del framework.

LogParser Gestore che si occupa dell'analisi dei log HTTP in ingresso e della conversione di essi in strutture dati utilizzate dal framework.

ManagerDB Espone le funzionalità per la scrittura e la gestione della base di dati.

ManagerClassification Gestore che si occupa della classificazione del traffico HTTP.

ManagerTrace Modulo per la correlazione delle richieste HTTP.

MangerSessions Componente responsabile dell'aggregazione del traffico in sessioni.

ManagerVideos Gestore che si occupa dell'analisi dei video visionati dagli utenti.

4.3.2 Dati in input

Il framework prende in input il log prodotto dalla sonda, lo elabora e costruisce degli oggetti contenenti i campi presenti nel log. Questi oggetti, vengono in seguito correlati, caratterizzati e aggregati in modo tale da estrarre le informazioni riguardanti la qualità dei servizi analizzati. Il file di log, come illustrato nel paragrafo 4.2.3, presenta alcuni dati fondamentali per l'analisi del servizio. Il framework legge i log in ingresso, preleva

i dati e costruisce una lista ordinata in base al tempo di inizio delle richieste per mezzo della classe `LogParser`, dopo di che questi dati verranno salvati nella base di dati. Per evitare di utilizzare troppe risorse in termini di memoria principale, la memorizzazione delle informazioni nella base di dati, viene effettuata dopo un massimo di righe lette, dove questo valore massimo dipende dalla macchina utilizzata per l'esecuzione del framework, comunque nell'ordine di migliaia di *entry*.

La classe che rappresenta un'entrata del file di log è chiamata *Entry*, essa ha i campi corrispondenti a quelli del file di log, espone i metodi *getter* e *setter* per leggeri e valorizzarli. In aggiunta ai campi specificati nel file di log, vi sono anche i seguenti campi:

- *id*: un intero che identifica univocamente la richiesta, utile per ottenere una determinata richiesta salvata nella base di dati.
- *bad*: un booleano utile per identificare delle richieste formattate male.

Questa classe implementa l'interfaccia *Comparable*, e di conseguenza implementa il metodo *compareTo*, nel quale viene definito un ordinamento degli oggetti *Entry* in base al tempo di inizio della richiesta; questo metodo viene utilizzato per operare successivamente sulle richieste ordinate per tempo iniziale.

4.3.3 Classificazione del tipo di servizio

La classificazione della tipologia di servizio di rete, implementa l'euristica illustrata nel paragrafo 3.3, dove attraverso i campi dell'intestazione dei pacchetti HTTP è possibile determinare quale servizio è stato richiesto. Nel framework sono stati classificati i seguenti servizi:

- Navigazione Web.
- *Download* di file.

- Posta elettronica su web.
- Servizi multimediali.
- Aggiornamento software di sistema.
- Servizi di messaggistica su web.

In questo lavoro di tesi, al fine di poter validare la metodologia, sono stati presi in considerazione solo alcune tipologie di servizio. In un lavoro futuro sarà possibile arricchire il framework con altri controlli per poter classificare altri servizi. Nell'implementazione l'idea di base è quella di paragonare i campi delle richieste, con alcuni valori predefiniti, inseriti staticamente nel framework, in modo tale da identificare il servizio. I campi controllati per i vari servizi sono quelli descritti nella tabella 4.2, in cui vengono illustrati alcuni esempi reali del valore assunto dai campi in base al servizio utilizzato.

Tipologia di servizio	Campi controllati	Esempio
Navigazione web	content-type, user-agent, url, method	**content-type**: text/html, **user-agent**: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.9.2.13) Gecko/20101206 Ubuntu/10.04 (lucid) Firefox/3.6.13, **url**: /, **method**: GET
Download di file	**content-type**, url	**content-type**: application/pdf, **url**: /imc2010.pdf
Posta elettronica su web	server, url	**server**: it.mail.yahoo.com, **url**: /
Servizi multimediali	content-type, url	**content-type**: video/x-flv, **url**: /video-playback...
Aggiornamento software di sistema	content-type, user-agent	**content-type**: application/x-debian-package, **user-agent**: Ubuntu APT-HTTP/1.3 (0.7.25.3ubuntu9.3)
Servizi di messaggistica su web	user-agent	**user-agent**: MSMSGS.

Tabella 4.2: Servizi classificati dal framework.

La classe che si occupa della classificazione del servizio è *ManagerClassification*, la quale espone dei metodi statici che paragonano i campi di una determinata richiesta per determinare la tipologia del servizio utilizzato.

4.3.4 Correlazione delle richieste HTTP

Come descritto in 3.4, si vogliono correlare le richieste HTTP, in modo tale da determinare quali richieste HTTP vengono generate a partire da un'altra richiesta HTTP, potendo in questo modo ricostruire le richieste effettuate per la ricezione di una pagina web. Il gestore che si occupa della correlazione delle richieste HTTP è la classe `ManagerTrace`, la quale crea e gestisce degli oggetti di tipo `Trace`. Un oggetto `Trace` è costruito a partire da una richiesta HTTP, a questo oggetto viene poi assegnato un altro oggetto di tipo `Trace` che rappresenta la richiesta che l'ha generato, più semplicemente verrà chiamata richiesta padre, ed inoltre una lista di oggetti sempre di tipo `Trace` che rappresentano le richieste che sono state generate a partire dalla richiesta inziale, queste verranno chiamate richieste figlie. La lista delle richieste figlie potrebbero essere vuota e la richiesta padre potrebbe essere nulla. L'immagine 4.7 aiuta a descrivere la classe.

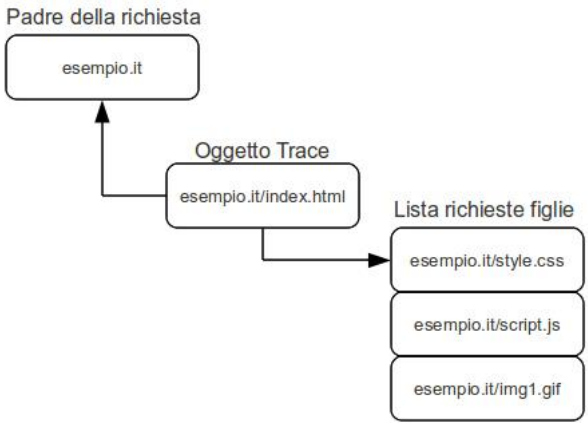


Fig. 4.7: Descrizione di un oggetto di tipo `Trace`

Attraverso questa struttura dati, si correlano tutti gli oggetti appartenenti una determinata pagina web, fino a formare un albero relativo alla pagina web presa in esame, in figura 4.8 un esempio di come potrebbe essere l'albero delle richieste costruito a partire da una richiesta iniziale, per comodità sono stati eleminati i riferimenti alle richieste padre.

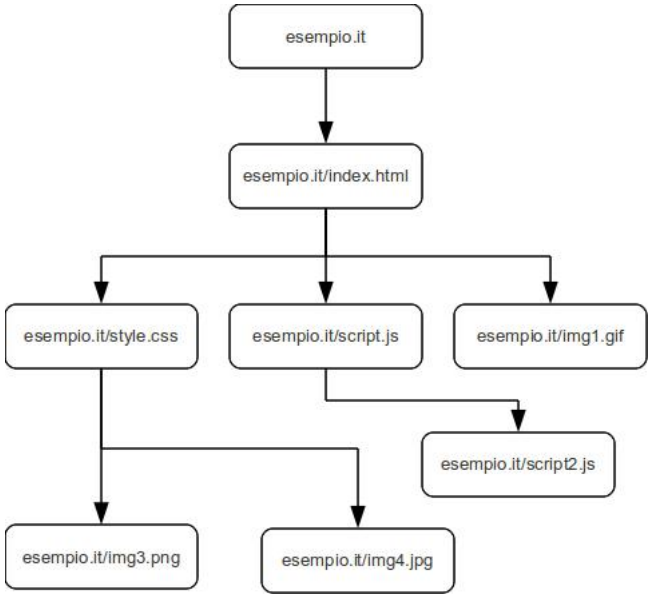


Fig. 4.8: Tracciamento degli oggetti presenti in una pagina web.

La costruzione di quest'albero, viene effettuata aderendo alla metodologia illustrata in 3.4.4. Nel framework è stato implementato un metodo statico, il quale, preso in ingresso un oggetto di tipo `Trace`, avente la richiesta che si vuole analizzare, una lista di oggetti `Entry`, che rappresentano le possibili richieste HTTP correlate, restituisce un oggetto `Trace`, rappresentante l'albero delle richieste correlate relative alla richiesta presente all'oggetto `Trace` passato come argomento.

Algoritmo

Il metodo inizialmente controlla che la richiesta non sia per un oggetto di tipo *favicon*, le quali solitamente non hanno il campo *referer* settato e non

generano richieste figlie, quindi in caso di un oggetto di questi tipo, esce dal metodo. Dopo di che viene effettuata un'iterazione sulla lista delle richieste passate come argomento e vengono effettuate le seguenti operazioni, dove per semplicità di scrittura, l'oggetto i-esimo della lista su cui si itera verrà chiamato `e` e l'oggetto di tipo `Trace` che si sta analizzando verrà chiamato `t`:

- Se la richiesta `e` è del tipo `favicon` allora:
  - Se il campo `flowHash` di `e` è uguale a quello di `t`, allora è sicuramente una richiesta figlia, quindi viene creato un oggetto `Trace` a cui viene passato `e` e come padre l'oggetto `t`, dopo di che viene aggiunto alla lista di richieste figlie di `t`.
  - Altrimenti viene controllato se il campo `server` di `e` è uguale a quello di `t`, allora è una richiesta figlia, quindi viene creato un oggetto `Trace`, con argomenti `e` e come padre l'oggetto `t`, ed in seguito viene aggiunto alla lista delle richieste figlie di quest'ultimo.
- Altrimenti:
  - se è una redirezione, allora viene confrontato il campo `location` di `e` con il campo `url` di `t`, nel caso siano uguali, viene creato un oggetto `Trace` con `e` come parametro. Dopo di che viene invocato lo stesso metodo ricorsivamente, al fine di aggiungere all'oggetto appena creato tutte le richieste figlie corrispondenti. Come risultato si avrà una porzione di albero identificata dalla radice, che è l'oggetto di tipo `Trace` appena creato. Questo viene aggiunto alla lista delle richieste figlie di `t`.
  - Se `e` ha il campo `referer` non nullo e questo corrisponde ai campi `server` e `url` concatenati, viene creato un nuovo oggetto `Trace`

con argomento `e`, viene invocato il metodo ricorsivamente su di esso ed infine viene aggiunto alle richieste figlie di `t`.

- In ultima analisi, nel caso in cui non si rientra nei casi appena descritti, viene controllato il campo `server` di `e` e in caso sia uguale a quello di `t`, allora viene creato un oggetto `Trace` con `e` come argomento e viene aggiunto alle richieste figlie di `t`.

Questo metodo può essere invocato su qualsiasi richiesta e restituirà un oggetto contenente una struttura ad albero, contente le richieste figlie generate dalla richiesta radice.

4.3.5 Aggregazione del traffico

La gestione dell'aggregazione del traffico viene affidata alla classe `ManagerSession`, la quale si occupa di creare delle sessioni HTTP e di aggregarle in oggetti che rappresentano appunto delle sessioni aggregate. L'aggregazione del traffico viene effettuata in due passi, prima vengono suddivise le richieste effettuate per `client` e per `user-agent`, quindi vengono aggregate le richieste appartenenti alla medesima sessione HTTP, al passo successivo vengono aggregate le sessioni in base al campo `host` e `referer` di queste. Di seguito verrano illustrati i passi effettuati e le scelte implementative.

Sessioni HTTP

Una sessione HTTP viene identificata mediante un oggetto chiamato `Session`, il quale ha i campi specificati nella tabella 4.3.

Nome del campo	Tipo	Descrizione
client	String	Indirizzo ip del client.
userAgent	String	`User-agent` che ha effettuato le richieste della sessione.
rootSite	String	Dominio verso cui sono state fatte le richieste appartenenti alla sessione.
referer	String	`url` da a cui sono correlate le richieste della sessione.
bytes	long	Totale del traffico generato dalla sessione in `bytes`.
sessionStart	double	Tempo iniziale delle richieste appartenenti alla sessione in formato *Unix time*.
sessiondEnd	double	Tempo finale delle richieste appartenenti alla sessione in formato *Unix Time*.
contentTypes	HashMap<String, Integer>	Tabella `hash` che tiene conto di tutti i *content-type* utilizzati nella sessione.
ids	ArrayList<Integer>	Lista di tutti gli identificativi delle richieste appartenenti alla sessione.
idKey	String	Identificativo della sessione, ottenuto concatenando il campo `flowHash`, il carattere '@' e il dominio verso cui sono state effettuate le richieste appartenenti alla sessione.
averageApplLatency	double	Tempo medio del tempo di risposta della sessione in secondi.
totalApplLatency	double	Tempo totale del tempo di risposta, utile per il calcolo della media di quest'ultimo.

Tabella 4.3: Descrizione dei campi di un oggetto di tipo `Session`.

Avendo la medesima connessione TCP, le richieste utilizzano gli stessi indirizzi IP e le stesse porte (sorgente e destinazione), dando la possibilità di poter aggregare più richieste in un'unica sessione, la quale viene identificata dal campo *flowhash*, calcolato dalla sonda e fornito nel log, e dal campo *Host* in cui sono state effettuate le richieste. La sonda calcola il *flowhash*, sommando i seguenti interi:

- Indirizzo IP sorgente, espresso come intero a 32 bit.
- Porta sorgente.
- Indirizzo IP destinazione, espresso come intero a 32 bit.
- Porta destinazione.

A questo intero viene concatenato il nome del dominio verso cui sono state effettuate le richieste, ottenendo l'identificativo della sessione. Il gestore delle sessioni, si occupa di creare una struttura dati, illustrata in figura 4.9, la quale è stata pensata per suddividere le sessioni HTTP in base al `client` e allo `user-agent`. Questa è composta da tre tabelle hash, una che ha come chiave l'indirizzo ip del client, la quale punta ad una tabella hash avente come chiave il campo `user-agent`, la quale punta ad un'altra che ha come chiave l'identificativo della sessione. Infine l'ultima tabella ha come valore un oggetto di tipo `Session`.

Algoritmo

L'algoritmo utilizzato per la creazione della struttura dati, effettua i seguenti passi iterando sulla lista di richieste:

- Nel caso in cui non vi sia il client della richiesta nella tabella hash, crea una nuova chiave con esso, altrimenti preleva il riferimento alla tabella dedicata agli `user-agent`.

- Nel caso in cui lo `user-agent` della richiesta in esame, non sia presente nella tabella hash relativa, viene creata una nuova entrata con il valore di esso, altrimenti viene preso il riferimento alla tabella relativa all'identificativo delle sessioni.

- Viene preso il riferimento alla tabella contenente l'identificativo delle sessioni, nel caso in cui non sia presente, viene creata un oggetti di tipo `Session`, altrimenti nel caso in cui fosse presente, vengono aggiornati i dati dell'oggetto, in particolare sommando i `bytes` dell'oggetto con quelli della richiesta in esame, aggiornando i valori di inizio e fine sessione, inserendo il `content-type` della richiesta nella tabella della sessione, inserendo l'identificativo della richiesta nella lista relativa, sommando il totale del valore del tempo di risposta e ricalcolando la media dei tempi di quest'ultimi.

Alla fine della computazione si avrà la struttura dati popolata con tutte le sessioni suddivise per `client` e `user-agent`. I dati verranno poi salvati nel database, in modo tale da poter poi essere recuperati per essere utilizzati da un software client.

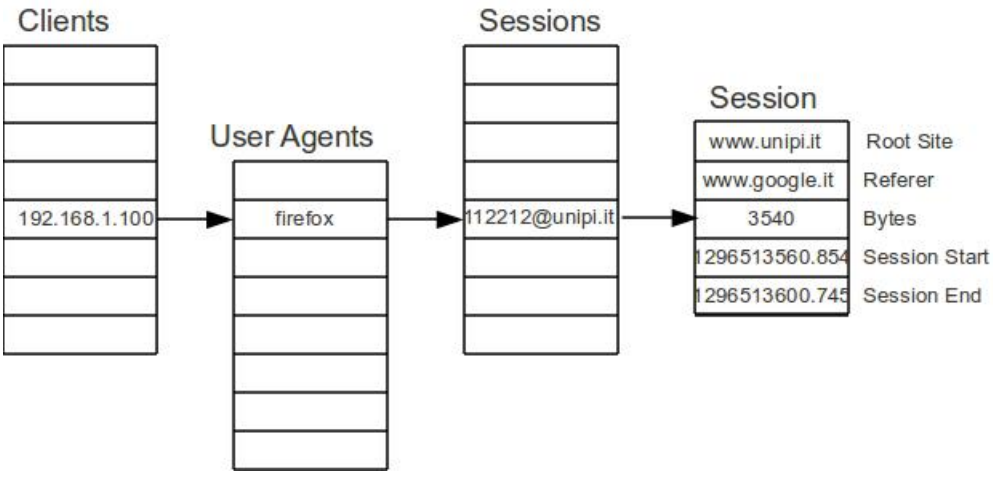


Fig. 4.9: Struttura dati utilizzata per la correlazione delle sessioni HTTP.

Aggregazione sessioni HTTP

Una volta creata la struttura dati contenente le sessioni, il framework, per mezzo del gestore `ManagerSessions`, le aggrega in base al loro campo `referer`, fornendo delle informazioni a grana più grossa. A questo scopo è stata creata una struttura dati denominata `AggregatedSession`, la quale presenta i campi illustrati nella tabella 4.4. Di questa classe è utile spiegare come viene effettuato il calcolo della durata totale di una sessione, il quale viene effettuato prendendo tutti gli intervalli di tempo delle sessioni appartenenti alla sessione aggregata.

Nome del campo	Tipo	Descrizione
client	String	Indirizzo ip del client.
userAgent	String	`User-agent` che ha effettuato le richieste della sessione aggregata.
site	String	Dominio verso cui sono state fatte le richieste appartenenti alla sessione aggregata.
bytes	long	Totale del traffico generato dalla sessione aggregata in `bytes`.
totalTime	double	Tempo totale in secondi della durata della sessione aggregata.
numberAggregatedSession	int	Numero delle sessioni facenti parte della sessione aggregata.
sessionStart	double	Tempo iniziale delle richieste appartenenti alla sessione aggregata in formato *Unix time*.
sessiondEnd	double	Tempo finale delle richieste appartenenti alla sessione aggregata in formato *Unix Time*.
sessionsIdKey	ArrayList<String>	Lista di tutti gli identificativi delle sessioni appartenenti alla sessione aggregata.
averageApplLatency	double	Tempo medio del tempo di risposta della sessione aggregata in secondi.
totalApplLatency	double	Tempo totale del tempo di risposta, utile per il calcolo della media di quest'ultimo.
numberEntries	int	Numero delle richieste effettuate appartenenti alla sessione aggregata, utile per il calcolo della media del tempo di risposta.
timeline	ArrayList<Time>	Lista di intervalli di tempo delle sessioni appartenenti alla sessione aggregata, questa lista è utile per il calcolo del tempo totale della durata della sessione aggregata.

Tabella 4.4: Descrizione dei campi di un oggetto di tipo `AggregatedSession`.

Calcolo della durata totale di una sessione aggregata

Al fine di poter calcolare accuratamente il tempo totale di una sessione aggregata, è stato necessario implementare una classe `Time`, la quale rappresenta un intervallo di tempo, definito da un inizio e da una fine. Questa classe espone un metodo statico che preso in ingresso una lista di oggetti `Time` e l'inizio e la fine della sessione che si vuole aggiungere alla *timeline*, ne resituisce una in cui gli intervalli sono stati fusi, in modo tale da poter avere una *timeline* precisa del tempo reale della durata della sessione aggregata. Nell'immagine 4.10 vi è un esempio dell'operazione di *merge*, in cui vi sono quattro intervalli di quattro sessioni, i quali verranno fusi, creando la *timeline* della sessione aggregata.

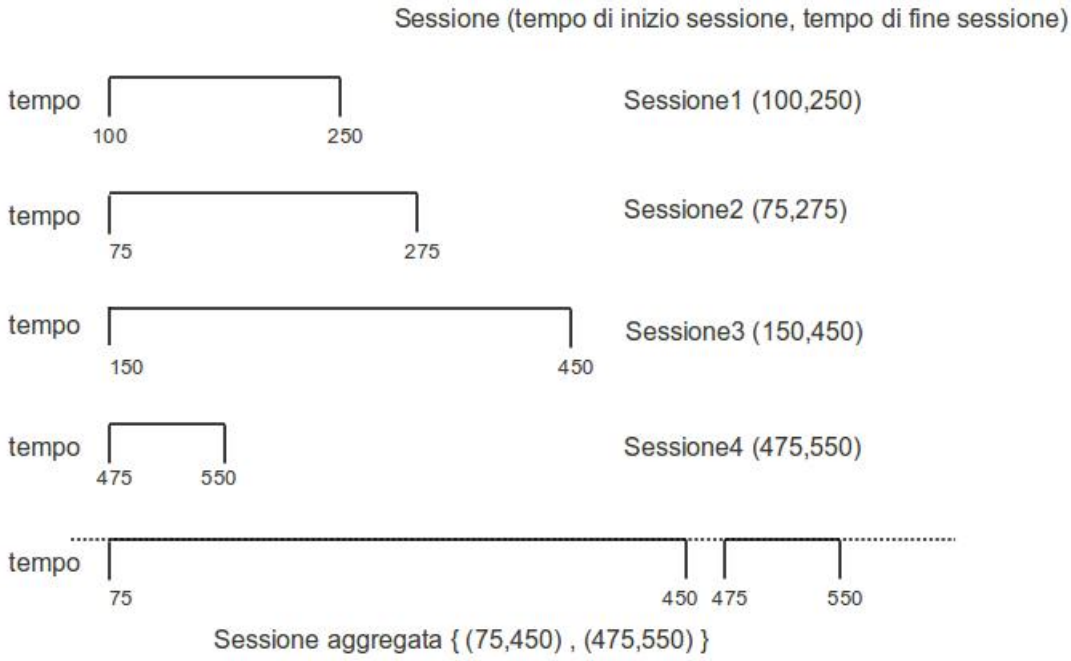


Fig. 4.10: Esempio di fusione degli intervalli di tempi relativi ad una sessione aggregata.

Il metodo prende come argomenti una lista di oggetti `Time`, e due `double` che rappresentano l'inizio e la fine della sessione di cui si vuole effettuare il

merge. Come prima operazione viene ordinata la lista in base al tempo di inizio degli intervalli, dopo di che si controlla se l'intervallo che si sta analizzando è incluso in un intervallo presente nella lista. In caso positivo, non vi sono altre operazioni da effettuare, in quanto la *timeline* non cambia, e il metodo resituisce la lista, in caso negativo, viene presa nota di tutti gli intervalli della lista che vengono intersecati dai tempi passati come argomenti. Nel caso in cui viene intersecato solo un intervallo, allora si guardano gli estremi di quest'ultimo e si aggiornano in base ai nuovi tempi passati come argomento. Se vi sono più intervalli intersecati, allora si crea una nuovo oggetto `Time` avente come tempo iniziale il minore fra questi e come tempo finale il maggiore, compresi i tempi passati come argomenti, si eliminano tutti gli intervalli intersecati e si aggiunge alla lista il nuovo intervallo, verrà quindi restituita la lista contenente la nuova *timeline*. Se invece non vi sono intervalli intersecati, allora viene aggiunto un nuovo intervallo con inizio e fine sessione uguali a quelli passati come argomenti.

Algoritmo

L'idea di base dell'algoritmo è quella di aggregare le sessioni in base al campo *referer* di esse, ovvero nel caso in cui una sessione abbia *referer* uguale al dominio di un'altra sessione, allora queste verranno aggregate, ottenendo come risultato delle sessioni aggregate per utente, in quanto verranno aggregate le sessioni aventi medesimo *client* e medesimo *user agent*. L'algoritmo itera le varie sessioni attraverso la struttura dati costruita per le sessioni 4.9, dove vengono prima prese in considerazione le iterazioni aventi `referer` nullo, queste verranno aggiunte ad una tabella hash, avente come chiave il dominio verso il quale sono state effettuate le richieste e come valore un oggetto di tipo `AggregatedSession`. Nel caso in cui la tabella hash, non abbia come chiave il dominio della sessione i-esima, allora questa verrà aggiunta, in caso contrario verrà effettuato un aggiornamento della sessione

aggregata, in cui verranno sommati i *bytes* totali della sessione aggregata con quelli della sessione i-esima, verrà aggiunta l'identificativo della sessione alla lista relativa, incrementato il numero delle sessioni presenti nella sessione aggregata, aggiornati il tempo di inizio e fine sessione aggregata, il numero totale di richieste HTTP, il totale del tempo di risposta e ricalcolata la media del tempo di risposta, ed infine verrà aggiornata la *timeline*. Dopo aver iterato le sessioni con *referer* nullo, verranno prese in considerazione le altre, dove nel caso in cui il *referer* sia uguale al dominio di una sessione aggregata, allora quest'ultima verrà aggiornata con i dati della sessione, altrimenti verrà aggiunta una nuova sessione aggregata alla tabella hash. Dopo aver iterato tutte le sessioni, verranno salvati i dati sul database, in modo tale da poterli recuperare in un successivo momento da parte di un software client.

Aggregazione in sessioni globali

Per poter dare una panoramica globale delle richieste effettuate dai vari utenti verso determinati domini, è stata effettuata un'aggregazione globale del traffico. La classe che rappresenta una sessione globale è `GlobalSession`, la quale ha i campi definiti nella tabella 4.5.

Nome del campo	Tipo	Descrizione
site	String	Il dominio verso cui sono state effettuate le richieste HTTP.
totalTime	double	La durata totale delle richieste effettuare verso il dominio.
bytes	long	Totale del traffico generato verso il dominio in `bytes`.
numberAggregatedSession	int	Numero delle sessioni aggregate relative al dominio.
totalApplLatency	double	Tempo totale del tempo di risposta, utile per il calcolo della media di quest'ultimo.
averageApplLatency	double	Tempo medio del tempo di risposta della sessione globale in secondi.
numberEntries	int	Numero delle richieste HTTP appartenenti alla sessione globale, utile per il calcolo della media del tempo di risposta.
listAggregatedSessions	ArrayList<AggregatedSession>	Lista delle sessioni aggregate facenti parte della sessione globale.
timeline	ArrayList<Time>	Lista di intervalli di tempo delle sessioni appartenenti alla sessione globale, questa lista è utile per il calcolo del tempo totale della durata della sessione aggregata.

Tabella 4.5: Descrizione dei campi di un oggetto di tipo `GlobalSession`.

Algoritmo

L'algoritmo utilizzato per creare le sessioni globali è semplice: viene analizzate le sessioni aggregate precedentemente elaborate, ordinate in base al campo `site`. Avendo la lista di queste, viene effettuata un'iterazione su di essa:

- Nel caso in cui sia la prima sessione analizzata, viene creato un oggetto `GlobalSession`, con i dati di questa, e viene salvato il riferimento all'oggetto appena creato.

- Se non è la prima sessione, allora viene controllato se la sessione globale creata al passo precedente, ha il medesimo dominio della sessione i-esima, in caso positivo viene effettuato l'aggiornamento della sessione globale, sommando i *bytes* della sessione i-esima al il totale della sessione globale, incrementato il numero di sessione aggregate, incrementando il numero di richieste effettuate, sommando il tempo di risposta totale della sessione i-esima a quello della sessione globale, calcolando la media del tempo di risposta, aggiungendo il riferimento della sessione aggregata alla lista di esse della sessione globale e infine effetuando il merge della *timeline*, con l'algoritmo descritto in 4.3.5 e ricalcolato la durata totale della sessione globale.

- In caso il dominio sia diverso, verrà creata una nuova sessione globale, con i dati della sessione i-esima, e sarà tenuto il riferimento ad essa per effettuare il controllo all'iterazione sucessiva.

4.3.6 Metriche

In questa sotto sezione verrà specificato come sono state ottenute le metriche specificate in 3.6.

Throughput

Il *throughput* di una richiesta è possibile calcolarlo prendendo il tempo di risposta di una richiesta e dividendolo per il tempo totale della richiesta, ovvero sottraendo al tempo di fine richiesta, il tempo di inizio richiesta.

Tempo di risposta

Il tempo di risposta delle richieste viene fornito dalla sonda, il framework fornisce anche i dati dei tempi di risposta medi relativi alle sessioni HTTP, alle sessioni aggregate e alle sessioni globali. Il calcolo è stato possibile in quanto, in tutte le sessioni viene tenuto conto della somma totale dei tempi di risposta delle richieste HTTP ed il numero totale delle richieste appartenenti ad una determinata sessione.

User Impatience

La determinazione delle richieste interrotte da parte dell'utente, è stata possibile grazie al dato `terminator` fornito dalla sonda. Questo dato può assumere tre valori:

- carattere 'C': il quale specifica che la richiesta è stata terminata da parte del client.
- Carattere 'S': richiesta terminata da parte del server.
- Carattere 'U': in questo caso la connessione TCP non è ancora terminata, di conseguenza essa verrà riutilizzata per effettuare altre richieste, in base al principio della connessione persistente di HTTP 3.4.1.

Avendo questi dati è possibile determinare quali richieste siano state terminate da un utente per mezzo di un'azione di *Abort*, aggiungendo inoltre una condizione di tempo di risposta minimo, come specificato in 3.6.3, in modo tale da eliminare eventuali falsi positivi.

4.3.7 Contenuti multimediali

Attraverso il framework disegnato si vuole dare delle informazioni riguardanti la visione di filmati su web. In questo lavoro di tesi si è preso in considerazione il servizio offerto da *YouTube* ed è stata analizzata la qualità dei video visualizzati attraverso il suo portale. Il gestore dei video è la classe `ManagerVideos`, la quale si propone di calcolare la qualità della visione dei video da parte degli utenti. Essa prima recupera tutte le richieste HTTP relative alla visione di un video attraverso *YouTube* [2], creando una lista di oggetti `Video`, i quali hanno i campi speficicati nella tabella 4.6.

[2] In ottica di una futura estensione del framework, sarà possibile modificare la classe per poter analizzare il servizio offerto da altri fornitori.

Nome del campo	Tipo	Descrizione
idVideo	String	Identificativo del video assegnato da *YouTube*.
client	String	Indirizzo ip del client.
userAgent	String	`User-agent` che ha effettuato le richiesta del video.
idEntry	int	Identificativo della richiesta HTTP.
link	String	*Url* del file video.
bytes	long	Totale del traffico generato per ricevere il video in *bytes*.
totalTime	double	Tempo totale impiegato per ricevere il file.
contentType	String	*Content type* della richiesta HTTP.
expectedBitRate	double	*Bitrate* richiesto per una visione ottimale del video in mBit/sec.
thoughput	double	*throughput* calcolato in mBit/sec.
goodQuality	boolean	*Booleano* che identifica se l'utente ha avuto una buona esperienza utente per la visione del filmato.
referer	String	*Url* alla pagina in cui l'utente ha potuto visionare il filmato.

Tabella 4.6: Descrizione dei campi di un oggetto di tipo `Video`.

Il costruttore dell'oggetto `Video` prende come parametro una richiesta HTTP, rappresentata da un oggetto di tipo `Entry`, attraverso questo è possibile calcolare il *throughput* effettivo e paragonarlo con il *bitrate* richiesto per un'esperienza utente ottimale durante la visione del filmato. Quest'ultimo viene determinato come descritto nella metodologia 3.7.1, ovvero analizzando la *url* relativa alla richiesta del video, estrando da essa il parametro `itag`, quindi recuperando il valore della qualità del video richiesto, da dei campi statici della classe aventi i valori descritti in tabella 3.4 e di conseguenza il valore del *Bitrate* richiesto con i valori corrispondenti specificati nella tabella 3.3. Gli altri campi di un oggetto `Video` vengono valorizzati nel costruttore di esso, a parte il campo `referer`, il quale viene elaborato successivamente prendendo le richieste aventi stessa connessione TCP, e di conseguenza medesimo *flowhash*, della richiesta HTTP relativa al file del video. Tra queste vi sarà la richiesta della pagina HTML che contiene il video, avente una *url* del formato:

```
http://www.youtube.com/watch?v=sZht_CtjNJc
```

la quale sarà il valore assegnato al campo `referer`. Il parametro `v` di questa *url*, è l'identificativo del video assegnato da *YouTube*, il quale permette di accedere alle informazioni del video attraverso le *API* di *YouTube*, tra le informazioni che è possibile reperire vi sono i vari formati video con cui è possibile visionare il filmato, il titolo, la durata del video e molte altre informazioni relative al video.

4.3.8 Dati in output

Il framework mette a disposizione per l'accesso alle informazioni calcolate l'oggetto `Main`, il quale da la possibilità di usufruire dei dati offerti da esso, utilizzando un approccio definito dal pattern *façade* [3]. Data la grossa

[3]Il pattern *façade* fornisce un unico punto di accesso al sistema, il quale permette l'utilizzo delle funzionalità di esso. Questo pattern offre i vantaggi di mascherare all'esterno

quantità di informazioni elaborate dal framework, si è scelto di utilizzare all'interno di esso una base di dati, come accennato nel paragrafo 4.3 in modo tale da non tenere in memoria principale troppe informazioni, appesantendo il sistema. Si è scelto di non dare accesso diretto alla base di dati ad un client, per semplificare lo sviluppo di quest'ultimo e quindi di delegare la gestione dei dati esclusivamente al framework. Alcuni dei metodi offerti dalla classe *Main* sono:

ArrayList<Entry> readInputFile(String dir) metodo che prende come argomento la directory in cui sono presenti i files di log e restituisce una lista di oggetti tipo `Entry` ordinati in base al tempo di inizio della richiesta.

public ArrayList<String> getClients() medoto che resituisce una lista degli indirizzi ip che hanno effettuato delle richieste HTTP.

public ArrayList<String> getUserAgents(String client) metodo che preso come argomento un client, resituisce la lista degli *user agents* utilizzati da esso per effettuare delle richieste HTTP.

public ArrayList<Session> getSessions(String client, String userAgent) metodo che prende come argomenti un client e uno *user agent* e resituisce la lista delle sessioni HTTP relative.

public ArrayList<AggregatedSession> getAggregatedSessions(String client, String userAgent) metodo che prende come argomenti un client e uno *user agent* e restituisce la lista di sessioni aggregate relative.

---

la complessità interna del sistema, semplificando lo sviluppo dei client che utilizzeranno la libreria, il quale utilizzerà un'unica interfaccia, rendendo il codice più leggibile e semplice da testare.

public ArrayList<GlobalSession> getGlobalsAggregatedSessions() metodo che resituisce la lista delle sessioni globali analizzate dal framework.

public ArrayList<Trace> getRootSites(String client, String userAgent) metodo che prende come argomenti client e *user agent* e restituisce una lista di oggetti `Trace` che rappresentano le richieste radice relative agli argomenti.

public Trace getTraceChilds(Trace t, String client, String userAgent) metodo che prende come argomenti un oggetto `Trace`, la quale rappresenta la richiesta di cui si vogliono correlare le richieste HTTP, client e *user agent* e restituisce un'oggetto `Trace`, il quale contiene i riferimenti alle richieste HTTP correlate a quella passata come argomento.

public ArrayList<Video> getVideos() metodo che restituisce una lista di oggetti di tipo `Video` che rappresentano le richieste HTTP relative alla visione di filmati su web.

Questi sono un'esempio dei metodi offerti dal framework. In caso di sviluppo futuro del framework, sarà possibile aggiungere altri metodi per poter reperire tutte le informazioni necessarie in base all'esigenze del singolo utente.

4.4 Client

Il client è un software che si interfaccia al framework, per utilizzare i dati elaborati da esso. In questo lavoro di tesi, al fine di validare il framework disegnato, è stato sviluppato un client, il quale utilizza la libreria descritta nel paragrafo precedente. Nel prossimo capitolo verranno illustrati i risultati

ottenuti attraverso l'elaborazione di dati reali, i quali sono stati visualizzato per mezzo del client.

4.5 Informazioni tecniche

Il framework è stato sviluppato utilizzando il linguaggio Java versione `JavaSE-1.6` [59], è stata utilizzata la libreria `sqlite3` [60], mediante il driver Java JDBC `sqlitejdbc-v056`, la quale ha il vantaggio di implementare un DBMS SQL senza la necessità di appoggiarsi ad un server, infatti il database è composto da un unico file; inoltre risulta essere leggero e veloce. La scelta di questo tipo di DBMS è ricaduta anche dal fatto che l'implementazione è solo una validazione alla metodologia proposta, in caso di implementazione del framework ad uso di produzione, probabilmente la scelta potrebbe ricadere su una tipologia di DBMS differente, in quanto `sqlite` supporta database di dimensioni relativamente piccole, non supporta funzionalità di clustering e con uno grossa quantità di dati, risulta essere meno perfomarmante rispetto ai classici DBMS.

Capitolo 5

Validazione

In questo capitolo si vuole validare e testare il framework proposto, andando ad analizzare i risultati ottenuti, paragonando il framework disegnato con altri framework citati nello stato dell'arte. Per poter visualizzare i dati ottenuti dal framework, è stato implementato un client, il quale prende in entrata i dati forniti dal framework, e mostra in uscita i dati sotto forma di grafici e tabelle. Il client sviluppato è una semplice applicazione web, la quale utilizza la libreria `JFreeChart` [61] per creare i grafici presentati.

5.1 Dati in ingresso

La validazione è stata effettuata prendendo il traffico fornito da *ISP* di media grandezza, quindi elaborato dalla sonda, la quale ha creato i files di log da passare al framework. I dati forniti sono stati opportunamente resi anonimi, prima di essere elaborati dalla sonda, modificando gli indirizzi ip sorgenti. Le richieste HTTP analizzate sono circa 21 milioni, le quali sono state memorizzate in file di log per un totale di circa 7 GB di spazio, il tempo di cattura del traffico è stato dalle ore 20 del 21 Marzo 2011 alle ore 8 del 22 Marzo 2011.

5.2 Risultati ottenuti

Nei seguenti paragrafi verranno descritti i risultati ottenuti dall'elaborazione dei dati in ingresso. Mostrando come i dati elaborati possono dare una panoramica sulla qualità dell'esperienza utente e sul comportamento degli utenti per quanto riguarda i servizi basati su HTTP.

5.2.1 Classificazione del servizio di rete

Presi in ingresso i dati, risulta possibile visualizzare la tipologia di servizi usufruiti da un determinato utente. L'immagine 5.1, illustra i servizi utilizzati dal *client* avente indirizzo ip 192.168.1.100, nell'arco di tempo che va dalle 22:32 alle 23:06. Come si può notare l'utente per lo più ha navigato sul web, ha prelevalto dei file, utilizzato la posta elettronica sul web, visualizzato qualche filmato sul web ed il sistema ha effettuato degli aggiornamenti software. Attraverso questa tipologia di dati, è possibile stimare il comportamento di un utente collegato alla rete, per quanto riguarda l'uso di specifici servizi basati su HTTP. Questo potrebbe essere utile anche per poter evidenziare eventuali richieste effettuate dalla macchina all'insaputa dell'utente, come ad esempio l'aggiornamento di software, oppure eventuali richieste scaturite da un virus.

5.2.2 Correlazione delle richieste HTTP

Il traffico viene correlato, come descritto nei paragrafi precedenti, dando come possibile risultato dei dati come quelli presentati nell'immagine 5.2. Da quest'ultima si può notare una struttura ad albero di richieste, dove la radice è rappresentata dalla pagina HTML, ed i figli sono gli oggetti contenuti in essa. Per ogni richiesta figlia, è possibile visualizzare il dettaglio della richiesta, come ad esempio mostrano l'immagine 5.3 e l'immagine 5.4, nella quale vi sono tutte le informazioni relative ad esse, nella prima si può

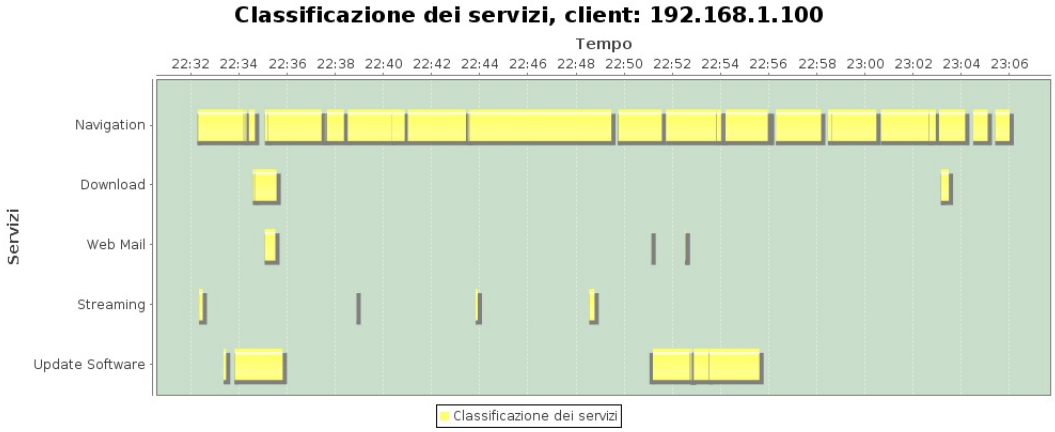


Fig. 5.1: Esempio di visualizzazione della classificazione dei servizi.

notare che l'esperienza utente non è stata soddisfacente, in quanto il tempo di risposta è elevato e l'utente ha terminato la sessione.

Da queste informazioni è quindi possibile determinare eventuali degradi di prestazioni, e quindi di possibile degrado dell'esperienza utente, relativa al servizio web.

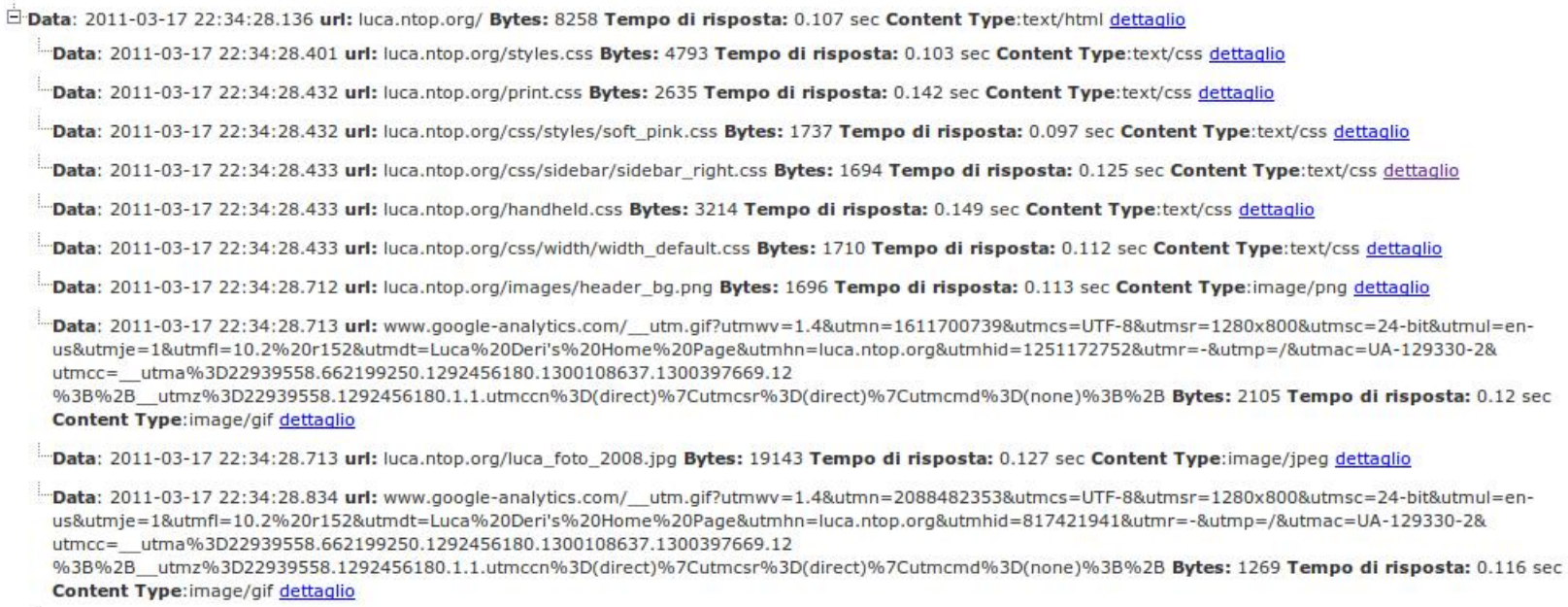


Fig. 5.2: Esempio di visualizzazione della correlazione delle richieste HTTP.

id	211378
client	----------
server	www.lavoro.gov.it
url	/Lavoro/Istituzionale/Ministero/UfficiTerritoriali/
Codice di ritorno	200
Referer	www.lavoro.gov.it/Lavoro
Location	--
User Agents	Mozilla/5.0 (Windows; U; Windows NT 5.1; it; rv:1.9.2.13) Gecko/20101203 Firefox/3.6.13 (.NET CLR 3.5.30729)
Content Type	text/html
Bytes	29046
Inizio richiesta	2011-03-22 09:36:04.234
Fine richiesta	2011-03-22 09:36:14.816
Totale tempo della richiesta	10.582000017166138 secondi
Tempo di risposta	10.375 s
Sessione chiusa dal	Client
Throughput	0.02195879792317636 Mbit/sec
Abort	Si

Fig. 5.3: Esempio di visualizzazione del dettaglio di una singola richiesta, in cui l'esperienza utente non è soddisfacente.

id	106238
client	----------
server	www.repubblica.it
url	/esteri/2011/03/22/dirette/diretta_libia_22-13933126/?fb_xd_fragment#?=&cb=f355d3c3d571a7a&relation=parent&transport=fragment&frame=f344d71533a6568
Codice di ritorno	200
Referer	www.repubblica.it/esteri/2011/03/22/dirette/diretta_libia_22-13933126
Location	--
User Agents	Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322; .NET CLR 2.0.50727)
Content Type	text/html
Bytes	14752
Inizio richiesta	2011-03-22 09:25:51.856
Fine richiesta	2011-03-22 09:25:55.120
Totale tempo della richiesta	3.2639999389648438 secondi
Tempo di risposta	0.019 s
Sessione chiusa dal	--
Throughput	0.03615686342121318 Mbit/sec
Abort	No

Fig. 5.4: Esempio di visualizzazione del dettaglio di una singola richiesta, in cui l'esperienza utente è soddisfacente.

5.2.3 Aggregazione del traffico

Attraverso l'aggregazione delle sessioni effettuato in precedente, è possibile visualizzare le richieste effettuate verso un dominio da parte del client, in modo tale da avere una panoramica del comportamento dell'utente. Attraverso le sessioni aggregate è possibile ad esempio visualizzare in quali orari l'utente ha effettuato delle richieste verso un determinato dominio, come ad esempio illustra l'immagine 5.5, il quale mostra il traffico generato a partire dalle ore 17:30 circa alle ore 17:44.

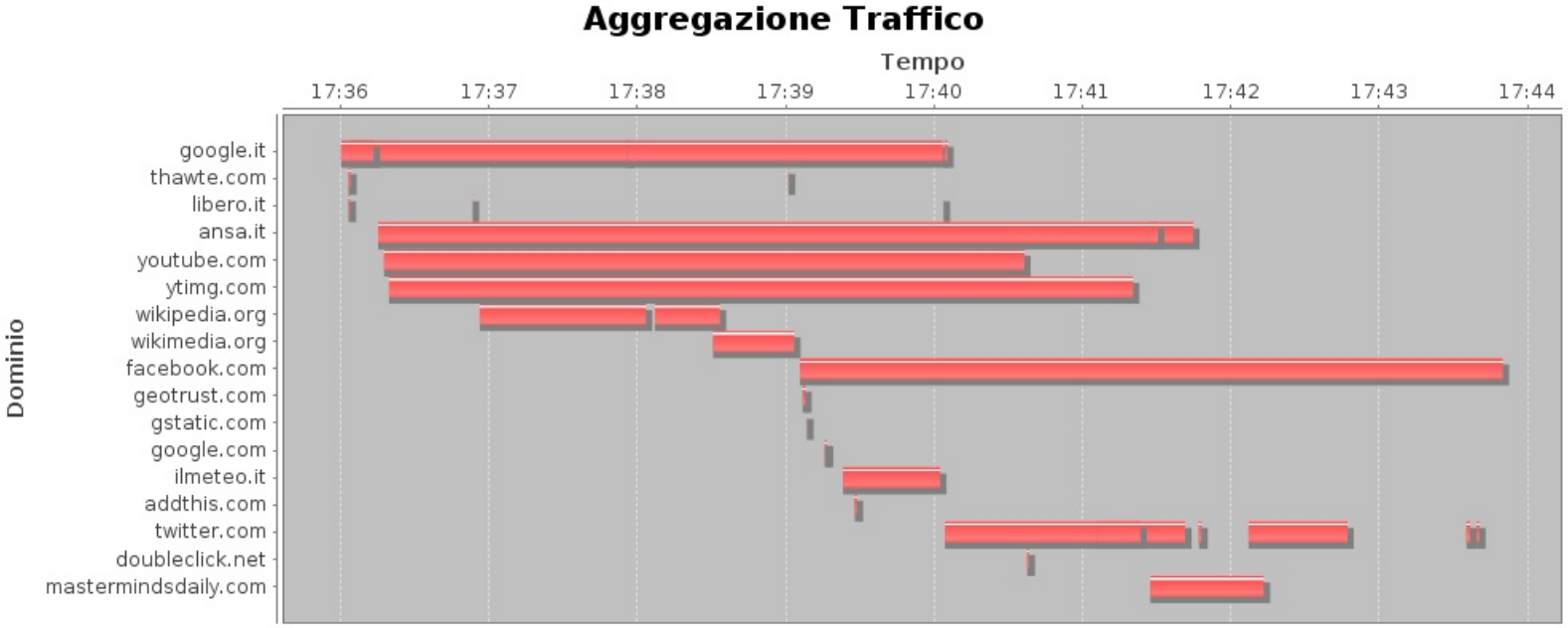


Fig. 5.5: Esempio di visualizzazione delle sessioni aggregate per un determinato client.

5.2.4 Sessioni Globali

Il traffico viene aggregato anche in sessioni globali, attraverso il quale è possibile visualizzare i domini più richiesti nel tempo dagli utenti. Risulta possibile effettuare dei grafici che indichino giornalmente per un determinato dominio le ore in cui vi sono state più richieste, come illustrato in figura 5.7, mostrare come varia il tempo di risposta durante il giorno 5.8, in modo tale poter determinare eventuali problematiche verso un dominio spefico. Negli esempi è stato illustrato il traffico relativo al dominio `ebay.it`. Risulta anche possibile visualizzare i domini più visitati, il numero di minuti di navigazione complessivi verso di essi e il totale dei *bytes* trasferiti verso essi 5.6. Come si può notare dalle immagini, il traffico è stato catturato e analizzato dalle ore 20 circa alle ore 8 del giorno successivo.

5.2.5 Contenuti multimediali

Attraverso il framework è possibile visualizzare i video visualizzati dagli utenti e determinare se l'esperienza relativa alla visione del filmato è stata soddisfacente, mostrando informazioni quali il link al video effettivamente prelevato, la qualità dell'esperienza utente, il totale dei *bytes* trasferiti, la durata del trasferimento, il *bitrate* richiesto per una visione ottimale del video, il *throughput* effettivo, l'identificativo del video e il link alla pagina da cui è stato visionato il video. Nelle figure 5.9 5.10 5.11 vengono mostrati degli esempi di come possono essere visualizzati i dati forniti dal framework, nella prima l'esperienza utente risulta soddisfacente, al contrario nella seconda e nella terza l'esperienza risulta soddisfacente anche se vi è un *bitrate* richiesto maggiore; gli indirizzi ip non sono stati specificati in quanto resi anonimi.

Dominio	Mbyte	Durata		
facebook.com	1,509.405	24:01:10	Dettaglio Minuti di Navigazione	Dettaglio Tempo di Risposta
vodafone.it	1,277.46	24:20:17	Dettaglio Minuti di Navigazione	Dettaglio Tempo di Risposta
rai.it	767.105	12:11:29	Dettaglio Minuti di Navigazione	Dettaglio Tempo di Risposta
ytimg.com	699.332	11:43:52	Dettaglio Minuti di Navigazione	Dettaglio Tempo di Risposta
google.it	637.328	12:05:35	Dettaglio Minuti di Navigazione	Dettaglio Tempo di Risposta
symantecliveupdate.com	616.28	12:24:06	Dettaglio Minuti di Navigazione	Dettaglio Tempo di Risposta
mediaset.it	393.865	13:07:55	Dettaglio Minuti di Navigazione	Dettaglio Tempo di Risposta
youtube.com	388.833	11:53:37	Dettaglio Minuti di Navigazione	Dettaglio Tempo di Risposta
google.com	373.999	24:20:19	Dettaglio Minuti di Navigazione	Dettaglio Tempo di Risposta
serving-sys.com	295.437	5:02:22	Dettaglio Minuti di Navigazione	Dettaglio Tempo di Risposta
libero.it	288.954	12:03:29	Dettaglio Minuti di Navigazione	Dettaglio Tempo di Risposta
apple.com	261.892	18:42:43	Dettaglio Minuti di Navigazione	Dettaglio Tempo di Risposta

Fig. 5.6: Visualizzazione del traffico e del tempo di navigazione dei domini visitati dagli utenti.

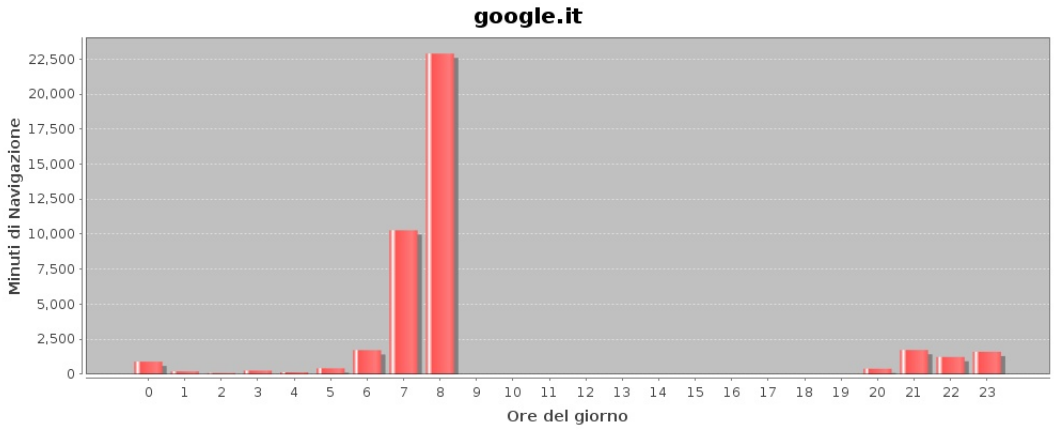


Fig. 5.7: Visualizzazione dei minuti di navigazione verso un determinato dominio durante il giorno, nel caso spefico *google.it*.

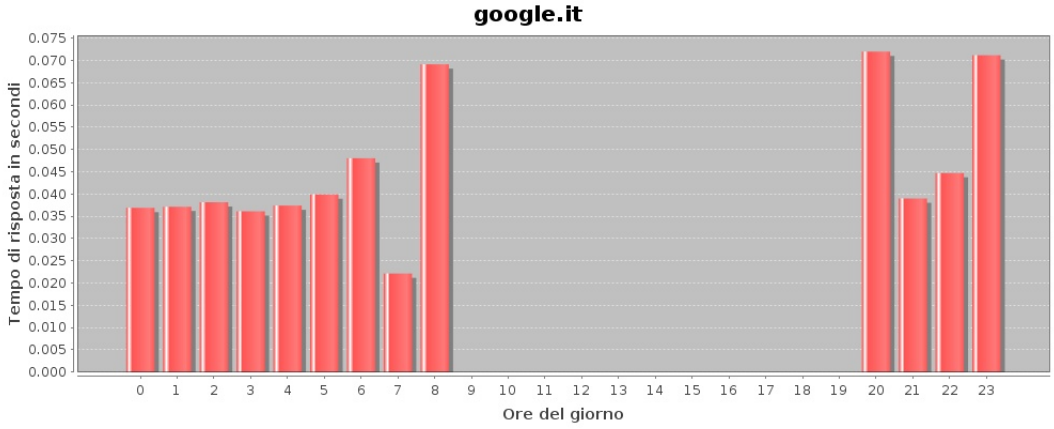


Fig. 5.8: Visualizzazione del tempo di risposta verso un determinato dominio durante il giorno, nel caso spefico *google.it*.

Client	----------
User Agent	Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6_6; it-it) AppleWebKit/533.20.25 (KHTML, like Gecko) Version/5.0.4 Safari/533.20.27
Link	v2.cache3.c.youtube.com/videoplayback?sparams=id%2Cexpire%2Cip%2Cipbits%2Citag%2Calgorithm%2Cburst%2Cfactor%2Coc%3AU0hPRVdRUF9FSkNOOV9PS1RJ&fexp=903802&algorithm=throttle-factor&itag=34&ipbits=0&burst=40&sver=3&signature=78565B493CE4A9C23B9409F6D7CE49AE040E4BA0.0C13ED4BAAAB92FA5A3EFA6174272E40C74A6DD1&expire=1300762800&key=yt1&ip=0.0.0.0&factor=1.25&id=4a00793862509409&redirect_counter=1
Data di visualizzazione	2011-03-21 21:54:22
Qualità	Soddisfacente
Bytes	517673 bytes
Durata Sessione (hh:mm:ss)	00:00:04.259
Bitrate richiesto	0.5 Mbit/sec
Throughput	0.9391875888436677 Mbit/sec
Identificativo del video	4a00793862509409
Referer	www.youtube.com/watch?v=SgB5OGJQlAk&feature=related

Fig. 5.9: Visualizzazione delle informazioni relative alla visione di filmati attraverso il servizio offerto da *YouTube* (Qualità video: 360p, Esperienza utente: soddisfacente).

Client	----------
User Agent	Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6_6; it-it) AppleWebKit/533.20.25 (KHTML, like Gecko) Version/5.0.4 Safari/533.20.27
Link	v5.lscache5.c.youtube.com/videoplayback?sparams=id%2Cexpire%2Cip%2Cipbits%2Citag%2Calgorithm%2Cburst%2Cfactor%2Coc%3AU0hPRVdRVF9FSkNOOV9PS1hF&fexp=903802&algorithm=throttle-factor&itag=34&ipbits=0&burst=40&sver=3&signature=6E60DFD88283E1EC742ECD0F9B33A36581372C85.2EF05E4343D4CAAEC3A648D3936BEDBEDFD81CB4&expire=1300766400&key=yt1&ip=0.0.0.0&factor=1.25&id=c3359d0c2b42d483
Data di visualizzazione	2011-03-21 22:07:26
Qualità	Insoddisfacente
Bytes	1065834 bytes
Durata Sessione (hh:mm:ss)	00:00:29.585
Bitrate richiesto	0.5 Mbit/sec
Throughput	0.2704073006484631 Mbit/sec
Identificativo del video	c3359d0c2b42d483
Referer	www.youtube.com/watch?v=wzWdDCtC1IM&feature=mfu_in_order&list=UL

Fig. 5.10: Visualizzazione delle informazioni relative alla visione di un filmato attraverso il servizio offerto da *YouTube* (Qualità video: 360p, Esperienza utente: insoddisfacente).

Client	----------
User Agent	Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6_6; it-it) AppleWebKit/533.19.4 (KHTML, like Gecko) Version/5.0.3 Safari/533.19.4
Link	v14.lscache6.c.youtube.com/videoplayback?sparams=id%2Cexpire%2Cip%2Cipbits%2Citag%2Calgorithm%2Cburst%2Cfactor%2Coc%3AU0hPRVhLVF9FSkNOOV9QRVhB&fexp=910300%2C904538&algorithm=throttle-factor&itag=35&ipbits=0&burst=40&sver=3&signature=2EB8F425C90690DE290E8A9CE90096428C201825.54267C5DF514B7171F1E6BC54281912504874167&expire=1300806000&key=yt1&ip=0.0.0.0&factor=1.25&id=dacef49af6e97266&begin=31533&ptk=Machinima
Data di visualizzazione	2011-03-22 09:13:34
Qualità	Soddisfacente
Bytes	2678862 bytes
Durata Sessione (hh:mm:ss)	00:00:09.848
Bitrate richiesto	0.8 Mbit/sec
Throughput	2.132412661885803 Mbit/sec
Identificativo del video	dacef49af6e97266
Referer	www.youtube.com/watch?v=2s70mvbpcmY

Fig. 5.11: Visualizzazione delle informazioni relative alla visione di un filmato attraverso il servizio offerto da *YouTube*, (Qualità video: 480p, Esperienza utente: soddisfacente).

5.3 Confronto con lo Stato dell'Arte

Il framework proposto implementa la metodologia esposta nel capitolo 3, la quale si propone di analizzare dei servizi di rete. In particolare in questo lavoro di tesi ci si è focalizzati su servizi basati sul protocollo HTTP. L'obiettivo è quello di analizzare il traffico di rete, in modo tale da ottenere delle metriche che illustrino il comportamento di un utente per quanto riguarda l'utilizzo di servizi di rete su HTTP, dando una stima della qualità dei servizi di rete studiati percepita dagli utenti.

Il framework si colloca in un'architettura in cui da una parte vi sono una o più sonde che effettuano monitoraggio di rete passivo, catturando i pacchetti che transitano sulla rete monitorata, e fornendo dei log di traffico, in cui vi sono tutte le richieste HTTP effettuate dai vari host della rete. Questi log sono i dati di ingresso al framework, il quale li interpreta e li elabora, estraendo delle metriche relative all'analisi del traffico HTTP e alla qualità dell'esperienza utente di alcuni specifici servizi di rete. Le informazioni computate dal framework possono essere utilizzate da un client software, in modo tale da mostrarle agli utenti.

Per comparare il framework con lo stato dell'arte, viene riproposta la tabella 2.1, in cui vi sono elencate le caratteristiche di alcuni framework.

La metodologia proposta a differenza dello stato dell'arte citato in 2.3, presenta le seguenti caratteristiche:

- Viene utilizzato il monitoraggio passivo del traffico di rete, quindi a differenza di un monitoraggio attivo non si ha un incremento del traffico di rete e non si crea del traffico artificiale.

- Le metriche estratte sono ad alto livello, quindi comprensibili anche a chi non è un esperto nel campo del monitoraggio di rete, e risulta di facile comprensione anche per quest'ultimi.

	`Il framework disegnato`	**OneClick**	**HostView**	**EtE**	**eQoS**	**sMonitor**
Servizio Specifico	Attualmente analizza il servizio web e quello di *Streaming on demand*, ma vi è la possibilità di estenderlo, aggiungendo altri servizi di rete.	Sì (Messaggistica Istantanea e Giochi Online)	No	Sì (Web)	Sì (Web)	Sì (HTTPS)
Accesso Macchina Utente	No	No	Sì	No	No	No
Richiesta di feedback	No	Sì	Sì	No	No	No
Monitoraggio	Passivo (`nProbe`)	Attivo o Passivo	Passivo (`libpcap`)	Passivo (`tcpdump`)	Passivo (nel kernel)	Passivo (`libpcap`)
HTTPS	Parzialmente, vi la possibilità di tracciare solo il campo *Host*.	No	No	No	No	Sì
Metriche principali	Classificazione dei servizi di rete su HTTP, Aggregazione e correlazione richieste web, qualità esperienza utente per il servizio web e servizio di *Streaming on Demand*, evoluzione del traffico nel tempo.	Qualità audio e video in relazione alle condizioni di rete.	Relative al traffico di rete, prestazioni del sistema	Tempo di risposta di pagine web percepito dall'utente, utilizzo cache e numero di *Abort* di una pagina web.	Tempo di risposta di pagine web percipito dall'utente	Tempo di risposta di pagine web percepito dall'utente per servizi su HTTPS

Tabella 5.1: Confronto frameworks analizzati

- L'elaborazione del traffico può essere effettuata *offline*, quindi non vi è la necessità di operare in *real-time*.

- Attraverso la metodologia studiata è possibile effettuare degli studi ad alto livello della qualità dei servizi, e nel caso di degrado di quest'ultima si può ricerca l'eventuale problematica a livello di rete, seguendo un approccio *top-down*, offrendo un monitoraggio di rete orientato ai servizi.

- La metodologia studiata si basa su servizi basati su HTTP, ma può essere estesa a qualsiasi altro servizio di rete, implementando gli opportuni *plugin* nella sonda e i moduli aggiuntivi al framework.

- Non si ha la necessità di installare alcun software sulle macchine degli utenti per l'analisi della qualità del servizio e non vengono richiesti dei *feedback* ad essi.

Capitolo 6

Conclusioni

Data la costante crescita di servizi di rete eterogenei, si ha la necessità di monitorarli fornendo delle metriche applicative, in quanto gli attuali strumenti per il monitoraggio offrono dei risultati orientati a metriche di rete, i quali sono di difficile interpretazione per coloro che non operano nel settore. Da queste considerazioni nasce questo lavoro di tesi, il quale propone una metodologia per l'analisi dei servizi di rete, in grado di fornire delle metriche applicative, interpretabili dagli utenti. Dato in ingresso il traffico di rete, questo viene correlato, aggregato e caratterizzato. Il risultato viene poi analizzato estraendo delle metriche relative ai servizi studiati, in termini di qualità del servizio e di qualità dell'esperienza utente. Si possono così ottenere delle informazioni relative alle attività svolte dagli utenti.

Nello specifico ci si è focalizzati sul servizio web e sul servizio di *streaming on demand* offerto da YouTube. La metodologia è stata validata attraverso il disegno e l'implementazione di un framework, il quale preso il traffico catturato da una sonda, elabora i dati e restituisce le informazioni riguardanti i servizi studiati. Per effettuare la validazione è stato utilizzato il traffico di rete, fornito da un *ISP* di media grandezza, il quale è stato opportunamente reso anonimo. Dai risultati ottenuti dall'elaborazione del framework, si sono potute evidenziare le attività svolte nel tempo da parte di

uno specifico utente, l'eventuale degrado durante l'utilizzo del servizio web e del servizio di *streaming*. Sono state ottenute le informazioni relative ai dati complessivi di navigazione, come i domini in cui vi sono stati più minuti di navigazione, quelli in cui sono stati trasferiti più dati, l'evolversi del traffico verso un determinato dominio durante il giorno, come varia il tempo di risposta nel giorno. Attraverso queste informazioni si ha quindi una panoramica di come viene utilizzato il servizio web e il servizio di *streaming on demand* da una parte per utenti specifici, dall'altra si ottiene una visione globale di come evolve il traffico di rete complessivamente. Queste possono essere utilizzate dagli utenti stessi per poter visualizzare in modo chiaro e immediato quali richieste sono state effettuate dalla propria macchina o rete, oppure dal fornitore del servizio per evidenziare eventuali degradi di qualità e quindi analizzare il traffico di rete a basso livello per risalire all'esatta causa dell'eventuale problematica, operando con un approccio orientato al servizio, di tipo *top-down*.

6.1 Lavori futuri

La metodologia proposta può sicuramente essere estesa su alcuni aspetti, da una parte sarebbe possibile monitorare altri servizi oltre a quelli già analizzati, dall'altra potrebbe essere necessario un raffinamento nella creazione delle metriche o l'aggiunta di esse. Di seguito verranno esposti alcuni possibili futuri sviluppi.

- Come anticipato è possibile aggiungere dei servizi di rete, oltre a quelli studiati, come ad esempio il servizio di *Instant Messaging*, i giochi online (*multiplayer*), il servizio di posta elettronica. Ad esempio per quest'ultimo studiare come l'invio di una singola email verso molti destinatari, generi un aumento di traffico; ricostruire le attività svolte

dall'utente che utilizza il servizio, per scopi di sicurezza delle rete ad esempio, o per fini statistici.

- Attualmente non viene studiato il caso di HTTPS, potrebbe essere ragionevole aggiungere questa caratteristica, in quanto il numero di applicazioni web che operano attraverso questo protocollo è molto ampio.

- È possibile aggiungere altre metriche in base alle esigenze dell'utilizzatore finale del framework, come suddividere le richieste tra determinati *user agent*, per poter avere ad esempio delle informazioni statistiche relative all'utilizzo di apparati *mobile*.

- Si potrebbe estendere la metodologia per poter monitorare servizi web specifici. Si pensi allo studio di un determinato servizio web, come ad esempio quelli offerti dai popolari *Facebook* o *Twitter*, potrebbe essere interessante ricostruire le attività svolte dagli utenti nel tempo, evidenziando quali richieste web corrispondono a determinate azioni, mostrare se vi sono dei degradi di qualità su determinate operazioni svolte, quali impatti hanno sulla rete.

- Sarebbe possibile aggiungere un secondo livello di aggregazione, si pensi ad esempio ad una richiesta di un video verso il portale *YouTube*, questo genererà molteplici richieste, tra cui alcune verso server come `ytimg.com`, il quale si occupa di restituire le anteprime delle immagini dei video di *YouTube*. Avendo questo genere di informazioni, è possibile aggregare il traffico verso il server *ytimg.com* con quello di *YouTube*, raffinando così l'aggregazione.

Ringraziamenti

Innazitutto vorrei ringraziare il mio relatore Prof. Luca Deri, il quale mi ha seguito durante questo lavoro, dandomi preziosi consigli e punti di vista differenti su come affrontare alcuni problemi di rete.

Vorrei ringraziare tutti gli amici incontrati qui a Pisa, i quali mi hanno accompagnato dal mio arrivo fino adesso, regalandomi momenti di felicità.

Un grazie anche agli amici di sempre, che mi hanno fatto svagare nei week end di ritorno a casa.

Un grazie speciale a Melissa, Felice e Gianluigi, per avermi sopportato e aiutato nei momenti più critici.

Infine grazie ai miei genitori, senza il loro costante aiuto e sostegno non sarei arrivato fino a qui. Grazie!

Bibliografia

[1] Compuware Corporation. Vantage for end-user experience monitoring. http://www.compuware.com/solutions/vantage-for-end-user-experience-monitoring.asp, 2010. [Online; accessed 27-settembre-2010].

[2] L.P Hewlett-Packard Development Company. Hp real user monitor software. https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?zn=bto&cp=1-11-15-25^1438_4000_100__, 2010. [Online; accessed 27-settembre-2010].

[3] Empirix. Assuring qoe on next generation networks - white paper, August 2003.

[4] International Telecommunication Union. P.10/g.100 (2006) amendment 1 (01/07): New appendix i - definition of quality of experience (qoe), 2007.

[5] I. Martínez-Yelmo, I. Seoane, and C. Guerrero. Fair Quality of Experience (QoE) Measurements Related with Networking Technologies. *Wired/Wireless Internet Communications*, pages 228–239, 2010.

[6] A. Van Moorsel. Metrics for the internet age: Quality of experience and quality of business. In *Fifth International Workshop on Performability Modeling of Computer and Communication Systems, Arbeitsberichte*

des Instituts für Informatik, Universität Erlangen-Nürnberg, Germany, volume 34, pages 26–31. Citeseer, 2001.

[7] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian. Internet inter-domain traffic. *ACM SIGCOMM Computer Communication Review*, 40(4):75–86, 2010.

[8] G. Maier, A. Feldmann, V. Paxson, and M. Allman. On dominant characteristics of residential broadband internet traffic. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 90–102. ACM, 2009.

[9] H. Schulze and K. Mochalski. Internet study 2008/2009. `http://www.ipoque.com/study/ipoque-Internet-Study-08-09.pdf`. [Online; accessed 27-settembre-2010].

[10] Sandvine Incorporated. 2009 global broadband phenomena. `http://www.sandvine.com/downloads/documents/2009%20Global%20Broadband%20Phenomena%20-%20Executive%20Summary.pdf`. [Online; accessed 27-settembre-2010].

[11] B. Ager, F. Schneider, J. Kim, and A. Feldmann. Revisiting Cacheability in Times of User Generated Content.

[12] Etsi ts 102 232-2:lawful interception (li); handover interface and service-specific details (ssd) for ip delivery; part 2: Service-specific details for e-mail services, 2010.

[13] A. Gupta and R. Sekar. An approach for detecting self-propagating email using anomaly detection. In *Recent Advances in Intrusion Detection*, pages 55–72. Springer, 2003.

[14] S. Khirman and P. Henriksen. Relationship Between Quality-of-Service and Quality-of-Experience for Public Internet Service. In *In Proc. of the 3rd Workshop on Passive and Active Measurement*.

[15] C.H. Muntean. Improving learner quality of experience by content adaptation based on network conditions. *Computers in Human Behavior*, 24(4):1452–1472, 2008.

[16] N. Melnikov and J. Schönwälder. Cybermetrics: User Identification through Network Flow Analysis. *Mechanisms for Autonomous Management of Networks and Services*, pages 167–170, 2010.

[17] F. Schneider, A. Feldmann, B. Krishnamurthy, and W. Willinger. Understanding online social network usage from a network perspective. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 35–48. ACM, 2009.

[18] B. Mobasher, H. Dai, T. Luo, and M. Nakagawa. Discovery and evaluation of aggregate usage profiles for web personalization. *Data Mining and Knowledge Discovery*, 6(1):61–82, 2002.

[19] M. Zink, K. Suh, Y. Gu, and J. Kurose. Watch global, cache local: YouTube network traffic at a campus network-measurements and implications. *Proceeding of the 15th SPIE/ACM Multimedia Computing and Networking (MMCN'08)*, 2008.

[20] P. Gill, M. Arlitt, Z. Li, and A. Mahanti. Youtube traffic characterization: a view from the edge. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, page 28. ACM, 2007.

[21] Cisco System Inc. `http://www.cisco.com`.

[22] B. Claise. Cisco Systems NetFlow Services Export Version 9. RFC 3954 (Informational), October 2004.

[23] B. Claise, M. Fullmer, P. Calato, and R. Penno. IPFIX protocol specifications. *draftietf-ipfix-protocol-03. txt*, 2004.

[24] P. Phaal, S. Panchen, and N. McKee. InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks. RFC 3176 (Informational), September 2001.

[25] S. Waldbusser. Remote Network Monitoring Management Information Base. RFC 2819 (Standard), May 2000.

[26] K. McCloghrie and M. Rose. Management Information Base for Network Management of TCP/IP-based internets:MIB-II. RFC 1213 (Standard), March 1991. Updated by RFCs 2011, 2012, 2013.

[27] F. Fusco, L. Deri, and J. Gasparakis. Towards Monitoring Programmability in Future Internet: challenges and solutions.

[28] The Tcpdump Group. libpcap. `http://www.tcpdump.org`. [Online; accessed 6-Ottobre-2010].

[29] Computer Networks Group (NetGroup) at Politecnico di Torino. The netbee library. `http://www.nbee.org/`, August 2004. [Online; accessed 7-Ottobre-2010].

[30] A. Orebaugh, G. Ramirez, J. Burke, and L. Pesce. Wireshark & Ethereal Network Protocol Analyzer Toolkit (Jay Beale's Open Source Security). 2006.

[31] L. Deri et al. Improving passive packet capture: beyond device polling. In *Proceedings of SANE*, volume 2004. Citeseer, 2004.

[32] A. Moore and K. Papagiannaki. Toward the Accurate Identification of Network Applications. Boston, USA, March 2005. Passive & Active Measurement Workshop.

[33] V. Paxson. Empirically-derived analytic models of wide-area tcp connections. *IEEE/ACM Trans. Networking*, 2(4):316–336, 1994.

[34] N. Perelman V., Melnikov and J. Schönwälder. Flow signatures of popular applications, 2010.

[35] K.T. Chen, C.C. Tu, and W.C. Xiao. OneClick: A framework for measuring network quality of experience. In *INFOCOM 2009, IEEE*, pages 702–710. IEEE, 2009.

[36] P. Recommendation. 862: Perceptual evaluation of speech quality (PESQ): An objective method for end-to-end speech quality assessment of narrow-band telephone networks and speech codecs. *Feb*, 14:14–0, 2001.

[37] International Telecommunication Union. Objective perceptual video quality measurement techniques for digital cable television in the presence of a full reference. ITU-T Recommendation J.144, 2004.

[38] D. Joumblatt, R. Teixeira, J. Chandrashekar, and N. Taft. HostView: Annotating end-host performance measurements with user feedback. June 2010.

[39] F. Gringoli, L. Salgarelli, M. Dusi, N. Cascarano, and F. Risso. Gt: picking up the truth from the ground for internet traffic. *ACM SIGCOMM Computer Communication Review*, 39(5):12–18, 2009.

[40] Y. Fu, L. Cherkasova, W. Tang, and A. Vahdat. EtE: Passive end-to-end internet service performance monitoring. In *Proc. USENIX Annual Technical Conference*, pages 115–130, 2002.

[41] The Tcpdump Group. tcpdump. `http://www.tcpdump.org`. [Online; accessed 5-Ottobre-2010].

[42] A. Feldmann. BLT: Bi-Layer Tracing of HTTP and TCP/IP1. *Computer networks*, 33(1-6):321–335, 2000.

[43] J. Wei and C.Z. Xu. eQoS: Provisioning of client-perceived end-to-end qos guarantees in web servers. *IEEE Transactions on Computers*, pages 1543–1556, 2006.

[44] D.P. Olshefski, J. Nieh, and E. Nahum. ksniffer: Determining the remote client perceived response time from live packet streams. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation-Volume 6*, page 23. USENIX Association, 2004.

[45] Jianbin Wei and Cheng zhong Xu. smonitor: A non-intrusive client-perceived end-to-end performance monitor for secured internet services. usenix. In *In Proceedings of USENIX Annual Technical Conference*, 2006.

[46] W. Li, A.W. Moore, and M. Canini. Classifying HTTP traffic in the new age. In *ACM SIGCOMM*, 2008.

[47] J. Postel. Transmission Control Protocol. RFC 793 (Standard), September 1981. Updated by RFCs 1122, 3168.

[48] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Updated by RFCs 2817, 5785.

[49] E. Rescorla. HTTP Over TLS. RFC 2818 (Informational), May 2000. Updated by RFC 5785.

[50] J. Young and S. Smith. Akamai and JupiterResearch Identify'4 Seconds' as the New Threshold of Acceptability for Retail

Web Page Response Times. *Disponible sur http://www. akamai. com/html/about/press/releases/2006/press_110606. html*.

[51] D. Rossi, M. Mellia, and C. Casetti. User patience and the web: a hands-on investigation. In *Global Telecommunications Conference, 2003. GLOBECOM'03. IEEE*, volume 7, pages 4163–4168. IEEE, 2004.

[52] YouTube. `http://www.youtube.com`.

[53] Patrick McFarland. Approximate youtube bitrates. `http://adterrasperaspera.com/blog/2010/05/24/approximate-youtube-bitrates`, 2010. [Online; accessed 19-febbraio-2011].

[54] Bigger and better: Encoding for youtube 720p hd. `http://webvideotechniques.com/123/bigger-and-better-encoding-for-youtube-hd`, 2008. [Online; accessed 19-febbraio-2011].

[55] Trevor Greenfield. Youtube's 1080p – failure depends on how you look at it. `http://trevorgreenfield.com/rants-and-raves/youtubes-1080p-failure-depends-on-how-you-look-at-it`, 2009. [Online; accessed 19-febbraio-2011].

[56] Billy Biggs. 1080p hd is coming to youtube. `http://youtube-global.blogspot.com/2009/11/1080p-hd-comes-to-youtube.html`, 2009. [Online; accessed 19-febbraio-2011].

[57] L. Deri and N.E.T. SpA. nProbe: an Open Source NetFlow Probe for Gigabit Networks. In *TERENA Networking Conference*. Citeseer, 2003.

[58] ntop.org. `http://www.ntop.org/nProbe.html`.

[59] Oracle Corporation. JavaSE. `http://download.oracle.com/javase/6/docs/index.html`. [Online; accessed 20-Marzo-2011].

[60] D. Richard Hipp. sqlite3. `http://http://www.sqlite.org/`. [Online; accessed 10-Marzo-2011].

[61] D. Gilbert and T. Morgner. JFreeChart. *Availab le at http://www. jfree. org/jfreechart/index. html*.