UNIVERSITÀ
DEGLI STUDI
DI PADOVA

DIPARTIMENTO
DI INGEGNERIA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA IN INGEGNERIA INFORMATICA

SVILUPPO DI UNO SPIDER SPERIMENTALE PER LA RACCOLTA DI DATI STATISTICI SUI WEB SERVER ATTIVI

Relatore:

Prof. Nicola Zingirian

Laureando:

Riccardo Benetti

ANNO ACCADEMICO 2021 – 2022

Data di laurea: 25 Novembre 2022

Abstract

Il lavoro di tesi ha avuto lo scopo di ricavare delle statistiche sui server Web oggi utilizzati e sulle loro configurazioni. Per ottenere questa statistica è stato sviluppato un client automatizzato (detto anche “spider”) in grado di visitare un elevato numero di server, generando indirizzi IP casuali e cercando di connettersi ai rispettivi server mediante protocollo HTTP 1.1. Quando lo spider ottiene una risposta dal web server ne memorizza tutti gli Header, e poi ricava tutti i link presenti nell'Entity-Body; ripetendo questo processo fino ad accumulare un numero ritenuto sufficiente di indirizzi IP. Al termine dell’esecuzione le informazioni ricavate vengono processate e sono qui riportate, insieme alla documentazione dei vari header analizzati, con riflessioni sullo stato della rete e analisi dei risultati ottenuti.

Capitolo 1

Introduzione

1.1 Web degli Standard & Web della Realtà

Gli standard a supporto del Web, ed in particolare il protocollo HTTP, sono stati negli ultimi decenni oggetto di continue evoluzioni volte al perfezionamento dell'efficienza e della sicurezza. L'indagine svolta nella presente tesi, di natura puramente sperimentale, è quella di caratterizzare tramite l'analisi delle risposte Web di server selezionati con un criterio completamente casuale, e tramite le successive statistiche, quali meccanismi sono recepiti sui server che si espongono ad indirizzi Internet pubblici, e con quale grado di dispiegamento.

I dati raccolti e le percentuali sono calcolate su una base di 10.000 indirizzi IP unici appartenenti a web server che hanno risposto a una richiesta HTTP/1.1 generata dal programma, descritto più in basso.

La tesi, dopo aver illustrato il metodo di raccolta delle informazioni, propone i risultati dell'analisi dei dati relativi al numero di siti con reindirizzamento ad HTTPS, e quindi dotato di Transport Layer Security (TLS), la presenza dei CDN nella rete e sull'utilizzo della memoria cache per le pagine Web.

Capitolo 2

Metodologia

2.1 Metodologia di Scanning

Il programma sviluppato genera indirizzi IP casuali alla ricerca di una web server che risponda, una volta ricevuta una risposta, passa in rassegna il contenuto ricevuto alla ricerca di collegamenti validi, visitandoli e ripetendo il processo con le nuove pagine, una volta esauriti tutti i collegamenti possibili, ricomincia la ricerca casuale di indirizzi.
Si nota che il 12.35% degli indirizzi sono frutto di visite guidate dai collegamenti presenti all'interno delle pagine web, mentre l'87.65% degli indirizzi sono stati raggiunti attraverso il processo di estrazione casuale.

2.2 Richiesta & Risposta HTTP

Nel protocollo HTTP lo scambio di informazioni tra client e server avviene utilizzando un modello in cui il client effettua una richiesta e il server restituisce una risposta.
La richiesta è il modo in cui il client si mette a contatto col server, indicando il metodo, l'URL della risorsa cercata e la versione del protocollo; possono essere incluse informazioni come la lingua, il tipo di file accettati o altri headers che possono essere utili al server.
La risposta del server è la parte che viene studiata nell'elaborato e la fonte dei dati analizzati, sono le informazioni inviate al client a seguito di una richiesta. Una risposta è composta da "Entity-Headers" e "Entity-Body", ovvero rispettivamente "Intestazione" e "Corpo", la prima contiene una serie di informazioni utili inviate al client per la gestione e la lettura della risorsa, accompagnate da un codice numerico che indica il risultato della richiesta, accoppiato a uno testuale che indica lo stato della risposta; mentre l'Entity-Body non è altro che la risorsa che abbiamo richiesto.
I dati raccolti ed analizzati nell'elaborato derivano unicamente dagli Entity-Headers ricevuti dai server che hanno risposto ad una richiesta "GET / HTTP/1.1".

2.3 Il Programma

Lo spider sviluppato è stato scritto in linguaggio C; le informazioni grezze raccolte, relative agli header, sono salvate in "append" in un file di testo che viene progressivamente aggiornato. Il tutto è gestito in cicli, dove ogni ciclo si occupa di una singola richiesta, in caso di fallimento in uno qualsiasi dei passaggi, il ciclo ricomincerà da capo con un nuovo indirizzo, proseguendo l'esecuzione del programma; inoltre, per velocizzare il processo di ricerca di indirizzi validi, vengono lanciate dieci istanze parallele del programma, che aggiungono informazioni allo stesso file.

Il funzionamento è iterativo. Ad ogni iterazione si seguono i passi necessari per effettuare una richiesta da client e leggere la risposta ottenuta: ad ogni ciclo, se la coda dei link da visitare è vuota, viene generato un indirizzo IP casuale di cui si trova il nome, altrimenti viene estratto il primo indirizzo presente nella coda, si crea un socket e si genera la classica request HTTP/1.1, sempre alla pagina home "/" per tutti i server visitati. Di questo processo iniziale la parte degna di nota è la chiamata a una funzione di "connect(...)" personalizzata, con un timeout che ne forzerà l'uscita una volta scaduto; questo è possibile tramite l'utilizzo di socket non bloccanti e la chiamata a sistema "poll()"; grazie a questa funzione, cercare di connettersi a indirizzi IP non associati a server non comporta perdite di tempo, rendendo possibile la realizzazione dello spider.

In caso si riceva una risposta, gli header vengono immediatamente salvati nel file, formattati per facilitarne la lettura in seguito; mentre il corpo viene passato in rassegna alla ricerca di link a cui connettersi; se se ne trovano, vengono inseriti in una coda e saranno visitati nelle prossime iterazioni, come gli eventuali link che le loro pagine home contengono, visitando tutto l'albero stando attenti a non rivisitare nulla, fino a che non si esaurisce la coda.

Il diagramma di flusso che descrive il funzionamento a grandi linee del programma è illustrato in figura 1.1.

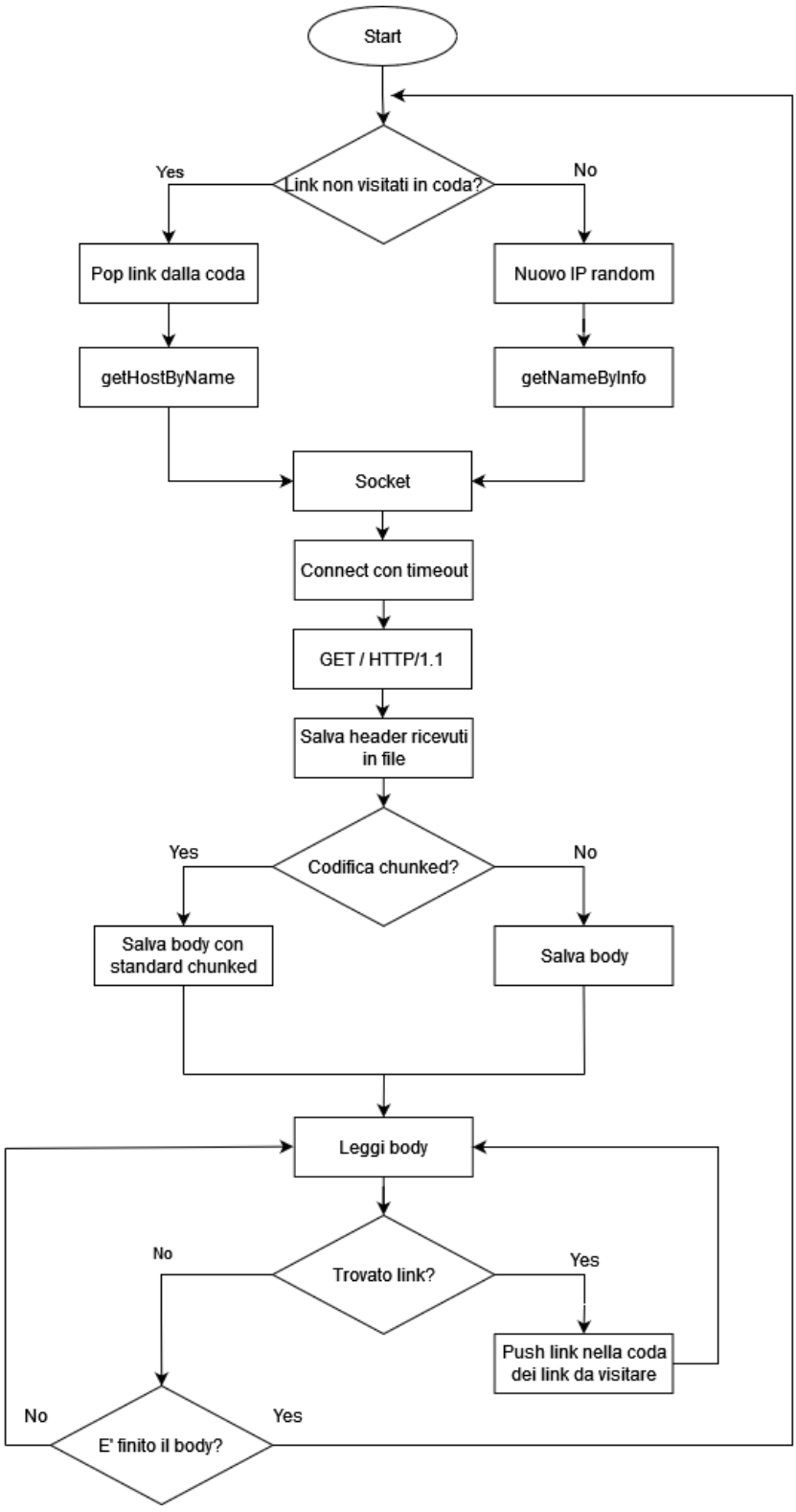

Figura 1.1: diagramma di flusso dello spider sviluppato

Capitolo 3

HTTPS

Il protocollo HTTP media la comunicazione tra client e server in modo non criptato, sfortunatamente non tutti i siti sono innocui e, specialmente in rete, non fidarsi è la scelta migliore. Con questo scopo nasce la versione sicura di HTTP, detta HTTPS in cui le comunicazioni sono crittografate con chiavi generate sul momento, impedendo a terze parti non autorizzate di intercettare qualsiasi dato trasferito.

3.1 Il Protocollo HTTP

HTTP è un protocollo per la condivisione di risorse, diventato la base di qualsiasi scambio di dati nel web come documenti HTML, immagini, video, file e molto altro. HTTP è un protocollo client-server, ovvero, perché ci sia una comunicazione la richiesta deve provenire dal destinatario della risorsa, solitamente il browser. Il client e il server comunicano attraverso uno scambio di messaggi, le richieste e le risposte, per visualizzare una pagina web, un mosaico di varie risorse ottenute attraverso richieste ai rispettivi server.

Lo "User-Agent" è un qualsiasi strumento che agisce per conto dell'utente, questo può essere il browser oppure un programma, ed è sempre colui che genera la richiesta, partendo dal documento HTML che rappresenta la pagina desiderata, una volta ricevuto viene analizzato per generare altre richieste allo scopo di ottenere tutte le sotto-risorse contenute nella pagina, come file CSS, javascript, immagini, video e audio, che vengono composte e presentate come pagina web.

Una delle caratteristiche principali di HTTP è la sua semplicità, sia nel funzionamento che nella comprensione dei messaggi; in più è estremamente estensibile, dando modo di definire la semantica di un nuovo header attraverso un semplice accordo tra client e server.

3.2 HTTP Secure

Uno dei più grandi problemi di HTTP è la sicurezza, proprio per la facilità di interpretazione dei messaggi scambiati, vengono esposti molti dati e informazioni sulla sessione, per questo motivo HTTP Secure introduce un protocollo di criptaggio, chiamato Transport Layer Security (TLS) o Secure Sockets Layer (SSL), che rende criptato il passaggio di dati.

Il meccanismo impiegato per garantire la sicurezza della connessione è chiamato "Crittografia Asimmetrica" (figura 3.1), ampiamente usato in questo contesto, che fa uso di due chiavi distinte: la chiave pubblica e la chiave privata; la prima è disponibile e visibile a tutti coloro che desiderano mettersi in contatto con un determinato server, mentre la seconda è segreta e appartiene al server che vogliamo contattare, utilizzata per decodificare i dati codificati con la chiave pubblica.

Un "TLS handshake" utilizza una chiave pubblica per autenticare l'identità del server e per scambiare dati usati per generare una chiave di sessione, ovvero una chiave ottenuta dalla combinazione della parte pubblica e privata, attraverso appositi algoritmi per lo scambio di chiavi, client e server si accordano per una nuova chiave di sessione ad ogni transazione, in modo da impedire di decifrare la comunicazione anche se si riuscisse ad entrare in possesso di una delle chiavi di sessione precedenti.

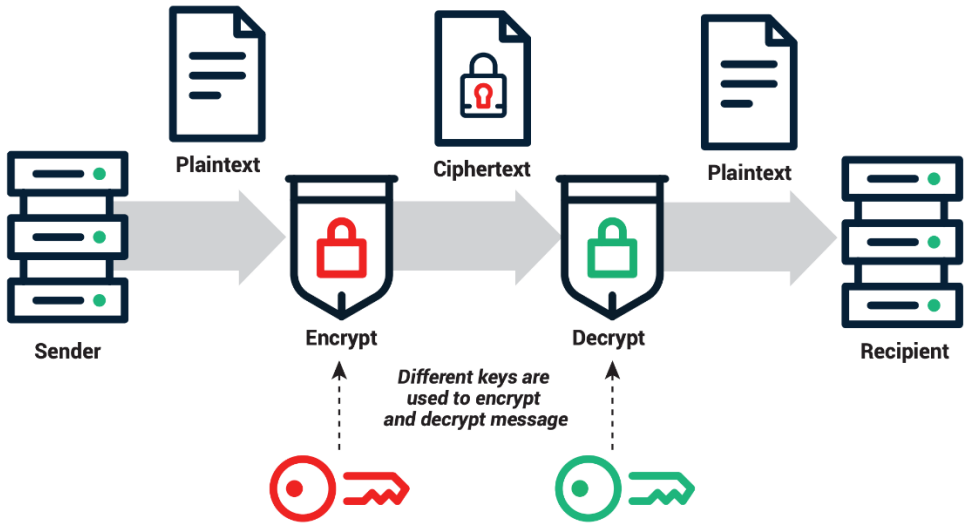

Figura 3.1: schema che rappresenta il funzionamento della crittografia asimmetrica

3.3 Risultati

Al giorno d'oggi, dove la tutela della privacy e la sicurezza in rete sono messi al primo posto, ci si aspetterebbe che una grande percentuale dei siti utilizzi il protocollo HTTPS, dal momento che il funzionamento è uguale alla controparte non sicura, a meno di piccole differenze come la porta utilizzata; invece soltanto il 21.73% delle risposte hanno reinderizzato la domanda a un URL "https://", di cui il 19.05% è stato raggiunto tramite l'estrazione casuale dell'indirizzo.
Sicuramente molti meno di quello che ci piacerebbe pensare, ma c'è da dire che i grandi motori di ricerca, Google in primis, da tanti anni cercano di imporre la transazione al protocollo sicuro a più siti possibile, allo stesso tempo tutelando l'utente in modo che vengano priorizzati siti sicuri come soluzioni ad una ricerca, diminuendo la probabilità di imbattersi di siti rischiosi per la privacy; inoltre, nell'interfaccia del browser viene costantemente indicato se ci troviamo in una pagina sicura o meno, attraverso un simbolo o un messaggio a lato della barra dell'URL.

Capitolo 4

Content Delivery Network

4.1 Server & Proxy

Il protocollo HTTP prevede una comunicazione tra due parti: un client, che richiede una risorsa, e un server, che la fornisce; un server è un sistema informatico che ha lo scopo di rispondere alle richieste provenienti da qualsiasi client necessiti le sue risorse e gestire il traffico di informazioni, rimanendo passivo fino al momento in cui viene interpellato.

Un server può rispondere a più client e un client può contattare più server, identificandoli attraverso il nome dell'host del dominio e l'indirizzo IP della loro componente hardware, su cui possono coesistere più server contemporaneamente.

La comunicazione tra client e server è preceduta da una fase di definizione di un socket, formato da una coppia indirizzo IP e porta, nel caso del server questo socket entrerà in funzione solo a fronte di una richiesta di connessione da un client tramite TCP, mantenendo costante la sua porta.

Quando un server ha la capacità di inoltrare la richiesta ricevuta, emulando a sua volta il comportamento di un client, viene definito "Proxy Server". Fungendo da nodo intermedio, un proxy server riesce a ricevere richieste da un client e a sua volta generarne ad altri server, restituendo le risorse ricevute al client destinatario che le aveva richieste, apparendo invisibile dai punti di vista di entrambe.

I proxy hanno svariate applicazioni in tutta la rete, in caso un client richieda un servizio, come una pagina web o qualsiasi risorsa appartenente ad un altro server, il proxy valuta la richiesta in modo da semplificarla e gestirne la complessità. I proxy sono largamente impiegati nel mondo della sicurezza, essendo server intermediari possono svolgere la funzione di monitoraggio e filtraggio dei contenuti che vi passano attraverso, supportando l'autenticazione degli utenti, estremamente utile per organizzazioni commerciali e non come le scuole, oppure possono rendere anonima la navigazione sul web, nascondendo alcune informazioni del client al server che vogliamo contattare.

4.2 CDN

Ottenere una risposta a una richiesta nel web può sembrare semplice e lineare, involvendo un client che genera la domanda e un server che restituisce la risposta, ma in realtà la risorsa desiderata potrebbe dover passare attraverso numerosi ISPs o centri dati, con svariate bande e da luoghi di tutto il mondo; comportando rischi per la sicurezza, inconsistenze di velocità e accessibilità, tutti problemi attribuiti alla distanza che i dati sono obbligati a percorrere. Per questo motivo i file più pesanti e richiesti di una pagina, come file javascript, immagini e HTML, vengono distribuiti e memorizzati in una rete di server collegati tra loro in tutto il mondo, in modo da coprire più aree geografiche possibile e in questo modo essere più vicini ad ogni utente, tagliando significativamente la strada percorsa dalle informazioni e di conseguenza il tempo necessario al caricamento (figura 4.1).

I CDN sono largamente utilizzati dalle aziende per i molti vantaggi che possiedono, oltre ad avvicinare i contenuti agli utenti, aiutando a diminuire i tempi di caricamento, introducono la ridondanza necessaria per assicurare che la pagina rimanga sempre operativa, fornendo protezione in caso di malfunzionamenti hardware o picchi di traffico, bilanciando il carico uniformemente su diversi server.

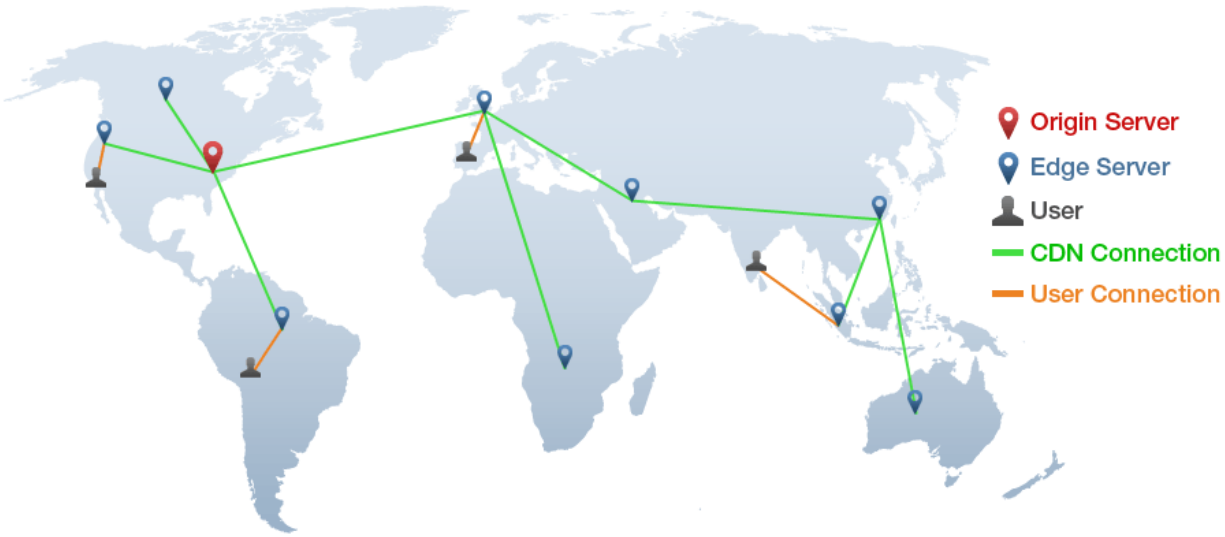

Figura 4.1: Rappresentazione del funzionamento di un CDN

4.3 Risultati

Proprio per il fatto che più la rete di server è fitta e distribuita, migliore sarà l'efficacia del servizio, questa ipotesi è rispecchiata dai dati, in cui si ritrovano alcune delle maggiori aziende che forniscono il servizio di CDN: Akamai, Amazon CloudFront, rispettivamente prima e seconda in termini di taglia, e Cloudflare. Queste tre aziende da sole sono responsabili del 17.88% delle risposte ricevute da indirizzi casuali, che equivale a dire che circa un sesto della rete è composta da questo tipo di servizio.
La tabella 4.1 rappresenta i server con il maggior numero di risposte rilevate, riportate in percentuale e comprendendo anche gli indirizzi visitati attraverso link; la tabella 4.2 rappresenta i server con maggior numero di risposte raggiunti unicamente tramite estrazione casuale.

Server	Indirizzi sul Totale
Nginx	21.75%
Apache	18.29%
AkamaiGHost	14.27%
Microsoft	6.80%
CloudFront	3.33%
Awselb	3.29%
Cloudflare	2.79%

Server	Indirizzi Casuali
Nginx	19.06%
Apache	16.03%
AkamaiGHost	12.51%
Microsoft	5.96%
Cloudfront	2.92%
Awselb	2.89%
CloudFlare	2.45%

Figura 4.1 – 4.2: percentuali di indirizzi associati ai rispettivi server

In conclusione si evince che molti contenuti vengono reperiti attraverso CDN per migliorare l'esperienza degli utenti che visitano i loro siti, poiché meglio un sito funziona, meno saremo

portati ad abbandonarlo. Per questo è logico pensare che generando richieste a server casuali, molto probabilmente saremo intercettati da uno dei tanti CDN che ci fornisce quello di cui abbiamo bisogno; questa struttura, e la vasta applicazione di CDN, implica che il contenuto che visitiamo online ogni giorno è molto frammentato e diffuso in tutto il globo, invece che concentrato in un singolo luogo a cui accediamo.

Capitolo 5

Caching

5.1 La Cache nel Web

Ottenere risorse in rete può essere lento e dispendioso: risposte lunghe possono richiedere più viaggi tra browser e server e ogni richiesta, con conseguente risposta, comporta un consumo di dati, banda e tempo che può essere evitato.
I meccanismi di cache usati da HTTP permettono di rendere più veloce ed efficiente l'accesso a un sito web, memorizzando i file e le risposte del server agli utenti, da riutilizzare o modificare a seconda di determinate condizioni, diverse per ogni meccanismo di caching.
Compatibile con ogni browser, la cache non è altro che una collezione di risorse a cui il browser accede prima di inviare una richiesta, nel tentativo di trovare il file desiderato senza il completo intervento di un server. Nel caso in cui il file venga trovato, sarà caricato direttamente dalla cache, eliminando il costo in dati per scaricarlo e la latenza causata dal server.
Il comportamento della cache è determinato dalla presenza di particolari headers, sia nella richiesta che nella risposta, spesso e volentieri il browser include automaticamente gli headers che gli sono utili nella richiesta, quindi, in questo capitolo, ci concentreremo sull'analisi di quelli contenuti nella risposta, che forniscono indizi sul meccanismo di cache impiegato e informazioni per il suo utilizzo.

5.2 Cache Privata & Condivisa

Esistono due tipi principali di cache: privata e condivisa, il cui comportamento viene determinato dall'header "Cache-Control".
Una cache privata è legata a un utente specifico, dove la risposta memorizzata è personalizzata per quell'utente e non è visibile da altri, scongiurando rischi per la privacy e garantendo che non ci siano perdite di informazioni sensibili.

Una cache condivisa invece memorizza e accede alle risposte di più utenti contemporaneamente, così da avere i benefici dell'utilizzo della cache anche senza avere effettuato l'accesso a una risorsa, si suddivide in proxy cache e managed cache.
Una proxy cache è semplicemente un server intermediario che risponde al posto del server di origine; memorizzando i contenuti direttamente nel proxy si accorcia la catena di connessioni ed è possibile condividerli con più utenti.
Le managed cache sono cache impiegate per ridurre il carico del server di origine, in modo da distribuire contenuti efficientemente, un esempio può essere un CDN o un reverse proxy (proxy che riceve richieste da internet e le inoltra a un web server di una rete interna). Le caratteristiche delle managed cache possono variare, anche se generalmente si riesce a controllarne il comportamento attraverso l'header "Cache-Control".

5.3 Cache-Control

"Cache-Control" è un header che può essere incluso dal server, specifica come e per quanto tempo il browser, e altre eventuali cache intermedie, devono gestire la risposta a cui è allegato, fornendo un insieme di comandi per impostare il comportamento della cache. Non includere "Cache-Control" non disabilita la cache, ma il browser cercherà di indovinare i comandi che possono avere più senso, a seconda del contesto e delle caratteristiche del file richiesto.
"Cache-Control" specifica informazioni e comandi importanti per l'utilizzo della cache, come ad esempio indicare per quanto tempo un file è considerato utilizzabile, se la cache è condivisa o privata o impedire che un file venga memorizzato, personalizzandone il funzionamento ed evitando spreco di risorse.
È importante ricordare che in "Cache-Control" possono essere presenti più comandi e le percentuali riportate indicano la frequenza con cui è stata riscontrata la presenza dei comandi più utilizzati presi singolarmente.

5.4 Last-Modified

"Last-Modified" è un header inviato dal server, indica l'istante, preciso al secondo, in cui il file per cui è incluso è stato modificato l'ultima volta, al fine di determinare se la versione già contenuta in

cache dello stesso file ha subito o meno modifiche dal momento della memorizzazione e quindi determinandone la validità.

Il server invia il file per la prima volta con l'header "Last-Modified" a fronte di una richiesta, il browser lo inserisce in cache con la data dell'ultima modifica inclusa nel valore dell'header; quando c'è bisogno di estrarre il file dalla cache, il browser invia una richiesta includendo l'header "If-Modified-Since", che contiene la data e l'ora precedentemente specificate da "Last-Modified", se la versione memorizzata è la più aggiornata allora il server risponderà con il messaggio "304 Not Modified" e verrà caricato il file in cache senza bisogno che intervenga il server (figura 5.1), altrimenti risponderà con "200 OK" e invierà il file aggiornato con la nuova data e ora di modifica da memorizzare in cache sovrascrivendo il file precedente.

Nel valore di "Last-Modified" si usa una formattazione specifica di data e ora, detta HTTP-date, descritta nell'RFC2616 sezione 14.29 "Last-Modified", dove vengono specificate data e ora con informazioni aggiuntive, includendo, seguendo l'ordine del formato standard: giorno della settimana, numero del giorno, nome del mese, anno, ore, minuti, secondi e nome del fuso orario (Tue, 15 Nov 1994 12:45:26 GMT), introducendo una struttura standard che rende semplice leggere e comparare il valore dell'header.

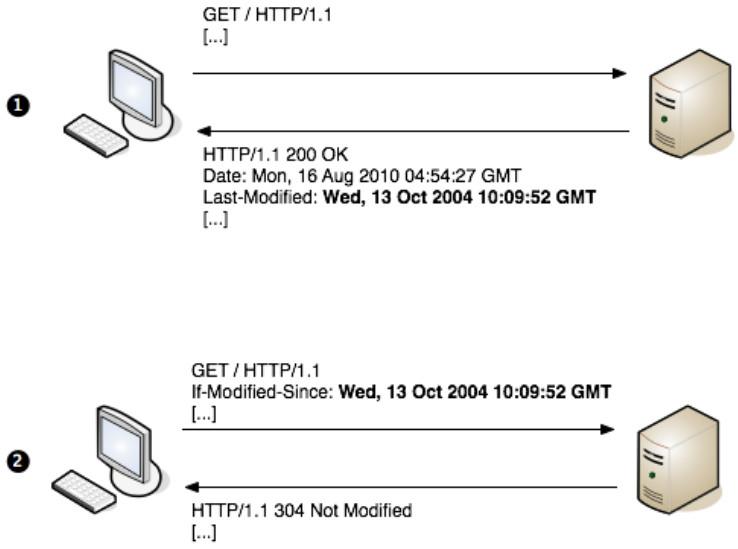

Figura 5.1: Schema di funzionamento dell'header Last-Modified

5.5 ETag

“ETag”, o “entity tag”, è un header inviato dal server il cui funzionamento è molto simile a quello di “Last-Modified”, specifica una chiave che identifica la versione di un file; ogni volta che tale file viene modificato il server genera un nuovo ETag, che viene confrontato con la chiave appartenente allo stesso file contenuto in cache, se le chiavi non coincidono allora vuol dire che è avvenuta una modifica e il file contenuto in cache non è più valido, il server invierà la versione aggiornata con il nuovo ETag, altrimenti verrà caricato il file dalla cache dopo aver ricevuto la risposta “304 Not Modified” (figura 5.2).

La chiave generata da “ETag” è una sequenza di caratteri ASCII posti tra doppi apici, il modo in cui viene generata non è specificato, ma può essere un numero di revisione oppure una traduzione hash del contenuto del file o della data e ora dell’ultima modifica.

Le chiavi possono avere due tipologie di validazione: forte e debole, una validazione forte vuol dire che i file confrontati sono identici in ogni byte, mentre una validazione debole garantisce solo l’uguaglianza semantica, quindi, il file non è esattamente identico all’ultima versione, ma è comunque utilizzabile.

Il funzionamento di “ETag” è molto simile a quello di “Last-Modified”, dal momento che entrambe necessitano di inviare una richiesta al server per accertarsi che il file richiesto, già in cache, sia aggiornato, la più grande differenza tra i due è appunto il modo in cui determinano la validità, rispettivamente, tramite una chiave o tramite data e ora di modifica, questo vuol dire che “Last-Modified” risulta più chiaro e leggibile dal client, ma il concetto che sta alla base è lo stesso.

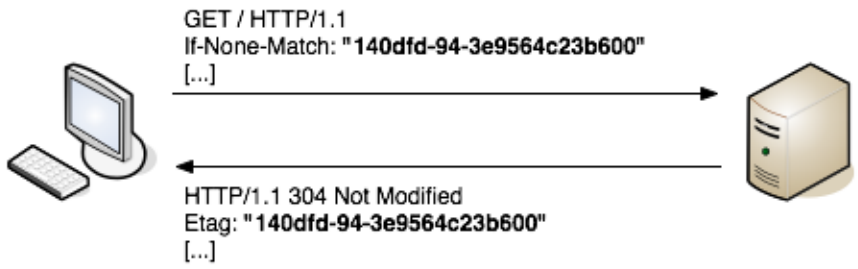

Figura 5.2: schema di funzionamento dell’header ETag

5.6 Expires

“Expires” è un header inviato dal server che contiene l’istante di tempo in cui, il documento a cui si riferisce, diventa inutilizzabile dalla cache e richiederà aggiornamento.
Un meccanismo praticamente equivalente lo si è visto nell’header “Cache-Control”, con il comando “max-age”, che indica il numero di secondi per cui la risorsa rimarrà rilevante prima che necessiti di essere aggiornata attraverso una nuova richiesta; in caso siano presenti entrambe, “max-age” avrà la priorità e il tempo indicato da “Expires” verrà ignorato.
Il tempo in cui la risorsa rimane rilevante può variare a seconda del file a cui fa riferimento, ad esempio ha senso impostare un tempo piuttosto breve per risorse che cambiano in fretta, mentre uno molto lungo per quelle che cambiano di rado o mai, come loghi e immagini; tenendo a mente che il tempo massimo impostabile è di circa un anno, anche se si è notato che valori di tempo troppo grandi possono creare problemi, quindi il browser con ogni probabilità lo rimuoverà ben prima.
Il formato in cui l’header “Expire” comunica il momento di scadenza è tramite HTTP-date, come “Last-Modified” e già citato nel relativo paragrafo, quindi facilmente comparabile e leggibile da un client, descritto nella RFC 7234 sezione 5.3 “Expires”.
Il vantaggio di questo meccanismo di cache è che riduce il numero di richieste effettuate al server, poiché il browser è capace di determinare da solo l’affidabilità di un file, senza dover aspettare una risposta dal server.

5.7 Pragma

“Pragma” è un header utilizzato unicamente in HTTP/1.0 e svolge praticamente la stessa funzione di “Cache-Control”, incluso per motivi di retrocompatibilità di cache HTTP/1.0 che non hanno header “Cache-Control”, anche se non è un così valido rimpiazzo.
Quasi sempre utilizzato con l’attributo “no-cache”, che ha un comportamento analogo al comando di “Cache-Control”, ovvero obbliga il server di origine a verificare la validità del file ad ogni suo utilizzo, pur rimanendo in cache.

5.8 Headers Non Standard

Nei dati raccolti sono presenti altri header che si occupano della gestione della cache, pur essendo decisamente più rari di quelli già descritti.
Generalmente nel nome contengono il prefisso “X-”, che sta ad indicare il fatto che sono header non standard, definiti e impiegati da qualche servizio o azienda su cui il server si appoggia.
Dai dati raccolti sono emersi trentacinque headers non standard diversi, con funzione di gestione della cache, presenti nel 5.58% delle risposte ottenute.

5.9 Risultati & Conclusioni sull’Uso della Cache

I meccanismi di cache analizzati: “Last-Modified”, “ETag” ed “Expires” insieme a “max-age”, hanno fondamentalmente due modi completamente diversi di operare; “Last-Modified” ed “ETag” verificano la validità del documento in cache attraverso una richiesta al server obbligatoria prima di decidere quale risorsa sarà caricata, mentre nel caso di “Expires” e “max-age” è il server che decide per quanto tempo il documento sarà valido e quindi la sua scadenza, togliendo qualsiasi responsabilità al client.

Dai dati raccolti si nota come “Last-Modified” ed “ETag” insieme costituiscano il 50.74% dei metodi di caching utilizzati, senza contare gli indirizzi che non richiedono cache, dimostrando quanto del processo di caching venga messo in mano al client.

Nella tabella 5.3 vengono classificati e comparati i metodi di cache più utilizzati, indicando la percentuale di risposte in cui sono stati rilevati; la sezione “Altro” raggruppa, oltre agli headers non standard, tutte le risposte in cui non era presente alcun header di cache oppure in cui l’header “Cache-Control” non specificava comandi come “no-cache”, “no-store” o “max-age” utili per la gestione della risorsa.

Metodi	Indirizzi sul Totale
Expires	25.36%
Last-Modified	22.16%
ETag	19.40%
No Cache	18.09%
Altro	14.99%

Figura 5.3: Classifica dei metodi di caching più utilizzati con relative percentuali

Comandi Cache-Control	Sul Totale	In Cache-Control
No-cache	11.31%	67.16%
No-store	6.99%	41.51%
Max-age	6.03%	35.81%
Private	4.25%	25.24%
Public	0.55%	3.27%

Figura 5.4: Frequenza di utilizzo dei comandi di Cache-Control più comuni

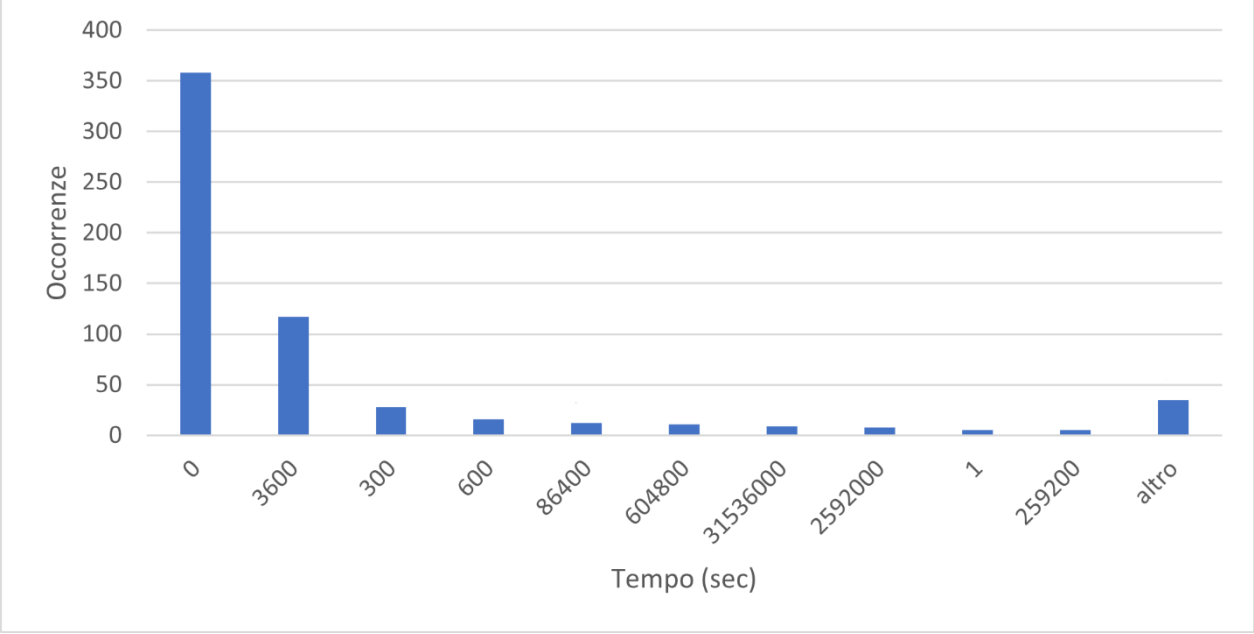

Figura 5.5: numero di occorrenze relative al tempo di permanenza (in secondi) indicate dal comando “max-age” di “Cache-Control”

Conclusioni

L’elaborato ha avuto come scopo l’analisi statistica dei dati raccolti da uno spider web sperimentale che richieda l’accesso a indirizzi IP casuali, dando un’idea della rete completamente libera da browser e motori di ricerca, concentrandosi su tre aspetti principali: la percentuale di indirizzi che implementa il protocollo HTTPS, quanto sono comuni i CDN, quali sono i meccanismi di caching più utilizzati e quanto controllo ha il client sul processo di caching.

Gli indirizzi che implementano il protocollo HTTPS, rendendo sicura la transazione, sono il 19.05%, vale a dire che poco meno di un quinto degli indirizzi casuali garantisce la sicurezza dei dati durante la navigazione, sicuramente meno di quello che ci aspetteremmo.

I server di CDN occupano il 17.88% delle risposte ricevute, che equivale a dire circa un sesto dei server contattati, mostrando quanto diffusi siano questi servizi, questo implica che i dati, oltre a essere ridondanti, diventano frammentati e diffusi in tutto il globo.

Il meccanismo di caching più utilizzato è quello di “Expires”, seguito da “Last-Modified” ed “ETag”, ma insieme gli ultimi due sono presenti nel 41.56% delle risposte e compongono il 50.74% dei metodi di caching utilizzati, mostrando che il processo di caching è spesso lasciato in mano al client, in modo da avere più controllo sulla versione della risorsa a discapito del traffico verso il server.

Collegamenti

Protocollo HTTPS

https://www.cloudflare.com/it-it/learning/ssl/what-is-https/

https://www.soluzioneinformatica.it/2018/12/https/

https://www.cloudflare.com/it-it/learning/ssl/how-does-public-key-encryption-work/

https://www.ionos.it/digitalguide/hosting/tecniche-hosting/cose-https/

Content Delivey Network

https://www.cloudflare.com/it-it/learning/cdn/what-is-a-cdn/

https://www.freecodecamp.org/news/an-introduction-to-the-akamai-content-delivery-network-806aa16d8781/

Caching

https://developer.mozilla.org/en-US/docs/Web/HTTP/Caching

https://web.dev/http-cache/

Cache-Control

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Cache-Control

Last-Modified

https://www.holisticseo.digital/pagespeed/last-modified/

ETag

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/ETag

https://www.holisticseo.digital/pagespeed/etag/

Expires

https://www.geeksforgeeks.org/http-headers-expires/

https://www.holisticseo.digital/pagespeed/expires/

https://gtmetrix.com/add-expires-headers.html

Pragma

https://www.geeksforgeeks.org/http-headers-pragma/

https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Pragma