Università degli Studi di Pisa

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea in Scienze dell'Informazione

Transparent Web Caching:

valutazione della tecnologia in un ambiente scientifico

Candidata

Loredana Martusciello

Relatori

Prof. Giuseppe Attardi

Dott. Marina Buzzi

Controrelatore

Prof. Roberto Grossi

Anno Accademico 1999-2000

Indice

Indice delle figure

1. Introduzione

Nato al CERN di Ginevra come strumento per la distribuzione di file multimediali (dati, immagini, foto, schemi, etc.) nell'ambiente dei fisici delle alte energie, il *World Wide Web* (WWW), in brevissimo tempo, ha letteralmente "rivoluzionato" il mondo dell'informatica (e non solo), avvicinando milioni di utenti al mondo delle reti di computer, con forti ripercussioni economiche, sociali e culturali [35].

La semplicità del WWW, la "ragnatela mondiale di informazioni", con le sue interfacce grafiche user-friendly e il semplice meccanismo dei *link* (collegamenti ipertestuali), ha consentito, anche ad utenti senza conoscenze tecniche, di avere accesso e navigare tra milioni e milioni di informazioni (pagine web) distribuite su server disseminati a livello mondiale. Grazie alle sue caratteristiche di flessibilità e trasparenza, ha permesso di semplificare l'accesso e la gestione di grandi moli di dati (database, archivi), così come il monitoraggio e l'amministrazione di prodotti, servizi e processi, ha contribuito a creare nuove professionalità e, soprattutto, ha dato l'impulso fondamentale per lo sviluppo della "new economy" nell'era dell'Internet Society.

L'accessibilità e la visibilità, a livello mondiale, delle informazioni pubblicate via web ha consentito anche a piccole aziende di avere, al pari delle grandi, opportunità per espandere il proprio business e strumenti per rendersi competitive nel mondo del commercio elettronico (e-commerce). In questo ambito, un fattore dominante è dato dalla qualità del servizio percepita dagli utenti.

D'altro canto, la crescente richiesta di servizi Internet e, soprattutto, l'enorme numero di accessi a dati e servizi resi disponibili da web server hanno determinato un considerevole aumento del traffico di rete che, in alcuni casi, può provocare il congestionamento delle linee di trasmissione con conseguente degrado nelle prestazioni.

Per cercare di limitare questi problemi, vengono comunemente utilizzate due differenti tecnologie: *Web Replication* e *Web Caching*. Entrambe le tecnologie cercano di ridurre il fenomeno della congestione della rete e di migliorare la qualità del servizio, ma utilizzano approcci diversi.

Web Replication si basa sul concetto di ridondanza. Più server web, dislocati in differenti aree geografiche offrono gli stessi servizi replicando, totalmente o solo in parte, dati e software disponibili in un particolare sito. Gli obiettivi sono: ridurre il traffico indirizzato a server con alta frequenza di accessi, diminuendo il carico computazionale della macchina che offre tali servizi, e dare all'utenza un servizio migliore, consentendo l'accesso a delle repliche dotate di una "migliore connessione" telematica. Questa tecnica non riduce effettivamente il traffico di rete, ma lo ripartisce su differenti cammini, dirigendo le richieste verso server distribuiti che replicano dati e servizi.

In contrapposizione, *Web Caching* indica la memorizzazione temporanea di pagine web in locazioni più vicine al richiedente (*cache server*) rispetto alla sorgente dell'informazione (*origin web server*) [1]. In tal modo, richieste successive dello stesso oggetto possono essere soddisfatte direttamente dal cache server, riducendo il tempo che trascorre tra la richiesta e la disponibilità dell'oggetto, cioè la latenza. Abbreviando il cammino di rete degli oggetti Internet, la tecnologia di web caching ottimizza l'utilizzo della banda trasmissiva, riduce il traffico sulla rete, e migliora la qualità del servizio offerto all'utente. Ma, di contro, può comportare problemi di inconsistenza tra l'oggetto originale e le copie temporanee.

Di recente sviluppo e di rilevante interesse sono le *Content Delivery Network* e *Content Distribution Network* (CDN), la tecnologia emergente nel campo della distribuzione efficiente di contenuti multimediali in tempo reale (multimedia streaming). Sebbene le CDN possano essere implementate in differenti architetture e con differenti protocolli, si basano su concetti comuni sia alla replicazione sia al web caching. Sono reti specializzate nella distribuzione e

nella consegna di contenuti multimediali on-line (come, ad esempio, film) o, comunque, di informazione soggetta a cambiamenti molto frequenti (news, riviste-on-line) e sono progettate per diffondere il maggior numero di copie di contenuti in punti di snodo della rete stessa (solitamente su server dedicati). Esse sfruttano informazioni sulla topologia della rete (per riconoscere la prossimità di un certo client ad una certa copia di contenuto) e particolari meccanismi per valutare il carico dei server (che distribuiranno i contenuti), in modo da connettere dinamicamente il client alla replica[2] "più conveniente" in termini di minore latenza nella consegna del contenuto richiesto.

Questo lavoro di tesi si basa sullo studio e sulla sperimentazione della tecnologia di *Transparent Web Caching* in diverse configurazioni, con particolare attenzione verso soluzioni a basso costo e ad alta scalabilità. La sperimentazione, effettuata nella rete locale dell'Istituto per le Applicazioni Telematiche del CNR di Pisa, si inserisce in uno studio sul monitoraggio del traffico e dei servizi di rete del CNR.

I principali obiettivi dello studio sono stati: valutare la tecnologia di transparent web caching in un ambiente operativo, misurare le prestazioni ottenute con l'utilizzo di differenti configurazioni, rilevare eventuali problemi e, infine, verificare la rispondenza degli utenti all'introduzione del servizio, al fine di comprenderne pienamente vantaggi e svantaggi.

Il lavoro si articola in cinque capitoli. Il primo capitolo è l'introduzione. Il secondo capitolo descrive la tecnologia World Wide Web e delinea i concetti alla base delle tecnologie di web replication, web caching e delle content delivery network.

Il terzo capitolo si concentra sulla tecnologia di web caching descrivendone protocolli, architetture e possibili implementazioni. Particolare attenzione è stata dedicata alle implicazioni della versione 1.1 del protocollo HTTP sulla tecnologia di web caching. Questo studio della

---

[1] Il server su cui una data risorsa risiede o è stata creata

[2] replica origin server, è un origin server che conserva una copia (replica) di una risorsa, ma che può agire come un riferimento ufficiale per le richiste dei client.

tecnologia è stato necessario e fondamentale per una piena comprensione della sperimentazione.

Nel quarto capitolo vengono descritti i servizi di cache server del CNR e del GARR, e sono introdotte le motivazioni che hanno indotto ad effettuare la sperimentazione. Viene quindi descritta in dettaglio la sperimentazione: l'ambiente operativo, i test effettuati, le variabili misurate, l'analisi dei risultati ed il confronto tra le differenti configurazioni adottate.

Il lavoro si chiude con conclusioni e sviluppi futuri.

2. La tecnologia World Wide Web

Internet è una rete di reti. Essa, infatti, è costituita da reti eterogenee (locali e geografiche) interconnesse tramite dispositivi chiamati *router*. Il suo "punto di forza" è rappresentato dall'utilizzo di una suite di protocolli di comunicazione, detti TCP/IP (Transmission Control Protocol/Internet Protocol), che offrono un'astrazione sia dall'HW/SW del nodo interconnesso, sia dal cablaggio di rete (ethernet, token ring, FDDI, etc.) utilizzato. Con l'utilizzo di questi protocolli, viene garantità l'interoperabilità tra sistemi e reti di differenti costruttori. I servizi offerti dalla rete Internet sono molteplici: la posta elettronica, il trasferimento di file (File Transfer Protocol), la connessione remota con emulazione terminale (*telnet*), ma certamente il più famoso e il più amato è il World Wide Web (WWW) o, più semplicemente, il web.

Nel seguito abbiamo assunto la conoscenza dei concetti base della rete Internet e ci siamo concentrati unicamente sulla tecnologia WWW. Per approfondimenti si rimanda alla letteratura [33], [35].

Il World Wide Web è una tecnologia che consente di distribuire dati e, più in generale, per condividere risorse in Internet, in modo semplice e robusto.

L'architettura WWW è di tipo client-server. La tecnologia si basa su due componenti fondamentali: il protocollo di comunicazione HTTP (HyperText Transfer Protocol), che definisce il formato dei messaggi e le regole per lo scambio di dati tra client e server, e il linguaggio HTML (HyperText Markup Language) che definisce la sintassi di base delle pagine web.

Il server WWW gestisce risorse Internet: nel caso più comune si tratta di informazioni, siano esse non strutturate (*flat file*) o organizzate in database e archivi, anche se le sue funzionalità non si limitano a questo. Il client, comunemente chiamato browser, invia una richiesta per l'accesso (in lettura, scrittura o esecuzione) ad una delle risorse gestite dal server, il server

esegue la richiesta e restituisce una risposta. Il protocollo di comunicazione consiste, perciò, di due elementi distinti: l'insieme delle *richieste* generate dai client e l'insieme delle *risposte* restituite dai server. La Figura 2.1 mostra uno schema logico della interazione tra client e server HTTP, che evidenzia gli RFC[3], cioè le specifiche di riferimento, a cui gli sviluppatori devono attenersi nella implementazione del protocollo.

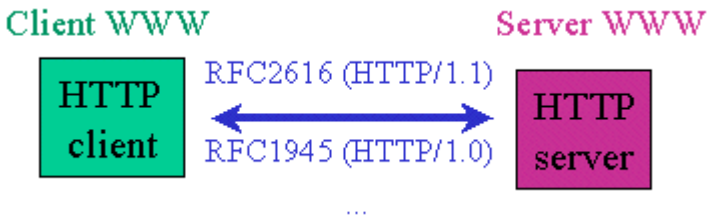

Figura 2.1 - Interazione Richieste/Risposte HTTP (schema logico)

In modo informale, potremmo dire che un server web esplica due funzionalità: "pubblica" informazioni (pagine web) e rende disponibili servizi.

Il termine *publishing* indica la possibilità di rendere disponibili contenuti multimediali (testo, immagini, animazioni, …) in Internet. Un server web, infatti, è in grado di rendere disponibili tutte le informazioni memorizzate in una certa porzione del File System della macchina su cui risiede. In fase di installazione/configurazione del server deve essere specificata la *documents root* (docs in Figura 2.2), cioè la directory radice che delimita la zona di disco resa "visibile".

---

[3] Gli RFC (Request For Comments) sono una raccolta di note informative, specifiche tecniche ed implementative relative ad architetture e protocolli della rete Internet (originariamente ARPANET). Il primo passo verso la definizione di un RFC è la stesura del documento in forma di document Internet Draft e la sua messa in rete, in modo che possa essere letto e commentato da tutta la comunità Internet. Le attività di discussione avvengono tramite mailing list ed incontri quadrimestrali che permettono agli specifici gruppi di consolidare, migliorare o, eventualmente, criticare le soluzioni proposte. Per passare alla fase successiva, l'Internet draft deve rimanere sotto discussione per un periodo di 6 mesi e deve avere almeno due implementazioni di costruttori diversi interoperabili.

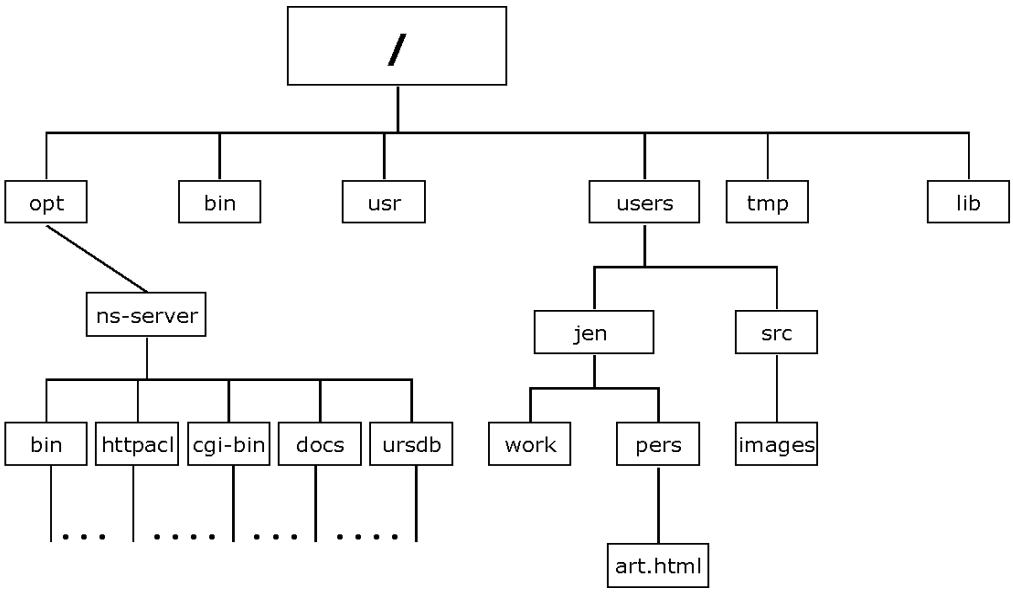

Figura 2.2 - Schema del File System

La possibilità di rendere disponibili servizi via web indica, in generale, l'esecuzione di programmi per conto dei client. L'esecuzione di alcuni di questi programmi richiede all'utente l'inserzione di parametri. Il meccanismo del passaggio dei parametri si basa sui valori direttamente forniti dall'utente al server (per esempio, tramite *form,* cioè moduli elettronici) e in alcuni casi, anche sul contenuto di variabili di ambiente, impostate dal server sulla base di informazioni ricevute dal client. L'output del programma viene trasformato in formato HTML e restituito dal server al client. Il server deve, perciò, conoscere quali eseguibili o script è autorizzato a "lanciare". In fase di configurazione del server, vengono specificate una o più directory[4], in cui saranno memorizzati i programmi o gli eseguibili, e/o vengono definite le estensioni accettabili (per esempio, .cgi). Si comprende facilmente come sia importante adottare

[4] Spesso viene utilizzata una directory chiamata cgi-bin, dal nome Common Gateway Interfaces (CGI) comunemente dato a questi programmi.

configurazioni restrittive (ad esempio, i diritti con cui i processi andranno in esecuzione), per evitare che hacker o, comunque, utenti mal intenzionati, possano approfittare di bachi nei programmi o di sistemi mal configurati e che possano prendere il controllo della macchina o danneggiarla. La pagina visualizzata all'utente (la risposta HTTP) verrà generata di volta in volta in relazione al valore dei parametri di input, e quindi può cambiare, per tale ragione queste pagine vengono chiamate *dinamiche*. Un altro esempio di pagine dinamiche sono pagine che modificano il loro aspetto in base al profilo dell'utente che ha richiesto la pagina inserendo, ad esempio, banner e/o advertisement "personalizzati".

Un server HTTP è un processo che attende richieste TCP, di solito sulla porta 80 (che è la porta di default del protocollo). Istanze del server possono, comunque, girare su porte differenti, al fine di rendere disponibili servizi ad accesso controllato come, ad esempio, interfacce per la gestione di compiti amministrativi, o per l'accesso a servizi a pagamento (come library on-line).

Nel seguito, indicheremo con il termine *oggetto* (Internet object) qualsiasi file trasmesso da un server web a un client (in risposta ad una richiesta), sia esso un file HTML, una immagine, un file audio, postscript, pdf, etc., indipendentemente dal fatto che sia stato generato staticamente o dinamicamente.

Nei prossimi paragrafi ci addentrermo brevemente nelle specifiche di riferimento. Come fonti di informazione, ci riferiremo alle specifiche elaborate dal W3C, il World Wide Web Consortium[5] e agli RFC sviluppati dall'IETF, l'Internet Engineering Task Force (http://www.ietf.org).

[5] Il W3C (http://www.w3.org/) nasce nell'Ottobre 1994 da un accordo tra MIT e CERN, con l'intento di guidare lo sviluppo del World Wide Web verso le sue piene potenzialità, mediante lo sviluppo di protocolli comuni che promuovono la sua evoluzione e assicurano la sua interoperabilità. Oggi il W3C conta più di 400 Organizzazioni Membri.

2.1. Il Linguaggio HTML

Da tutti conosciuto come il linguaggio del web, l'HTML (HyperText Markup Language) è un linguaggio di markup derivato dalla famiglia dei linguaggi SGML (Standard Generalized Markup Language). L'HTML è il linguaggio usato per creare pagine web attraverso il posizionamento di speciali istruzioni, dette *tag*, che trasmettono al browser le informazioni necessarie su come visualizzare il contenuto del documento. I tag seguono una particolare sintassi: il nome dell'elemento è sempre racchiuso tra parentesi angolate. Non vengono visualizzati nella finestra del browser ma forniscono le istruzioni necessarie per la struttura della pagina web, la sua visualizzazione e le funzioni in essa contenute, come i link. La maggior parte dei tag HTML sono contenitori, nel senso che hanno un tag di apertura e un tag di chiusura: il nome dell'elemento è lo stesso, ma nel tag di chiusura è preceduto da uno slash (<TITLE> … </ TITLE >).

Una pagina web può contenere testo, ma anche link o *embedded items*, cioè quegli oggetti quali immagini, audio, video, etc., che risiedono su file autonomi. Un documento HTML è formato da due parti principali: una intestazione (head) e un corpo (body).

L'*head* contiene informazioni generali sul documento (*metainformazioni*) che non vengono visualizzate sulla finestra del browser, quali autore, parole chiave, codifica dei caratteri, lingua usata, data di scadenza (expired date), link a file esterni come javascript o fogli di stile, titolo della pagina visualizzata. Quest'ultima è l'unica informazione visibile che ritroviamo sulla barra del titolo della finestra del browser e che viene memorizzata nei bookmark.

Il *body* è il contenuto vero e proprio della pagina HTML, quello cioè che verrà visualizzato sulla finestra del browser. Le Figure 2.3, 2.4 mostrano un esempio di codice HTML, la sua resa sulla finestra del browser è mostrata in Figura 2.5

```
<head profile="http://www.w3.org/2000/08/w3c-synd/#">

  <meta http-equiv="Content-Type" content="text/html; charset=us-ascii" />
  <meta http-equiv="PICS-Label" content='(PICS-1.1 "http://www.classify.org/safesurf/" l gen true for
        "http://www.w3.org" by "philipd@w3.org" r (SS~~000 1 SS~~100 1))' />
  <meta http-equiv="PICS-Label" content='(PICS-1.1  "http://www.rsac.org/ratingsv01.html" l gen true
        comment "RSACi North America Server" by "philipd@w3.org" for "http://www.w3.org" on
        "1996.04.16T08:15-0500" r (n 0 s 0 v 0 l 0))' />
  <meta name="keywords" content="W3C, World Wide Web, Web, WWW, Consortium, computer,
        access, accessibility, semantic, worldwide, W3, HTML, XML, standard, language, technology,
        link, CSS, RDF, XSL, Berners-Lee, Berners, Lee, style sheet, cascading, schema, XHTML,
        mobile, SVG, PNG, PICS, DOM, SMIL, MathML, markup, Amaya, Jigsaw, free, open source,
        software" />
  <meta name="description" content="W3C is over 400 organizations leading the World Wide Web to
        its full potential. Founded by Tim Berners-Lee, the Web's inventor. The W3C Web site hosts
        specifications, guidelines, software and tools. Public participation is welcome. W3C supports
        universal access, the semantic Web, trust, interoperability, evolvability, decentralization, and
        cooler multimedia." />

<title>The World Wide Web Consortium</title>

<link rel="stylesheet" type="text/css" href="StyleSheets/home" />

</head>
```

Figura 2.3 - Esempio di codice dell'*head* di una pagina HTML

```
<body>

  <h1 id="logo"><img alt="The World Wide Web Consortium (W3C)" height="48" width="315"
        src="Icons/w3c_main" /></h1>
  <h2 id="slogan"><i>Leading the Web to its Full Potential...</i></h2>

  <div class="banner"><a class="bannerLink"href="Consortium/Activities">Activities</a> | <a
        class="bannerLink" href="TR/">Technical Reports</a> | <a class="bannerLink"
        href="Help/siteindex">Site Index</a> | <a class="bannerLink"
        href="Consortium/">About W3C</a> | <a class="bannerLink"
        href="Consortium/Contact">Contact</a></div>
  …..
</body>
```

Figura 2.4 - Esempio di codice del *body* di una pagina HTML

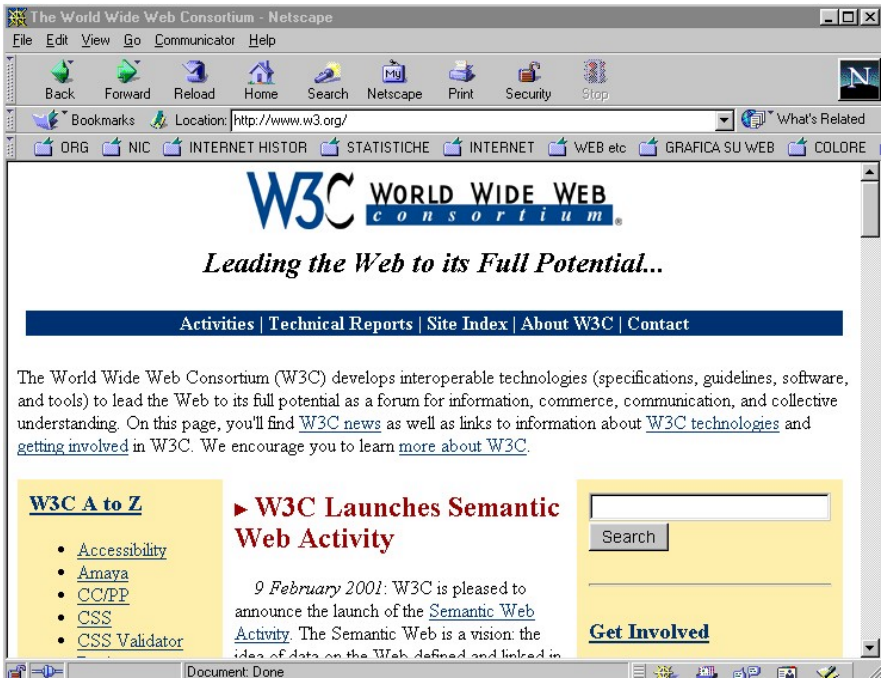

Figura 2.5 - Resa del codice HTML sulla finestra del browser

Una delle caratteristiche più importanti del web è proprio la sua natura ipermediale, attraverso i link possiamo relazionare documenti HTML diversi, usare mappe sensibili o creare link a protocolli non web.

Nel seguito, non ci soffermeremo sulla sintassi HTML ma introdurremo solo il link, che rappresenta il meccanismo per individuare le risorse Internet disponibili via web server. Per approfondimenti si rimanda a [30].

Per entrare nella terminologia degli RFC di riferimento, indichiamo con il termine *risorsa* qualsiasi oggetto di rete o servizio, e, con il termine *entità* (entity) l'informazione effettivamente trasferita in una richiesta o una risposta. Infatti, dato che le risorse possono avere differenti rappresentazioni (per es. diversi linguaggi, formati dei dati, size, risoluzione, etc.), l'entità

restituita può variare. La negoziazione del contenuto, tra client e server, è il meccanismo che consente di selezionare l'appropriata rappresentazione di una richiesta.

L'entità trasferita nel messaggio[6] HTTP consiste di due parti: le metainformazioni (generali, specifiche del messaggio o dell'entità) trasmesse in forma di *header*, e il suo contenuto (la rappresentazione) detto *entity body*.

Ogni risorsa in Internet è identificata per mezzo di una stringa detta URI (Uniform Resouce Identifier), che identifica in modo univoco la risorsa. La URI può specificare una locazione, cioè una URL (Uniform Resource Locator), oppure un nome, cioè una URN (Uniform Resource Name), che consente di individuare la risorsa indipendentemente dal nodo su cui è allocata (ad esempio, nel caso di web replication). Per approfondimenti si rimanda alla lettura dell'RFC 2396 [10].

Ritornando alla sintassi di un link, esso è costituito da due parti: un riferimento ipertestuale, specificato per mezzo di una URI (che identifica in modo univoco la risorsa) ed una etichetta (che dà informazioni sul contenuto riferito), che sarà visualizzata dal browser in carattere sottolineato (ad indicare il collegamento): <A HREF ="URI"> etichetta </A>.

Di solito, una risorsa viene identificata dalla sua URL, che è conforme alla seguente sintassi[7]:

"protocol:" "//" host [":" port] [abs_path ["?" query]]

dove *protocol* è il protocollo utilizzato (http), *host* è il nome o l'indirizzo IP del server web, *port* indica la porta TCP su cui il server è in ascolto (la porta 80 è il default e può essere omessa), *abs_path* rappresenta il path assoluto della risorsa all'interno del server (a partire dalla document root), e *query* è una stringa opzionale che può essere utilizzata per il passaggio

[6]L'unità base della comunicazione HTTP, consistente di sequenze strutturate di byte trasmessi attraverso la connessione.

[7] I parametri tra parentesi quadre sono opzionali

(metodo PUT[8]) dei parametri all'eventuale programma.

Negli ultimi anni, abbiamo visto una esplosione nel mercato degli strumenti di produzione di pagine web. Gli editor HTML di tipo WYSIWYG (what-you-see-is-what-you-get) infatti, forniscono interfacce grafiche che permettono una semplice e rapida creazione di pagine HTML.

Anche il linguaggio del web si è evoluto. Man mano che le applicazioni sono diventate più sofisticate, è nata l'esigenza di un linguaggio "aperto" cioè estendibile con nuovi tag (per soddisfare differenti esigenze), da qui nasce la definizione di XML (eXtensible Markup Language), che permette di separare il contenuto dei dati dalla loro presentazione, o di altri linguaggi specifici, come SMIL (Synchronized Multimedia Integration Language) per la sincronizzazione ed integrazione di multimedia.

Di recente definizione è il linguaggio XHTML (eXtensible HTLM) che unisce i vantaggi dell'HTML con quelli dell'XML. Comunque, per gli scopi di questo lavoro, gli aspetti del linguaggio non sono rilevanti e per questo non saranno ulteriormente approfondite. Per approfondimenti si rimanda al sito del World Wide Web Consortium (W3C): http://www.w3.org.

2.2. Il Protocollo HTTP

L'HTTP è un protocollo a livello applicativo, generico e stateless, che può essere utilizzato per moltissimi scopi, che vanno dall'ipertesto alla gestione, il monitoraggio e l'amministrazione di servizi o di sistemi distribuiti. Il protocollo è di tipo richiesta/risposta. I messaggi HTTP consistono di richieste dal client al server e di risposte dal server al client:

```
HTTP-message   = Request | Response
```

Come abbiamo già detto in precedenza, la comunicazione HTTP di solito avviene su

[8] § 2.2

connessioni TCP/IP, ma questo non è un vincolo, dato che il protocollo richiede solo l'utilizzo di un trasporto affidabile.

La prima versione del protocollo (HTTP/0.9) risale al 1990 e definisce un semplice trasporto di dati (raw data) tra client e server. Nel 1996 è stata introdotta la versione 1.0 (definita nell'RFC 1945 [9]), che introduce notevoli miglioramenti al protocollo, tra cui l'inclusione di dati in formato MIME[9] e l'introduzione di *modificatori* (modifiers) semantici delle richieste/risposte. Ciò nonostante, anche l'HTTP/1.0 non prende ancora sufficientemente in considerazione gli effetti di gerarchie di proxy o cache e i problemi di efficienza del protocollo. Da qui l'esigenza di una nuova versione con requisiti più stringenti (per evitare implementazioni inaccurate o incomplete) e, al tempo stesso, più flessibile sia per supportare la vasta diversità delle configurazioni già presenti (o future), sia per soddisfare le necessità degli sviluppatori di applicazioni web che richiedono alta affidabilità e, in caso di fallimento, indicazioni precise sulle cause.

Per il trasferimento delle entità, i messaggi HTTP scambiano dati in un formato *MIME-like*[10]. Il client invia una richiesta al server nella forma di un metodo, la URI della risorsa su cui il metodo deve essere applicato, e la versione del protocollo, seguito da zero o più header, una linea vuota (ad indicare la fine degli header) e il corpo (body) del messaggio, secondo la

---

[9] MIME, the Multipurpose Internet Mail Extensions (RFCs 2045 [Fre 96a], 2046 [Fre 96b], 2047 [Moo 96], 2048[Fre 96c], 2049 [Fre 96d]) ridefinisce il formato del body di un messaggio descritto nell'RFC822, al fine di includere dati non-testuali e parti strutturate.
MIME definisce i tipi di dati: text, image, audio, video, application, message and multipart. Per ogni tipo di dato possono essere definiti molteplici sottotipi (per es. image/tiff, image/gif, etc.). Il tipo e sottotipo (media type e subtype) dei dati sono espressi dall'header Content-type (p.es. Content Type: image/gif). Il meccanismo è estendibile con l'introduzione di nuovi formati (ad esempio, un nuovo sottotipo).
Un messaggio MIME multiparte è un messaggio strutturato composto da più parti che possono contenere differenti tipi di dato. Le parti del messaggio sono separate da un limitatore: una stringa univoca detta "boundary". Ogni boundary è preceduto da due trattini (--). Soltanto l'ultimo boundary è anche terminato dai due trattini, allo scopo di indicare la fine del messaggio.

[10] L'RFC 2110 [32] descrive come applicare MIME nello scambio di messaggi HTTP.

seguente sintassi:

```
Request      = Method    Request-URI    HTTP-Version    CRLF
                 *(( general-header
                  | request-header
                  | entity-header ) CRLF)
                 CRLF
                 [ message-body ]
```

Gli header contengono metainformazioni (o metadati) che descrivono il contenuto del messaggio. I *general-header* contengono informazioni generali (ad esempio, la data) e possono essere presenti sia nella richiesta che nella risposta. I *request-header,* come dice il nome, sono relativi alla sola richiesta e ne danno le informazioni relative. Infine, gli *entity-header* sono metainformazioni relative all'entità trasmessa (dimensione, data di creazione, etc.), che possono comparire sia nella richiesta, se trasporta un entity body, sia nella risposta.

I metadati sono veramente importanti, poiché consentono, ad esempio, di validare la consistenza di una risorsa, senza il trasferimento dell'intera entità (ma solo degli header), sulla base (sempre a titolo di esempio) della dimensione e della data di creazione del file.

Nel seguito, sono descritti in modo sintetico i metodi (method) descritti nella versione 1.1 del protocollo.

Il metodo GET è utilizzato per accedere ad una risorsa. Se la URI riferisce una informazione, praticamente l'accesso è in lettura e nell'entity body viene restituita la "opportuna" rappresentazione della risorsa. Se la URI, invece, riferisce un processo che produce dati, viene restituito il risultato dell'esecuzione. La Figura 2.6 mostra un esempio di sessione HTTP. Il client stabilisce la connessione, invia quindi una richiesta contenente il metodo GET e una linea bianca a significare la fine della richiesta. Il server (www.cnr.it) restituisce delle informazioni sullo stato della richiesta (status), le metainformazioni (general-header e entity-header), una linea bianca (per indicare la fine degli header) e, infine, il body vero e proprio dell'entità (il file

html).

```
telnet www.cnr.it 80
Trying...
Connected to www.cnr.it.
Escape character is '^]'.
GET / HTTP/1.1
Host: www.cnr.it

HTTP/1.1 200 OK
Date: Tue, 30 Jan 2001 14:27:47 GMT
Server: Apache/1.3.9 (Unix) ApacheJServ/1.0 PHP/3.0.12
Last-Modified: Mon, 18 Oct 1999 12:44:52 GMT
ETag: "1f900f8-6fd-380b1644"
Accept-Ranges: bytes
Content-Length: 1789
Content-Type: text/html
X-Pad: avoid browser bug

<html>
<head><title>CNR prima pagina</title></head>
<frameset cols="100%" rows="60,*,42" border="1">
...
```

Figura 2.6 - Esempio di Richiesta/Risposta HTTP

La semantica della GET viene modificata in *conditional GET* se la richiesta include i campi If-Modified-Since, If-Unmodified-Since, If-Match, If-None-Match, o If-Range dell'header di richiesta. Questi campi rappresentano dei meccanismi per validare il dato, cioè per capire se la copia autoritativa, quella residente sull'origin server, è cambiata rispetto a quella temporanea, residente, ad esempio, su un cache server che si trova lungo il cammino della richiesta. Una GET condizionale richiede che l'entità sia trasferita solo se la condizione descritta è soddisfatta, altrimenti la semantica si trasforma in quella di un metodo HEAD, cioè sono trasferite solo le metainformazioni della risorsa. La semantica del metodo GET si modifica in una *partial GET* se la richiesta include l'header Range. Una GET parziale richiede il trasferimento di una parte dell'entità. Sia la conditional GET che la partial GET consentono di ridurre l'utilizzo della banda trasmissiva, evitando di trasferire dati già posseduti (e aggiornati) dalla cache o dal client.

Il metodo HEAD è utilizzato per richiedere i metadati di una risorsa, cioè i dati che la descrivono come, ad esempio, la data di creazione e la dimensione, o che la possano validare (come il campo `ETag` del response-header) ed è spesso usato per testare la validità, la accessibilità e le recenti modifiche di un link.

I metodi GET e HEAD sono *safe*, nel senso che effettuano il recupero di dati esistenti, mentre gli altri metodi possono operare sulla risorsa e, quindi, una loro interazione è più delicata. Le risposte di questi due metodi possono essere "cacheate" (quando non esplicitamente vietato) [11].

Il metodo POST è utilizzato per richiedere che l'origin server accetti l'entità racchiusa nella richiesta come subordinata della risorsa specificata nella URI. Questo metodo permette di inviare messaggi a newsgroup o mailing list, di aggiornare database con operazioni di *append*, di sottomettere il contenuto di un form ad un processo in grado di gestire tali dati. In sostanza, la funzione effettuata dal metodo POST è determinata dal server e, di solito, dipende dalla URI richiesta.

Il metodo OPTIONS rappresenta una richiesta di informazioni sulle opzioni di una comunicazione disponibili nella catena richieste/risposte identificata dalla URI richiesta. Questo metodo permette ai client di identificare le opzioni e/o i requisiti associati ad una risorsa o le caratteristiche del server, senza implicare alcun'azione sulla risorsa.

Il metodo PUT richiede che l'entità inclusa sia memorizzata sotto la URI specificata. Se la URI riferisce una risorsa già esistente, dovrebbe essere considerata come una modifica di quella originaria, altrimenti il server dovrebbe creare una nuova risorsa e dovrebbe informare il client nella risposta (codice 201, Created).

Il metodo DELETE richiede che l'origin server cancelli la risorsa identificata dalla URI.

Il metodo TRACE è utilizzato per invocare un remoto, application-layer loop-back sulla

---

[11] § 3.8.2

richiesta. Il desutinatario finale (l'origin server o il primo proxy o gateway) che riceve il valore zero nel campo Max-Forwards del request-header, dovrebbe rispedire il messaggio indietro al client. TRACE permette al client di vedere cosa è stato ricevuto all'altro capo della catena della richiesta ed utilizzare questi dati per testing o informazioni diagnostiche.

Infine, il metodo CONNECT è riservato per usi futuri. I metodi PATCH, LINK, UNLINK erano stati definiti nelle precedenti versioni del protocollo, ma non sono implementati.

La Figura 2.1 mostra un esempio di metodo GET: una richiesta di lettura di una risorsa. Il server riceve la richiesta, la interpreta, se possibile la esegue, e invia una risposta al client. La risposta inizia con una linea di status, che include la versione del protocollo, un codice di stato (successo o errore) e un messaggio MIME-like, contenente informazioni sul server, metainformazioni sull'entità e, eventualmente, un entity-body.

```
Response      = HTTP-Version   Status-Code   Reason-Phrase CRLF
                *(( general-header
                 | response-header
                 | entity-header ) CRLF)
                CRLF
                [ message-body ]
```

Lo Status-Code è un codice di ritorno (un intero di tre cifre) del tentativo di interpretare ed eseguire la richiesta. Al codice può essere aggiunta una stringa (Reason-Phrase), intesa a fornire una descrizione testuale del codice stesso.

La prima cifra dello Status-Code definisce la classe di risposte (le altre due cifre non hanno nessuna funzione di categorizzazione). Sono ammessi 5 valori:

1xx: Informazione - Richiesta ricevuta, il processo continua

2xx: Successo - L'azione è stata ricevuta, accettata e interpretata con successo

3xx: Redirezione - Occorrono altre azioni per completare la richiesta

4xx: Client Error - La richiesta non è sintatticamente corretta o non può essere completata

5xx: Server Error - Il server non è riuscito a servire una richiesta apparentemente valida

Per dettagli riferire l'Appendice A.

Un altro problema affrontato nella versione 1.1 del protocollo HTTP è relativo all'efficienza del server (utilizzo delle risorse come la CPU e memoria). Abbiamo già descritto l'interazione tra client e server. Il client richiede una pagina, il server restituisce la risposta, il client interpreta il codice HTML ed effettua le richieste per eventuali oggetti (*embedded object*) presenti. Nelle prime versioni del protocollo, il client HTTP utilizza una differente connessione per ogni oggetto. Una pagina ad esempio, che contiene del testo e due immagini, richiede tre connessioni. L'uso di immagini in linea e di altri dati associati conduce, spesso, un client a fare richieste multiple allo stesso server in un piccolo intervallo di tempo, cioè a stabilire una connessione TCP separata per acquisire l'oggetto corrispondente ad ogni URL, con aumento del carico sui server HTTP e possibili congestioni della rete.

La versione 1.1. del protocollo è stata progettata per ottimizzare il dialogo tra client e server e consente sia il *pipeline* di comandi da parte del client, sia l'invio da parte del server di più oggetti nella stessa connessione (persistent connection). Essa permette di stabilire connessioni persistenti, consentendo di incanalare (pipeline) richieste e risposte in una unica connessione. Il pipeline permette ai client di fare richieste multiple senza aspettare ogni risposta, consentendo l'utilizzo più efficiente di ogni singola connessione TCP. In questo modo, si riduce anche la congestione della rete, perché si diminuisce il numero di pacchetti dovuti alle susseguirsi di connessioni TCP e si concede al protocollo (TCP) tempo sufficiente per determinare lo stato dell'eventuale congestione.

Una differenza significativa, tra l'HTTP/1.1 e le versioni precedenti, è che le connessioni persistenti sono il comportamento di default di ogni connessione HTTP (nella versione 1.0 sono presenti implementazioni sperimentali di connessioni persistenti). Le connessioni persistenti hanno un meccanismo per cui un client e un server possono segnalare la chiusura di una connessione, usando il campo Connection dell'header. Un client, che supporta connessioni

persistenti, può fare il pipeline delle sue richieste, spedendo più richieste multiple, senza aspettare per ogni risposta. Il server deve spedire le corrispondenti risposte nello stesso ordine in cui ha ricevuto le richieste. Studi precedenti hanno dimostrato che le connessioni persistenti hanno migliori prestazioni di quelle separate.

2.3. Web Replication

Come abbiamo già accennato nell'introduzione, la tecnologia di *Web Replication* implica la replicazione di dati e servizi su server dislocati in differenti aree geografiche. In questo paragrafo descriviamo brevemente i principi che ispirano le tecniche utilizzate per il collegamento dinamico di un utente alla replica "più conveniente", cioè le tecniche e le politiche di distribuzione che possono essere utilizzate nel bilanciamento del carico. Conti et al., in [16] differenziano tra strategie di distribuzione Mirror-based, che indirizzano il browser verso la replica geograficamente più vicina, tecniche DNS-based[12] che, nella configurazione più semplice, distribuiscono il carico utilizzando una politica round robin (basata su una rigida rotazione degli indirizzi IP[13]) e tecniche più sofisticate QoS-based (basate sulla Qualità del Servizio), che mirano ad ottimizzare l'URT (User Response Time), il tempo di risposta per l'utente, inteso come l'intervallo temporale che intercorre tra l'istante in cui l'utente effettua una richiesta e il rendering completo della pagina da parte del browser.

Le strategie di distribuzione del carico, sono classificabili in *statiche* e *dinamiche* [16]. Le tecniche statiche, solitamente, si basano su algoritmi deterministici o stocastici, utilizzando informazioni riguardanti il carico medio dei sistemi, piuttosto che quello corrente (*current*

[12] Il Domain Name System (o Service) è un servizio Internet che traduce i nomi di dominio in indirizzi IP. Poiché i nomi di dominio sono alfabetici, sono più facili da ricordare per gli utenti, mentre Internet è basata sugli indirizzi IP. Ogni qualvolta che viene utilizzato un nome di dominio, viene invocato il DNS per tradurlo nel indirizzo corrispondente IP.

[13] In Internet, ogni sistema è univocamente individuato da un indirizzo che identifica la rete e all'interno di essa l'host: l'indirizzo IP.

workload). Per questo, anche se la loro implementazione risulta di limitata complessità, spesso possono condurre a prestazioni non soddisfacenti. Le tecniche adattive implementano il bilanciamento del carico, reagendo dinamicamente ai cambiamenti dello stato del sistema. In generale, queste tecniche sono intrinsecamente più complesse di quelle statiche, poiché necessitano di mantenere informazioni a run-time, in modo da rispondere a possibili cambiamenti dello stato, ma sono sicuramente più efficaci. La distribuzione adattiva del carico dei server web, comunque, non è facile da ottenere, poiché la sua implementazione richiede un binding (collegamento) dinamico dei browser verso i server web. Il binding dinamico verso la replica "più conveniente" diviene difficile, a causa della tendenza ad utilizzare schemi di naming basati su locazione. Poiché le URL contengono l'hostname del server e la locazione della risorsa su uno specifico server, c'è un mapping intrinseco tra l'istanza di una risorsa e la sua locazione. Questo rende difficile la distribuzione del carico tra un certo numero di macchine, senza aumentare la "smartness" del software dal lato client. Da queste problematiche è, anche, nata l'esigenza di definire le URN[14].

In particolare, poiché i client accedono alle risorse specificando direttamente l'hostname del server, le tecniche che regolano il collegamento dinamico di un client ad un insieme di server (senza un qualsiasi aumento del traffico di messaggi in rete) devono gestire il binding dell'indirizzo IP corrispondente al nome di host e/o il routing del messaggio.

Per abilitare un certo numero di macchine a condividere indirizzi IP sono state sviluppate diverse soluzioni. Per esempio, per aumentare progressivamente la potenza di calcolo su un dato sito, l'implementazione di un servizio web può essere basata su un cluster di macchine cooperanti localmente distribuite, accedute tramite un singolo indirizzo IP, utilizzando una tecnica chiamata NAT (Network Address Translation)[15].

---

[14] § 2.1

[15] Uno standard Internet che abilita una LAN ad utilizzare un insieme di indirizzi IP per il traffico interno e un secondo insieme di indirizzi per il traffico esterno [21].

Cherkasova et al. in [13] descrivono una efficiente strategia per repliche residenti su cluster. La distribuzione di carico avviene sulla base di due parametri: la dimensione del working set e l'access rate del sito. Questo tipo di strategia può essere implementata utilizzando la feature Dynamic Update descritta nell'RFC 2136 [37] e presente nelle ultime versioni di BIND, che permette ad agenti autorizzati di aggiornare zone inviando speciali messaggi di update per aggiungere o cancellare record, senza dover riavviare il DNS server. È ovvio che una replicazione di contenuti a livello locale offre meno benefici rispetto ad una distribuzione di repliche a livello geografico.

Un approccio alternativo per la distribuzione del carico è fare uso della funzionalità di redirezione (*redirect)* del protocollo HTTP, che redirige le richieste dei client in arrivo verso un server, facente parte di un insieme di repliche. I client fanno una connessione iniziale al server principale (a cui punta la URL) ed esso risponde con un codice HTTP 302 (found: la risorsa richiesta risiede temporaneamente sotto una differente URL), indicando il server a cui fare riferimento. Quindi, il client invia la richiesta corrente (e le successive) al server specificato. Questo meccanismo ha, comunque, lo svantaggio di rendere visibili le URL di tutti i server disponibili, che potrebbero essere memorizzate in hot-list o indicizzate da motori di ricerca, rendendo quindi inefficiente il bilanciamento del carico [16].

2.4. Web Caching

Quando un browser richiede una pagina web, effettua una connessione HTTP verso l'origin server su cui risiede la risorsa.

L'idea alla base del funzionamento di un cache server è una estensione del concetto di "cache locale" utilizzato dai browser web oggi più diffusi, come Netscape Navigator e MS Internet Explorer. Il principio è semplice: quando viene richiesta una pagina web, essa è salvata su disco, in una cache locale. Se viene richiesta di nuovo, il browser, anziché inoltrare la richiesta al server, carica la copia precedentemente salvata su disco. Nei file di configurazione

di un browser è possibile condizionare il comportamento del browser perché vada, ad esempio, a controllare la validità degli oggetti presenti nella cache, con quelli presenti in rete, ad ogni sessione oppure ad ogni richiesta o mai.

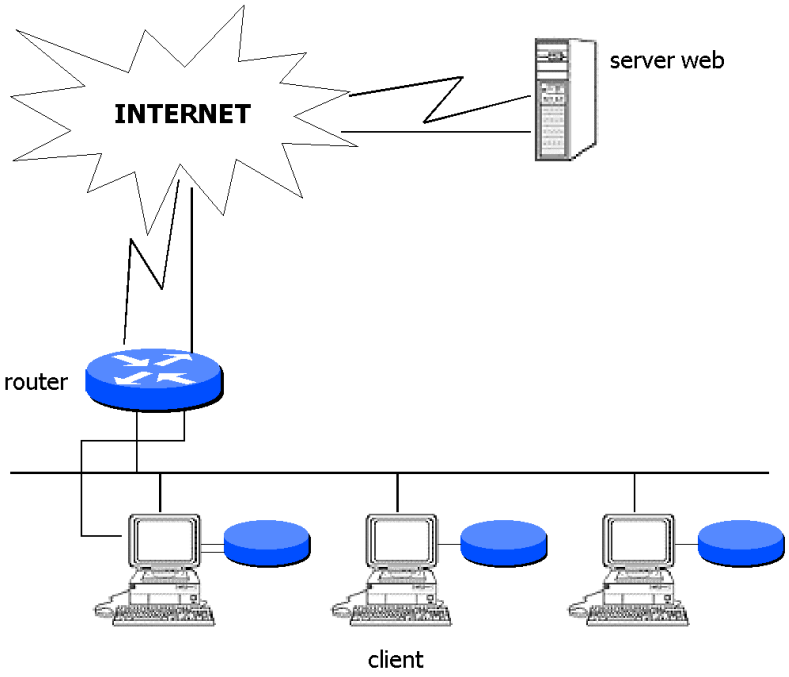

Figura 2.7 - Schema di Cache locale

Spostando questo concetto a livello di rete locale, un cache server è un sistema attraverso cui passano le richieste HTTP generate dai client presenti nella rete locale. Le richieste originate dagli utenti e indirizzate al server web (porta 80) sono trasmesse al cache server, che verifica la disponibilità dell'oggetto, all'interno della propria cache memory o dei dischi (*cache disk*), e, se questo è presente, lo fornisce al client che lo aveva richiesto. In caso di richiesta relativa ad un oggetto non presente nella cache, il cache server provvede a contattare il sito web originario definito nella URI e a recuperare l'oggetto restituendolo al client, conservandone però una copia, in modo da poter soddisfare direttamente eventuali richieste future dello stesso oggetto. In questo modo, questa tecnica riduce l'utilizzo della banda e diminuisce la latenza nel servire le richieste. Come nel caso della web replication un altro vantaggio indiretto è che i server web

con contenuti molto popolari vengono "alleggeriti" nel carico di richieste ricevute.

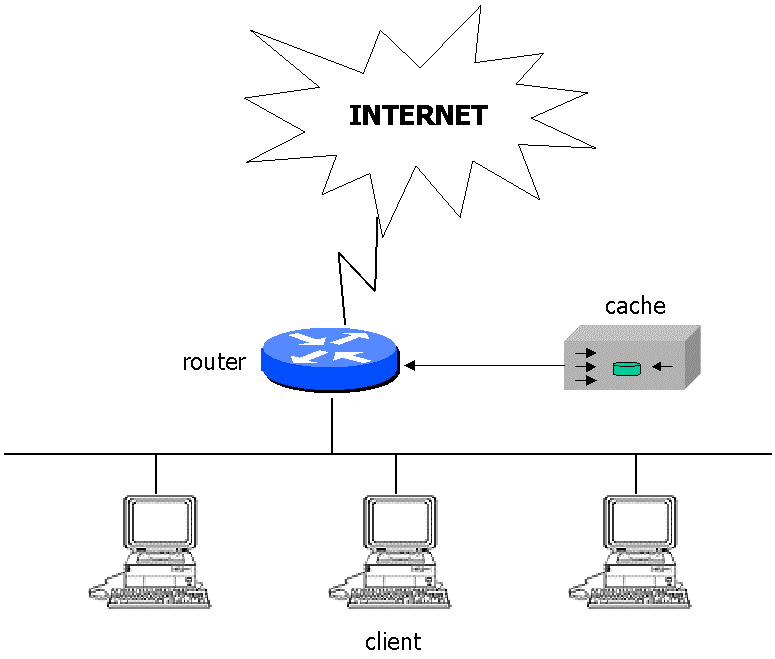

Figura 2.8 - Schema di una Cache inserita in una LAN

Come accennato in precedenza, esiste un problema di consistenza e validità degli oggetti serviti da cache. Per controllare la consistenza degli oggetti, sono impiegati meccanismi di scadenza o di validazione dei dati, resi disponibili dal protocollo HTTP/1.1, descritti più approfonditamente nel prossimo capitolo, interamente dedicato alla tecnologia di web caching.

Nel prossimo paragrafo descriveremo un'altra modalità di utilizzo di un proxy cache server come HTTPd-accellerator, che può essere interposto tra un server web e la rete Internet, allo scopo di rendere più veloce il server.

2.4.1. HTTPd-accelerator

Un HTTPd-accelerator [38] (comunemente detto anche reverse proxy[16]) è un cache server

[16] Vedere Nota 17

che può essere collocato "davanti" a un server web, allo scopo di servire le richieste in arrivo da client HTTP, relative ai dati che esso pubblica in rete. L'accellerator viene attivato sulla porta 80 (per ricevere le usuali richieste HTTP dei client), mentre il server web viene spostato su un'altra porta per servire le richiese dinamiche. L'accellerator sottrae, così, carico al server HTTP e alla rete interna. Dall'esterno non si vedono differenze, tranne una maggiore velocità. L'unica cosa di cui l'accelerator ha bisogno è sapere dov'è il server, per potere estrarre i dati.

Oltre che diminuire il carico di un server web, un accelerator può anche essere posto al di fuori di un firewall o di altri colli di bottiglia per dialogare con i server HTTP all'interno, riducendo il traffico e semplificando la configurazione. Due o più accelerator comunicanti via ICP[17] (Internet Cache Protocol) possono incrementare la velocità e la capacità di recupero di un servizio web per ogni singolo fallimento.

Il *redirector* (il server che fa la redirezione) può far operare un accelerator come un singolo front-end[18] per server multipli. Un accelerator può essere conveniente da utilizzare se si ha la necessità di spostare parte del file system da un server ad un altro oppure se server HTTP, amministrati separatamente, devono apparire sotto una singola gerarchia di URL. Se si vuole soltanto "cacheare" il "resto del mondo" per migliorare la performance del browsing degli utenti locali, l'accelerator mode è irrilevante. I siti che possiedono e pubblicano una gerarchia di URL, possono utilizzare un accelerator per migliorare l'accesso di altri siti verso questa gerarchia; i siti che desiderano migliorare l'accesso dei propri utenti locali verso altri siti possono utilizzare le web cache; molti siti implementano entrambe le configurazioni.

Una cache può, quindi, essere usata come un HTTPd accelerator, configurata per agire come un server HTTPd primario di un sito (sulla porta 80), spedendo i riferimenti miss al reale server HTTPd (sulla porta 81).

---

[17] § 3.5.1

[18] Nelle applicazioni client/server, la parte di client del programma è spesso chiamata "front-end" e la parte del server è detta "back-end**".**

In una tale configurazione, il web administrator rinomina tutte le URL non-"cacheabili" perché siano dirette verso la porta HTTPd (81). La cache serve i riferimenti agli oggetti "cacheabili", come pagine HTML o immagini GIF, mentre i riferimenti ad oggetti non-"cacheabili", come query e i programmi cgi-bin, sono serviti dal vero HTTPd sulla porta 81. Se le caratteristiche di utilizzo del sito tendono verso oggetti "cacheabili", questa configurazione può ridurre il carico di lavoro del sito web.

Con questo approccio, ponendo l'HTTPd-accelerator davanti al server HTTPd, una volta che un oggetto è nel cache-accelerator tutti i futuri *hit*[19] andranno a quella cache, e i *miss* andranno all'HTTPd nativo. Fin quando il vantaggio di usare un HTTPd-accelerator dipende dallo specifico workload dagli oggetti "cacheabili" e non-"cacheabili", l'HTTPd-accelerator non può degradare la performance di un sito. Inoltre, oggetti che ad una prima occhiata non appaiono "cacheabili", possono essere "cacheati" con una lieve perdita della trasparenza. Dato un workload impegnativo, ad esempio, si può accelerare l'accesso ad oggetti non-"cacheabili" come punteggi sportivi o quotazioni di azioni, se si ritiene che gli utenti possono tollerare un breve ritardo di refresh. Proprio da queste considerazioni è nata l'idea delle Conten Delivery Network e dei meccanismi che vi stanno sorgendo intorno, come l'Internet Draft WCIP (Web Cache Invalidation Protocol)[20] [28] che permette di "cacheare" oggetti dinamici, garantendo un intervallo di *freshness*[21].

2.5. Content Delivery Network

Con il termine *Content Distribution Network* (CDN) viene definita un'architettura di

---

[19] Con i termini hit/miss rate (ratio) viene definita la percentuale di successo/insuccesso, nel reperimento di un internet object: hit rate indica la percentuale di oggetti restituiti direttamente dalla cache rispetto al numero di quelli richiesti, miss rate la percentuale di oggetti non trovati o non validi, che sono stati richiesti dalla cache all'origin server.

[20] § 3.8.4

elementi di rete "web-based" ottimizzati per avere un efficiente delivery di contenuti web, mentre il termine *Content Delivery Network* descrive un'azione leggermente più ampia, poiché indica la distribuzione di un contenuto per iniziativa del *service provider* (indipendentemente dal trasporto utilizzato). Comunque, entrambi i termini sono generalmente usati con riferimento allo stesso servizio: la distribuzione di contenuti.

Un *content provider* (fornitore di contenuti) con sorgenti di dati molto popolari, cioè con un elevato numero di accessi, presenta forti rischi di congestionamento locale, perché una grande mole di richieste si concentra verso un'area circoscritta. Per limitare questi rischi, spesso il provider investe in un ampio insieme di server che implementano politiche di load balancing e potenzia le linee di comunicazione con connessioni ad alta velocità, in modo da poter soddisfare le crescenti richieste. Nonostante questi investimenti, l'utente finale può, comunque, percepire una visualizzazione del contenuto non soddisfacente, dovuta alla congestione della rete. Infatti, se l'insieme dei server rimane localizzato, anche se i client accedono a server diversi contemporaneamente, le comunicazioni si concentrano in un'area locale e il collegamento rischia di divenire il collo di bottiglia.

Una CDN abilita un service provider ad agire per conto del content provider, per trasmettere copie del contenuto di un origin server da più locazioni. L'aumento del numero e della diversità delle locazioni è progettato in modo da diminuire i tempi di attesa degli utenti.

Una CDN è composta di tre infrastrutture, per la redirezione, la consegna dei contenuti (*content-delivery)* e la distribuzione. La prima infrastruttura consiste di meccanismi che indirizzano il client verso uno specifico server; l'infrastruttura di content-delivery consiste di un insieme di server, solitamente HTTPd accellerator (detti *surrogate*[22] *server*), che trasmettono

---

[21] Una risposta è fresh se la sua età non ha già ecceduto il freshness lifetime (Nota 32), è stale se la sua età ha superato il freshness lifetime

[22] Un gateway collocato presso un origin server, o in un punto differente nella rete, delegato d'autorità ad operare in favore di uno o più origin server, lavorando tipicamente in stretta cooperazione con essi. Le risposte sono in genere salvate da una cache interna. I surrogate possono derivare le entry della cache da

copie del contenuto ad un insieme di client; l'ultima infrastruttura consiste di meccanismi che spostano il contenuto dall'origin server verso i surrogate. Una CDN serve le richieste di un client prendendo il contenuto da un server (solitamente un HTTPd accellerator o reverse proxy) "ben posizionato" nei confronti del client. Permettendo a diversi surrogati di agire al posto di un origin server, quindi, la CDN sposta il delivery del contenuto da un sito centralizzato a siti multipli (usualmente) molto distribuiti. Inoltre, la CDN può essere costruita con copie del contenuto poste vicine all'utente finale risolvendo alla radice il problema della dimensione, congestione o dei fallimenti della rete [12].

Per la distribuzione dei contenuti possono essere utilizzate tecniche differenti. Di solito le copie dei contenuti sono distribuite in accelerator-mode, e la redirezione delle richieste avviene via DNS. Johnson et al. in [26], pur riconoscendo la validità del servizio di distribuzione dei contenuti, confrontano le prestazioni di due CDN commerciali (offerte dai Content Provider Akamai e Digital Island), mostrando come nessuna delle due riesca a scegliere in modo consistente il miglior server tra quelli disponibili (in termini di latenza dell'oggetto recuperato). È, comunque, chiaro che nella filosofia del Content Provider non è necessario scegliere la "migliore" replica, ma una "accettabile", con tempi di risposta brevi.

Una altra strategia consiste nel separare le pagine HTML di base dai riferimenti ipertestuali e dagli embedded item (che di solito risiedono sullo stesso server), memorizzandoli su server diversi e inserendo nel codice HTML della pagina di base le URI assolute del link e degli embedded item. Il content provider che ha deciso di utilizzare questo servizio identifica gli item del sito web che vorrebbe duplicare sulla rete associata (la CDN) per avere un più veloce accesso ai dati. Quindi, sceglie gli item con il costo maggiore (in termini di numero di richieste e/o in termini di dimensioni) e rimpiazza ogni riferimento a quegli item con un nuovo riferimento, che punta alla rete del servizio di content distribution. Quando un client,

---

uno degli origin server. *reverse proxy* e (origin) *server accelerator* possono entrambi essere più propriamente definiti surrogate.

generalmente un browser, si connetterà a quel sito, la pagina HTML richiesta sarà composta sia da item ospitati sullo stesso server che dagli item ospitati dalla CDN, tutto ciò in modo completamente trasparente all'utente. La CDN provvederà a creare e distribuire il più possibile le repliche dei contenuti. Kangasharju et al. in [27], utilizzando sia simulazioni sia sperimentazioni, mostrano come gli schemi di redirezione che richiedono ai client di ritrovare le differenti parti di una pagina web da differenti server, generano performance sub-ottimali rispetto a quando un client accede ad un unico server per tutti gli elementi di una pagina web. I risultati mostrati, quindi, che la redirezione completa del client verso una unica replica, dà migliori risultati rispetto ad una redirezione parziale. Infatti, l'uso delle connessioni persistenti (in parallelo per embedded object) favorisce l'utilizzo di un unico server.

3. La tecnologia Web Caching

3.1. Configurazioni di Web Caching

Nella progettazione di un servizio di web caching, esistono diverse configurazioni che possono essere utilizzate per far fluire le richieste HTTP attraverso una (o più) cache, e non direttamente verso l'origin server. Il punto cruciale è il modo con cui le richieste HTTP generate dai client arrivano al cache server, questa è una scelta di progettazione del servizio che dipende da vari fattori. Nel seguito proviamo a classificare le principali opzioni di configurazione, sulla base della collocazione del cache server all'interno della topologia della rete. Le possibilità sono sostanzialmente due:

1. Il server cache si trova naturalmente sul cammino dei dati (cioè delle richieste HTTP). In questa configurazione, oltre alle normali funzionalità di caching, il server deve analizzare il traffico in ingresso per filtrare (e redirigere) il traffico web inoltrando i pacchetti relativi ad altri protocolli direttamente in Internet (ciò vuol dire che deve avere funzionalità di routing). Questa opzione ha il vantaggio di non richiedere una configurazione dei browser, ma il ruolo del cache server diventa critico: se esso va giù, l'utente perde la connessione ad Internet (non solo per il web, ma per ogni servizio). In aggiunta, a differenza di router o switch, l'architettura di un cache server non è ottimizzata per analizzare grandi volumi di pacchetti di dati. Di conseguenza, il server può rallentare tutte i servizi Internet e, al tempo stesso, (sottraendo cicli di CPU alla sua funzione primaria) anche le prestazioni del servizio di cache. In [42], viene ulteriormente approfondito questo approccio, chiamato *transparent*

proxy-caching[23].

2. Il cache server non si trova sul cammino dei dati, quindi il traffico web deve essere indirizzato verso di lui. Questo può essere ottenuto in modo esplicito, via browser, o implicito con una redirezione del traffico HTTP verso il cache server, come descritto nel seguito:

   a) I browser sono configurati per inviare le loro richieste HTTP direttamente al cache server. Questo approccio è comunemente detto *proxy-caching*. Con questa opzione ovviamente non c'è alcun impatto sul traffico non-web, e inoltre l'amministrazione delle cache è più semplice perché è limitata alla sola configurazione di base. D'altra parte, uno dei principali problemi di questa soluzione è il carico amministrativo dovuto alla configurazione dei client (la somma totale del carico unitario, in medio-grandi organizzazioni può avere un impatto notevole). Comunque, nel corso degli ultimi anni, c'è stata una rapida evoluzione in questo settore che ha portato ad una notevole riduzione del carico sistemistico perché la configurazione può essere (quasi totalmente) automatizzata. Per ragioni di completezza descriviamo le tre modalità di configurazione possibili, anche se certamente la configurazione manuale è sconsigliabile.

      - **Manuale**. L'utente deve specificare per ogni protocollo (HTTP, FTP) il nome (o indirizzo IP) del server verso cui il browser deve inoltrare le richieste e la porta

[23] Anche se spesso le funzionalità di un cache e quelle di un proxy sono associate su un unico server, per ragioni di chiarezza è bene distinguere tra le funzioni dell'uno e dell'altro. Un proxy è un gateway posto tra la rete locale e Internet, allo scopo primario di controllare il traffico in uscita e/o in ingresso, ad esempio tramite autenticazione dell'utente, anche se esso può essere utilizzato per altri scopi, come per esempio per fornire agli utenti servizi aggiuntivi come conversioni di formato o dei filtri di anonimizzazione, etc. Una organizzazione che vuole controllare l'accesso ad Internet dei suoi utenti ad esempio, potrebbe decidere che tutte le richieste HTTP verso server esterni, per potere essere inoltrate in Internet, debbano passare attraverso il proxy. Questo, comunque, non implica che le risposte di ritorno saranno cacheate, potrebbero essere semplicemente scartate dopo l'inoltro.

su cui esso è attivo. Questa soluzione presenta forti svantaggi: non è scalabile, ma soprattutto è soggetta a fallimenti. Se il cache server riferito diviene non operativo, gli utenti non hanno alcun accesso web, fin quando non riconfigureranno il browser, specificando l'accesso diretto ad Internet, oppure la cache non tornerà in servizio.

- **Automatica** (Proxy-Auto Configuration). La configurazione automatica può essere attivata specificando (nella configurazione del browser) la URL di un file (con estensione *.pac*) contenente uno JavaScript che l'amministratore ha preparato e memorizzato su un server web. Lo script specifica la policy utilizzata dal browser per riferire uno o più proxy cache server [3]. Il server web deve essere correttamente configurato per associare il corretto MIME-type (application/x-ns-proxy-autoconfig) al file .pac, in modo che il client, quando lo scarica, lo riconosca come file di configurazione automatica e sia quindi in grado di utilizzarlo. Questa soluzione presenta maggiore scalabilità rispetto alla prima e soprattutto non è soggetta a fallimenti. Infatti, nel caso in cui il cache server non sia raggiungibile, il comportamento di default del browser è accedere direttamente all'origin server.
- **WPAD**[24]. Questa ultima soluzione tende ad automatizzare ulteriormente la configurazione automatica descritta nel punto precedente. WPAD (Web Proxy Auto-Discovery) [23] è un protocollo di resource auto-discovery (ricerca automatica di risorse), in questo caso la risorsa da individuare è il JavaScript. Una volta individuata la URL dello script, il client lo scaricherà in modo automatico. Il compito del processo WPAD è ricercare la URL corretta. Questo protocollo richiede l'interazione con meccanismi di resource discovery già esistenti quali DHCP (Dynamic Host Configuration Protocol) o DNS records

che vanno gestiti direttamente dalle organizzazioni. L'unica compito che rimane per l'utente è decidere se attivare o no il servizio, e questo può essere fatto con un semplice click in un check button del file di configurazione.

b) la redirezione del traffico web viene fatta con device dedicati a tale funzione come switch o router. Questa soluzione è chiamata transparent web caching, a sottolineare il fatto che il funzionamento del cache server è completamente trasparente agli utenti. A sua volta questa modalità di configurazione può essere implementata utilizzando due tecniche differenti:

   - Redirezione statica (via router);
   - Redirezione dinamica (con l'utilizzo di appositi protocolli, via router o switch). Questa seconda possibilità offre maggiori vantaggi rispetto alla redirezione statica.

Dato che questo lavoro di tesi si basa sulla sperimentazione di un'architettura di transparent web caching, questa ultima opzione di configurazione viene trattata approfonditamente nel prossimo paragrafo.

3.2. Transparent Web Caching

Il termine "transparent web caching" si riferisce alla tecnologia cache in cui il traffico web viene automaticamente intercettato e rediretto verso uno o più server e, viceversa. Una cache si definisce "trasparente" se i client possono accedervi senza la necessità di configurare il browser. La redirezione dei dati web può essere realizzata usando uno switch o router, come mostrato in Figura 3.1, e può essere effettuata:

- in modo statico, ma in tale configurazione, a parte la semplificazione della configurazione dei browser, non ci sono altri benefici,

[24] Ad oggi l'implementazione di questo meccanismo è disponibile solo su MS IE5.

- dinamicamente usando un protocollo di comunicazione tra device (router o switch) e cache server, in modo che possano conoscere reciprocamente il proprio stato. In questa seconda configurazione, è possibile ottenere una cooperazione "spinta", che offre notevoli benefici, descritti nel seguito.

Lo switch (o il router) intercetta, in maniera trasparente, il traffico web originato dai client e, applicando una politica di bilanciamento del carico, redireziona la richieste HTTP verso uno dei server cache disponibili. Se un server diviene irraggiungibile, lo switch ridistribuisce il traffico HTTP sui rimanenti server cache attivi.

Come si sa il protocollo HTTP è sempre in evoluzione e possono essere aggiunti nuovi metodi (estensioni del protocollo HTTP). Se le cache ricevono richieste contenenti metodi che non conoscono, li potrebbero scartare perché non in grado di interpretarli. Gli switch "intelligenti" sono in grado di mantenere il traffico HTTP di questo tipo al di fuori della cache, inoltrandolo direttamente in Internet. Per filtrare le richieste contenenti tali metodi (o per applicare tecniche di routing più sofisticate) gli switch devono essere capaci di "spiare" lo stream di dati in arrivo per il livello applicativo.

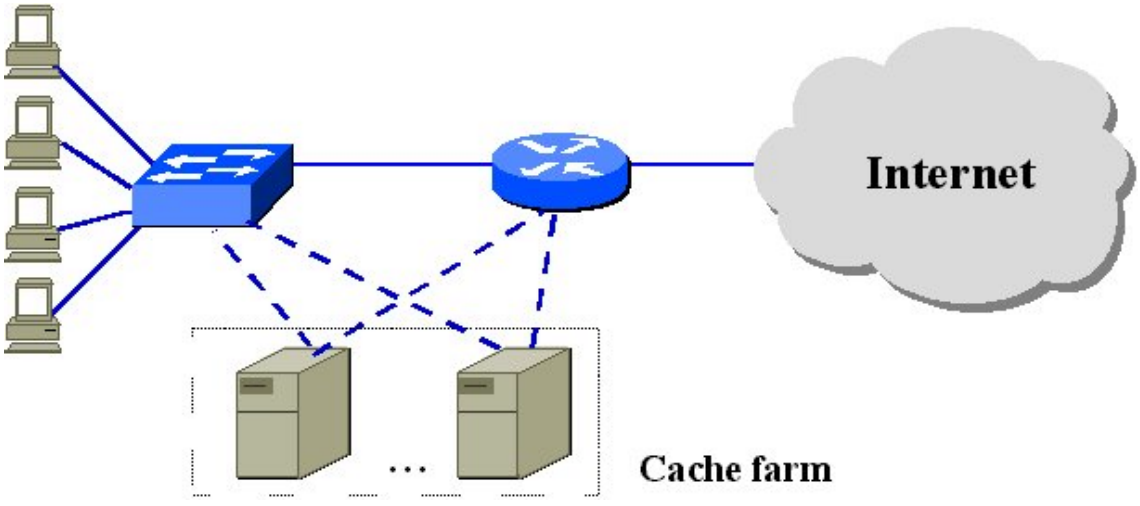

Figura 3.1 - Schema logico di un'architettura di Transparent Web Caching

La tecnologia di transparent web caching (con redirezione dinamica) offre molteplici

vantaggi.

- Non richiede alcuna configurazione dei client, riducendo i compiti amministrativi.
- Il traffico è automaticamente rediretto, gli utenti non possono evitare il servizio cache. I benefici quindi dell'utilizzo della tecnologia possono essere maggiori.
- Il servizio è sicuro (*fail-safe*), poiché lo switch o il router possono by-passare il server cache se questo dovesse essere non funzionante.
- L'architettura è scalabile. Più server possono essere aggiunti in modo del tutto trasparente.
- Si può applicare il bilanciamento del carico tra i cache server.
- I cache server possono continuare a cooperare facilmente con altri cache server in una gerarchia.

Questa tecnologia comunque presenta alcuni svantaggi:

- Può violare le direttive di cache del client [7]. L'utente può ricevere dati non conformi alla copia permanente residente sull'origin server, anche quando ha esplicitamente chiesto un reload della pagina. La mancanza di direttive per il cache server in una richiesta HTTP può generare inconsistenza dei dati. Dato che il client non è conscio della redirezione del traffico, ma pensa di parlare con il server originario, allo scopo di risparmiare banda, può non includere direttive critiche di "cache-control" (come "must-revalidate" o "no cache") nella richiesta HTTP[25]. L'omissione di queste direttive può far sì che il client riceva dati non sincronizzati con l'oggetto originario, anche se aveva richiesto la riconvalida dei dati. Questo problema è superabile se le future implementazioni dei client includeranno tutte le direttive specificate, anche a costo di trasmettere più byte.
- In modalità transparente le cache possono risultare leggermente più lente di quelle "non-trasparent", poiché i browser mantengono connessioni permanenti con le cache tradizionali, ma fanno cadere le connessioni con le transparent cache, appena il browser si collega ad un

differente sito web.

- Il maggiore difetto della trasparenza è la potenziale riduzione della robustezza della rete: viola il principio base di Internet, per cui la connettività della rete è un meccanismo *end-to-end* [18].

3.3. Architetture di cache server cooperanti

Le macchine che offrono un servizio di web caching possono essere organizzate in modo da cooperare, utilizzando appositi protocolli di comunicazioni inter-cache. Un cache server *stand-alone* (autonomo), infatti, offre minori vantaggi rispetto all'adozione di un insieme di più cache server che cooperano (cache peer, gerarchia di cache, cache farm) per la memorizzazione e il recupero di oggetti (internet object), sia in termini di prestazioni sia in termini di numero di oggetti trovati dentro le cache (hit rate), di scalabilità e di tolleranza ai guasti (fail-safe policy) [8]. Utilizzare una struttura di cache server cooperanti (*cache mesh*) consente di usufruire maggiormente dei vantaggi offerti della tecnologia di web caching:

- riduzione del traffico di rete (HTTP, FTP[26]). In relazione alla configurazione dei sistemi, alle caratteristiche degli oggetti (pagine statiche o dinamiche) e alla natura delle richieste provenienti dai vari browser, il traffico può essere considerevolmente ridotto;
- miglioramento della qualità del servizio (*QoS*) offerto all'utente, mediante la riduzione dei tempi di accesso alle informazioni;
- riduzione delle richieste inviate a web server molto "visitati", che risultano, pertanto, gravati da un minor numero di richieste da esaudire;
- maggior stabilità del servizio, poiché la realizzazione di una rete di cache server cooperanti evita situazioni che presentino un "single point of failure" per quanto

---

[25] § 3.8.2

[26] File Transfer Protocol: protocollo per il trasferimento di file nel modello TCP/IP.

concerne l'accesso a Internet;

- scalabilità della struttura di web caching mediante la definizione di nuovi livelli di gerarchia, l'inserimento di nuovi server o la riconfigurazione dei ruoli di cooperazione tra i *cache* server in risposta alle variazioni delle esigenze dell'utenza.

3.4. Protocolli

Web caching e web replication hanno assunto una importanza fondamentale, come dimostra la creazione, all'interno dell'IETF, del gruppo di lavoro WREC (Web Replication and Caching). Lo scopo del gruppo è studiare l'attuale sviluppo di web caching e web replication, sottomettere protocolli e standard non pubblicamente documentati all'IESG (Internet Engineering Steering Group) per la pubblicazione in forma di RFC (Request For Comments) e fornire raccomandazioni per l'eventuale sviluppo di nuovi protocolli.

Anche se la replica dei servizi web e il caching di risposte HTTP sono paradigmi distinti che richiedono soluzioni differenti, essi devono essere in grado di poter coesistere. Parte del lavoro del gruppo è definire le modalità con cui repliche e cache server possono essere automaticamente scoperti e utilizzati da client. Di recentissima costituzione è il gruppo Webi (Web Intermediaries), indirizzato allo studio di problemi specifici nella infrastruttura WWW, allo scopo di fornire meccanismi generici utili in più domini applicativi (proxy, content delivery surrogate, etc.).

Il gruppo WREC ha prodotto un documento [17] che fornisce una tassonomia dei correnti sviluppi di caching e replication e definisce la terminologia di riferimento. Sono stati, inoltre, pubblicati altri draft [14], [23], [29], [41]. Tutta la documentazione relativa alle attività del gruppo (draft, RFC, archivi della lista di discussione operante all'interno del gruppo, etc.) è accessibile sul sito web del gruppo, alla URL: http://www.wrec.org/.

Nel seguito saranno brevemente introdotti i protocolli di interazione tra cache. Per i dettagli di funzionamento dei protocolli e la definizione del formato delle informazioni, si rimanda alla

bibliografia allegata.

I protocolli di comunicazione tra cache (inter-cache communication protocol) determinano il tipo di architetura del mesh: piatta o gerarchica.

3.5. Architetture Gerarchiche

Più cache cooperanti sono logicamente strutturate in una gerarchia mediante la definizione di relazioni di peering tra cache "vicine" (neighbor). Esistono due tipi di relazioni: parent e sibling.

In una relazione di tipo *parent* viene definita una forma di subordinazione gerarchica tra *cache server* mentre in relazioni di tipo *sibling*, i server agiscono allo stesso livello.

Anche le funzionalità implementate sono differenti: i sibling, ricevuta una richiesta, restituiscono l'oggetto solo se esso è memorizzato nella cache e conserva ancora la sua validità, mentre, in caso di mancanza, non si preoccupano del recupero dell'oggetto stesso (fanno un servizio passivo).

In una relazione di tipo parent, invece, esiste una relazione gerarchica tra le cache: il *cache server parent*, nella eventualità che l'informazione richiestagli dal *cache server child* sia assente dalla cache, provvede a richiederla alle sue cache peer (peer può essere una cache parent oppure una cache sibling.) o, in mancanza di essi, direttamente al sito web originario (fa un servizio attivo).

In una rete di cache server cooperanti, può essere definito un ordinamento gerarchico tenendo conto che più di tre livelli di cache sono comunque sconsigliati: un oggetto ritrovato da un server d'origine o da una cache a livello più alto, usando cache intermedie, sarà memorizzato in tutte le cache usate per convogliare l'oggetto verso l'utente. Ciò significa che le cache a livello più alto necessitano di un'apprezzabile quantità di spazio su disco, altrimenti, ad esempio usando la tecnica LRU (Last Recently Used) potrebbero essere scartati degli oggetti prima che siano realmente divenuti "stale" (vecchi). Nella gerarchia, il livello più basso è vicino all'utente (cache locale) mentre i livelli superiori sono cache regionali o nazionali. I cache server di primo

livello (top level) interagiscono con i server web per il recupero delle pagine, se non già disponibili nella loro cache.

Nella definizione di una gerarchia di cache server cooperanti è importante tenere conto di alcuni fattori:

- più di tre livelli di cache non sono consigliati (a livello nazionale);
- i cache server di livello più alto (top cache) dovrebbero essere collocati "vicini" ai link nazionali/internazionali;
- nella definizione delle relazioni tra cache è necessario tenere conto delle politiche di routing (per ottimizzare i flussi dei dati e non creare colli di bottiglia);
- cache server di alto livello devono disporre di adeguate risorse al fine di poter fronteggiare l'elevato carico computazionale che deriva dal loro ruolo primario.

3.5.1. ICP

Internet Cache Protocol [29], [39], [40], è un protocollo di comunicazione utilizzato in architetture gerarchiche. È utilizzato per determinare la presenza di oggetti nelle cache vicine. Le cache, scambiandosi messaggi ICP, raccolgono informazioni che utilizzano per determinare la locazione più appropriata da cui recuperare un oggetto. Il protocollo ICP si basa sul protocollo UDP (User Datagram Protocol), che non garantisce l'affidabilità della comunicazione (UDP è un protocollo *connectionless*). La perdita di pacchetti ICP può, pertanto, essere utilizzata per fornire una stima della congestione della rete e sulla disponibilità dei server. Il problema di questo protocollo è la mancanza di informazioni su alcuni header HTTP associati ad un oggetto, che contengono informazioni vitali per l'accettazione o il rifiuto dell'oggetto stesso. Il protocollo, disponibile nella versione 2, è largamente implementato sia in prodotti free che commerciali.

3.5.2. HTCP

Hyper Text Caching Protocol [41] è un protocollo di comunicazione inter-cache nato dal desiderio di colmare le lacune di ICP. HTCP, a differenza di ICP, include nel pacchetto l'header HTTP che contiene informazioni importantissime per le cache: uno dei maggiori cambiamenti nel passaggio dal protocollo HTTP/1.0 a HTTP/1.1 è l'aggiunta del supporto per web caching. Gli header abilitano i fornitori delle informazioni a poter specificare direttive per le cache.

Utilizzando HTCP è, inoltre, possibile scoprire cache e dati in esse memorizzati, gestire insiemi di cache e monitorarne le attività. Attualmente il protocollo è supportato da un numero limitato di prodotti.

3.5.3. Cache Digest

Il meccanismo di Cache Digest [25] affronta il problema della latenza e della congestione associati ai protocolli ICP e HTCP. A differenza di tali protocolli, il Cache Digest supporta peering tra cache server senza richiedere uno scambio richiesta-risposta, consentendo ai peer di mantenere un sommario (Digest) del contenuto delle cache con cui essi hanno relazioni. Questi Digest sono calcolati internamente dai cache server e possono esistere come oggetti "cacheati", soggetti a regole di refresh e di scadenza come ogni altro oggetto memorizzato nelle cache. I server, ad intervalli regolari, si scambiano i Cache Digest. Utilizzando un Cache Digest è possibile determinare con grado di accuratezza relativamente alto se l'oggetto corrispondente a una certa URL è memorizzato all'interno di un particolare cache server. Questa funzionalità è ottenuta dando in input ad una funzione hash la URL richiesta e il metodo HTTP corrispondente, e confrontando il risultato ottenuto con i contenuti dei Cache Digest.

Questa tecnica consente quindi di rendere disponibile in formato compatto un sommario del contenuto di un server riducendo fortemente le comunicazioni tra cache.

3.6. Architetture piatte

Cache farm o cache array sono esempi di organizzazione di cache non gerarchiche. In questo caso lo spazio delle URL viene suddiviso tra più cache in base ad algoritmi di distribuzione (funzioni hash) in modo da ottimizzare le prestazioni del sistema e pertanto le richieste originate dai client vengono ripartite in modo uniforme tra più macchine, evitando di duplicare gli oggetti all'interno di più server. In questo modo viene anche ottimizzato l'utilizzo delle risorse di memoria e disco.

3.6.1. CARP

Cache Array Routing Protocol [36] consente di realizzare un cluster di proxy cache. Utilizzando una funzione hash, CARP consente di dividere lo spazio delle URL tra un insieme di cache. Il protocollo include la definizione di una Proxy Array Membership Table, e le regole per scaricare queste informazioni.

Un client HTTP che implementa CARP può, quindi, instradare in modo bilanciato le richieste di URL a tutti i membri del proxy array e, utilizzando l'ordinamento e la suddivisione delle richieste, viene eliminata (o ridotta) la duplicazione dei contenuti delle cache, con un possibile miglioramento nella velocità dei cache hit. CARP non è un protocollo di comunicazione, strettamente parlando, infatti il suo utilizzo permette di evitare le comunicazioni inter-cache. I client possono utilizzare CARP direttamente come funzione hash basata sul meccanismo di selezione dei proxy.

3.6.2. WCCP

Come già visto nei paragrafi precedenti, l'evoluzione della tecnologia di web caching consente oggi di realizzare soluzioni scalabili e affidabili in modo completamente trasparente all'utente (*transparent caching*). Il traffico web viene automaticamente intercettato e rediretto verso una o più web cache. L'utente non deve più configurare il suo browser per l'utilizzo di un

proxy web cache, né in maniera manuale, né in modalità automatica utilizzando un proxy auto-configuration file.

Le funzionalità di *transparent caching* possono essere ottenute a livello di switch o a livello di router. Il protocollo WCCP v.1 (Web Cache Coordination Protocol) [14] è utilizzato per associare un singolo router con una o più web cache (*cache farm*) allo scopo di redirezionare in modo trasparente il traffico HTTP. Il protocollo permette ad una delle cache (designated web cache) di decidere come il router debba distribuire il traffico rediretto tra le web cache associate, per avere un bilanciamento del carico.

Il router, analizzando i pacchetti che lo attraversano, reindirizza quelli diretti alla porta 80 verso una delle cache associate. Il protocollo permette al router di determinare la disponibilità delle cache, redirezionando le richieste verso una nuova cache aggiunta alla cache farm, e, in caso di guasto di uno dei cache server, il router provvede automaticamente a disattivare l'inoltro delle richieste verso tale macchina utilizzando le cache rimanenti o, nel caso di un singolo cache server, inviando i dati direttamente verso il sito WWW (*fail-safe policy*).

La comunicazione tra cache rimane invariata: esse possono continuare ad interagire utilizzando protocolli standard quali ICP e HTCP.

La versione 2.0 del protocollo, definito dalla CISCO, è stata resa di pubblico dominio in forma di draft [15], in modo da consentire il supporto del WCCP anche in prodotti freeware o di altre aziende, maggiori dettagli di funzionamento saranno descritti più avanti, nel § 4.5.1.

3.7. Protocolli Multicast

I protocolli basati su trasmissioni multicast non sono molto diffusi anche perché richiedono l'utilizzo di reti che supportino tali servizi (come M-BONE). A questa motivazione si aggiungono ulteriori considerazioni: le infrastrutture, le configurazioni, i tunnel multicast sono spesso instabili con possibile perdita di connettività; l'utilizzo di trasmissione multicast riduce il numero di ICP query, ma non il numero di risposte perciò i benefici sono relativi; dato che non

ci sono particolari requisiti per unirsi ad un gruppo multicast, l'utilizzo di trasmissioni multicast può esporre la cache a problemi di privacy: chiunque può unirsi a un gruppo e cercare di intercettare i pacchetti ICP [8].

Il protocollo ICP implementa il supporto per reti multicast, anche se questa funzionalità è ad oggi scarsamente utilizzata.

Nell'ambito del progetto LSAM (Large-Scale Active Middleware) è stata sviluppata una estensione del proxy cache Apache (LSAM Proxy Cache) che usa push multicast di pagine correlate, basate su gruppi di interesse automaticamente selezionati, in modo da caricare le cache nei loro naturali punti di aggregazione di rete (http://www.isi.edu/~lsam/).

Lo scopo del progetto Adaptive Web Caching (AWC) è invece progettare e implementare protocolli per supportare un sistema di web caching globalmente distribuito, auto-configurante e altamente adattativo (maggiori informazioni sono disponibili alla URL http://irl.cs.ucla.edu/AWC/).

Infine, il protocollo WCCP [15] v.2 ha implementato un supporto per il multicast, per il quale si rimanda al § 4.4.1.

3.8. Il Caching nel Protocollo HTTP/1.1

Il protocollo HTTP [9] [22] non era stato progettato originariamente per includere sistemi di caching nella catena richieste/risposte client-server ed è stato, quindi, esteso per includerne i meccanismi di base. Il fine del caching è eliminare la necessità di spedire richieste attraverso la rete, riducendo il numero di round-trip richiesti per molte operazioni (con un meccanismo di "expiration", cioè scadenza), o la necessità di inviare risposte complete, riducendo la richiesta di banda (con un meccanismo di *validation*, cioè validazione). Per questi motivi, la versione 1.1 del protocollo include un numero di elementi intesi a supportare il caching, come la data di

scadenza (expiration time[27]) o i *validator*[28] specificati dal server, che sono meccanismi per la validazione degli oggetti, aventi lo scopo di ridurre il traffico non necessario.

3.8.1. Meccanismi di Controllo della Consistenza

3.8.1.1. Server-Specified Expiration

Il caching è tanto più efficace quanto più le cache possono evitare di fare richieste all'origin server. Il principale meccanismo usato per ottenere questo risultato è un expiration-time esplicito specificato dall'origin server (nel campo Expires dell'entity-header), che indica per quanto tempo un oggetto può essere utilizzato per soddisfare le richieste successive a quella che ne ha causato l'invio. In altre parole, una cache può restituire un oggetto fresh (quindi valido), in base all'expiration-time, senza dover prima contattare il server. I server dovrebbero, naturalmente, assegnare i futuri expiration-time agli oggetti inviati, nella convinzione che probabilmente l'entity[29] non cambierà in modo significativo prima della scadenza assegnata. Questo meccanismo si applica soltanto alle risposte già presenti in una cache e non a quelle *first-hand*[30], spedite direttamente al client che ha fatto la richiesta.

Il campo Expires fornisce la data/tempo dopo il quale la risposta deve essere considerata stale, e quindi deve essere rivalidata con l'origin server (o con una cache intermedia che possiede una copia fresh dell'entità). Un esempio è:

```
Expires: Thu, 06 Feb 2001 16:00:00 GMT
```

---

[27] L'explicit expiration time (scadenza esplicita) è l'istante dopo il quale un oggetto non dovrebbe essere restituito da un cache server senza una ulteriore validazione con l'origin server.

[28] Un elemento del protocollo, ad esempio un entity tag o un Last-Modified time, utilizzato per scoprire se una entry della cache è una copia equivalente dell'entity.

[29] Entity (o entità) e oggetto sono qui considerati come sinonimi, dato che non c'è ancora uno standard riconosciuto per molti dei termini usati.

[30] Una risposta è detta *first-hand* se proviene direttamente dall'origin server o se la sua validità è stata appena controllata direttamente con l'origin server.

3.8.1.2. Heuristic Expiration

Poiché gli origin server non specificano sempre esplicitamente gli expiration time, le cache hanno la possibilità di assegnare un *heuristic expiration time*, impiegando un algoritmo che usa altri valori, come il campo Last-Modified dell'entity-header (l'istante in cui l'oggetto è stato modificato per l'ultima volta), per stimare un expiration time plausibile. Le specifiche del protocollo HTTP/1.1 non danno un algoritmo particolare, ma impongono dei limiti ai risultati ottenuti nel caso peggiore.

3.8.1.3. Calcolo dell'Età

Per sapere se un oggetto memorizzato è fresh, una cache deve sapere se l'età[31] dell'oggetto eccede il suo *freshness lifetime*[32]. Il protocollo richiede agli origin server di includere nella risposta un campo Date del general-header, che dà il momento in cui la risposta è stata generata. Il valore del campo Age è la stima effettuata dalla cache della quantità di tempo passato dal momento in cui la risposta è stata generata dall'origin server o convalidata con successo. In sostanza, il valore Age è la somma del tempo in cui la risposta è stata in ognuna delle cache lungo il path dall'origin server, più il tempo totale di transito lungo il cammino di rete. I valori di questo campo sono espressi in interi decimali non negativi, che rappresentano il tempo in secondi, e sono utilizzati per verificare che una risposta "cacheata" sia fresh. Una risposta è fresca se la sua età (il valore del campo Age) non eccede il valore del freshness lifetime. L'età di una risposta può essere calcolata utilizzando il valore del campo Age, se tutte le cache lungo il percorso implementano l'HTTP/1.1, oppure in base al valore del campo Date, se l'orologio locale è sincronizzato con l'orologio dell'origin server.

---

[31] L'Age (età) di una risposta è il tempo trascorso da quando è stata spedita o convalidata con successo dall'origin server.

[32] Il freshness lifetime (tempo di vita) è l'intervallo di tempo tra la generazione di una risposta e la sua expiration.

3.8.1.4. Calcolo della Scadenza

Per sapere se un oggetto "cacheato" è fresh o *stale*, bisogna confrontare il suo freshness lifetime con la sua età (calcolata come indicato nella sezione precedente), se è maggiore, l'oggetto è ancora utilizzabile senza dover essere convalidato.

3.8.1.5. Modello di Validazione

Se una cache ha una entry stale, prima di poterla mandare in risposta alla richiesta di un client, deve validarla con l'origin server (o con una cache intermedia che possiede una risposta fresh), per verificare che sia ancora utilizzabile. Esegue, cioè, una *validazione* dell'oggetto, evitando, però, di pagare l'overhead della ritrasmissione dell'intera risposta, se l'oggetto "cacheato" è valido o di una sequenza richiesta-risposta in più se l'oggetto va riconvalidato, tramite l'uso dei metodi condizionali supportati dal protocollo[33].

I cache validator sono meccanismi per supportare i metodi condizionali. Quando un origin server genera una risposta completa, genera con essa una sorta di validatore, che sarà mantenuto insieme all'oggetto "cacheato". Quando un client (uno user agent o una cache) fa una richiesta condizionale, include nella richiesta anche il validator associato alla risorsa. Quindi, il server confronta questo validator con il validatore corrente dell'entity e, se corrispondono, risponde con un particolare codice di stato (usualmente 304, Not Modified), senza restituire l'entity-body. Se non corrispondono, restituisce l'intera risposta. In questo modo, evita di trasmettere interamente la risposta se questa è ancora valida e, contemporaneamente, di avere un round-trip in più se i due validator non corrispondono. Nell'HTTP/1.1, una richiesta condizionale appare esattamente come una richiesta normale per la stessa risorsa, eccetto che include uno speciale campo dell'entity-header, contenente il validator, e converte il metodo (GET, in genere) in un metodo condizionale[34].

Come cache validator viene spesso utilizzato il campo Last-Modified dell'entity-header; in

---

[33] § 2.2

questo caso, una entry è considerata valida se l'entity non è più stata modificata dal momento indicato dal valore del campo Last-Modified.

Un altro cache validator è dato dal valore del campo ETag del response-header, un entity tag[35] che permette una convalida in situazioni in cui non è conveniente memorizzare o utilizzare la data di modifica.

3.8.1.6. Weak e Strong Validator

Un validator, associato ad una risorsa, è detto *strong* (forte) se cambia quando l'entity cambia in un qualsiasi modo (contenuto del body campi degli header). Un validator che non cambia ad ogni cambiamento della risorsa è detto *weak* (debole). Si può pensare che un validator sia strong se cambia ogni volta che cambiano i bit dell'entity e sia weak se cambia quando cambia il significato dell'entity. Un esempio di weak validator può essere dato dal tempo di modifica dell'entity, se rappresentato con una risoluzione di un secondo. Poiché è possibile che la risorsa sia modificata più volte in un singolo secondo, il valore Last-Modified è considerato implicitamente weak. I weak validator sono utilizzabili soltanto quando il contesto non dipende dall'esatta uguaglianza delle entity, mentre gli strong validator sono utilizzabili in ogni contesto. Ad esempio, entrambi i validator possono essere utilizzati per una GET condizionale di una intera entity, ma per la ricerca di un suo sottoinsieme può essere usato solo uno strong validator, altrimenti il client potrebbe ottenere un oggetto inconsistente. Gli entity tag sono, normalmente, strong validator.

---

[34] § 2.2

[35] Un entity tag è usato per confrontare due o più entity della stessa risorsa richiesta. Consiste di una stringa posta tra virgolette, che può essere preceduta da un indicatore di weakness (debolezza).

entity-tag = [weak] opaque-tag
weak = "W/"
opaque-tag = quoted-string

3.8.1.7. "Cacheabilità" delle Risposte

A meno di particolari restrizioni dovute a direttive di cache-control[36], un sistema di caching può salvare una risposta in arrivo, può spedirla come risposta senza doverla convalidare (se è ancora fresh) e può utilizzarla successivamente dopo una convalida avvenuta con successo. Se non sono associati alla risposta né un cache validator né un esplicito expiration time, questa non dovrebbe essere "cacheata". Alcune cache possono, però, memorizzare questo tipo di risposte, per utilizzarle, ad esempio, quando la rete non è connessa; un client può controllare una risposta di questo tipo confrontando il campo Date con il tempo corrente. Esistono dei contesti in cui non è conveniente che una cache memorizzi un certo oggetto, o lo restituisca in risposta ad una richiesta per considerazioni di sicurezza, ad esempio, o di privacy. Il protocollo fornisce delle direttive di cache-control che permettono ad un server di indicare che una risorsa, o parte di essa, non deve essere "cacheata".

3.8.1.8. Interpretazione delle Risposte

Utilizzando i meccanismi di validazione, introdotti nei paragrafi precedenti, la cache deve essere capace di combinare opportunamente le nuove informazioni ricevute con quelle già possedute. Quando una cache fa una richiesta di convalida ad un server e quest'ultimo dà una risposta del tipo Not-Modified (codice 304) o Partial-Content (206, data in risposta ad una partial GET[37]), la cache costruisce la risposta da spedire al client sulla base dell'entry memorizzata. Se il codice di stato è 304, la cache usa l'entity-body posseduto come entity-body della risposta che sta inviando. Se il codice è 206 e i campi Etag o Last-Modified corrispondono, la cache può combinare il contenuto conservato nella propria entry con il contenuto ricevuto all'interno della risposta ed utilizzare il risultato come entity-body della risposta da inviare.

3.8.1.9. Invalidazione dopo Aggiornamenti o Cancellazioni

L'effetto di modifiche eseguite su una risorsa (presso l'origin server) potrebbe essere

---

[36] descritte in § 3.8.2

[37] § 2.2

l'invalidazione non trasparente di una o più entity "cacheate". Questo vuol dire che, anche se tali entry continuano ad essere fresh, esse non riflettono più la risposta che l'origin server darebbe ad una successiva richiesta su quella risorsa. Da qui nasce il problema di risposte stale. Con il protocollo HTTP, non c'è modo di garantire che tutte queste entry siano effettivamente invalidate. Ad esempio, non è detto che la richiesta che ha causato il cambiamento sull'origin server sia arrivata dallo stesso proxy in cui è conservata l'entry della cache. La frase "invalidare una entity" significa che la cache dovrà o rimuovere tutte le istanze di quell'entity dalla propria memoria o dovrà segnarle come "invalide" e riconvalidarle prima di spedirle in risposta ad una richiesta seguente. A questo proposito si rimanda al protocollo WCIP[38].

3.8.1.10. Rimpiazzamenti nella Cache

Se una cache riceve una nuova risposta "cacheabile" mentre esistono altre risposte relative alla stessa risorsa in memoria, la nuova risposta dovrebbe essere utilizzata per rispondere alla richiesta corrente e memorizzata per le richieste future. Una nuova risposta con un campo Date più vecchio delle risposte già "cacheate", è considerata non "cacheabile". In questa descrizione, gli algoritmi di rimpiazzamento utilizzati dalle cache non saranno trattati, perché non rilevanti ai fini di questo lavoro di tesi. Per approfondimenti, si rimanda agli atti del WCW'99 (Web Caching Workshop) e del 5th International Web Caching and Content Delivery Network, disponibili on-line all'indirizzo http://www.terena.nl/conf/wcw/

3.8.2. Meccanismi di Controllo delle cache

I meccanismi di base dell'HTTP/1.1 (scadenza e validazione) sono direttive implicite per le cache. In alcuni casi, sia il server sia il client potrebbero volere fornire direttive esplicite. A questo scopo, viene utilizzato l'header Cache-Control, che permette ad un client o ad un server di trasmettere direttive riguardanti sia le richieste sia le risposte. Queste direttive, tipicamente, hanno la precedenza su quelle risultanti dagli algoritmi di caching e includono restrizioni su

[38] § 3.8.4

cosa è "cacheabile", restrizioni su cosa può essere memorizzato da una cache imposte dal server di origine, modifiche del meccanismo di scadenza di base, controlli sulla riconvalida e il reload degli oggetti contenuti nella cache, imposti da uno user agent (il client che inizia una connessione). Come regola generale, se c'è un conflitto tra i valori degli header, viene applicata l'interpretazione più restrittiva.

Il campo Cache-Control del general-header (utilizzabile sia in richieste sia in risposte) è usato per specificare direttive che devono essere rispettate da tutti i meccanismi di caching, lungo la catena richiesta-risposta, e che specificano un comportamento inteso a prevenire interferenze negative da parte delle cache. Le direttive di cache sono unidirezionali, cioè la presenza di una direttiva in una richiesta non implica che la stessa direttiva sia presente anche nella risposta. Queste direttive possono essere:

Direttive di richiesta: no-cache, no-store, max-age, max-stale, min-fresh, no-transform, only-if-cached, cache-extention.

Direttive di risposta: public, private, no-cache, no-store, no-transform, must-revalidate, proxy-revalidate, max-age, s-maxage, cache-extention.

Si possono, inoltre, suddividere le direttive di cache-control in alcune categorie generali:

Restrizioni su cos'è "cacheabile", possono essere imposte solo dall'origin server. Per default, una risposta è "cacheabile" se è così indicato dai requisiti del metodo della richiesta, dai campi dell'header della richiesta e dal codice di stato della risposta. Le direttive che permettono ad un origin server di sovrascrivere la "cacheabilità" di default di una risposta sono:

- public: indica che la risposta può essere "cacheata" da qualsiasi cache.
- private: indica che (tutto o in parte) il messaggio di risposta è inteso per un singolo utente e non deve essere "cacheato" da una cache condivisa.
- no-cache: se è presente questa direttiva una cache non deve usare la risposta per soddisfare una richiesta seguente, senza averla riconvalidata (con successo) con l'origin server. Permette all'origin server di prevenire il caching, anche in cache configurate per

restituire risposte stale alle richieste dei client.

Restrizioni su cosa può essere memorizzato da una cache, possono essere imposte sia dall'origin server sia dallo user agent, tramite la direttiva no-store. Lo scopo di questa direttiva è prevenire la diffusione involontaria o la conservazione di informazioni sensibili. Si applica all'intero messaggio e può essere spedita sia in una richiesta sia in una risposta. L'uso di questa direttiva in molti casi può migliorare la privacy, anche se non è sempre un meccanismo sufficiente o affidabile per assicurare la privacy; in particolare, alcune cache potrebbero non riconoscere tale direttiva o non seguirla oppure le reti di comunicazione potrebbero essere vulnerabili all'intercettazione delle comunicazioni. In tal caso, è meglio utilizzare un canale di comunicazione cifrato[39].

Modifiche sul meccanismo di scadenza di base, possono essere imposte sia dall'origin server sia dallo user agent. Il momento della scadenza di una entity può essere specificato dall'origin server, tramite il campo Expires dell'entity-header, oppure inserendo la direttiva max-age in una risposta. Quando è presente questa direttiva, la risposta è stale se la sua età corrente è maggiore del valore di età dato (espresso in secondi) nel momento in cui arriva una nuova richiesta per la stessa risorsa. Se in una risposta è presente una direttiva max-age, la risposta è "cacheabile" a meno che non siano presenti altre direttive più restrittive. Se sono presenti sia la direttiva max-age sia il campo Expires, quest'ultimo, anche se è più restrittivo, viene sovrascritto dalla direttiva max-age.

s-maxage: se una risposta include questa direttiva, per una cache condivisa, l'età massima qui specificata sovrascrive i valori specificati dalla direttiva max-age o dal campo Expires. Implica anche che una cache condivisa non deve utilizzare una entry divenuta stale, senza prima averla riconvalidata con l'origin server.

Le altre direttive (min-fresh, max-stale) permettono allo user agent di modificare il

[39] Per approfondimenti, si rimanda agli RFC 2246, 2817, 2818.

meccanismo di scadenza di base, ponendo un limite all'età di una risposta o permettendo di accettare una risposta stale.

I controlli su reload e rivalidazione della cache possono essere imposti solo dallo user agent, nel caso in cui voglia rivalidare l'entry di una cache direttamente con l'origin server (e non con la cache successiva lungo il path verso l'origin server) oppure desideri fare il reload dall'origin server. La rivalidazione end-to-end può essere necessaria se la cache o l'origin server hanno sovrastimato l'expiration time della risposta "cacheata"; il reload end-to-end può essere necessario se l'entry è divenuta corrotta.

- Reload end-to-end: la richiesta include una direttiva no-cache oppure, per compatibilità con i client HTTP/1.0, una direttiva Pragma: no-cache. Quando è presente la direttiva Pragma: no-cache in un messaggio di richiesta, l'applicazione deve spedire la richiesta verso l'origin server anche se ha una copia "cacheata" di ciò che sta richiedendo, no-cache richiede una copia autoritativa della risorsa. Le cache HTTP/1.1 trattano la direttiva Pragma: no-cache come se il client avesse spedito una Cache-Control: no-cache. Questa direttiva di configurazione permette al client di rinfrescare la copia memorizzata nella cache divenuta stale o inconsistente. Il campo Pragma in un header è impiegato per includere direttive specifiche di implementazione da applicare a tutti i destinatari lungo la catena richiesta/risposta. Tutte le direttive Pragma specificano un comportamento opzionale dal punto di vista del protocollo.
- Rivalidazione end-to-end: quando il client ha già una copia locale "cacheata", la rivalidazione è detta end-to-end specifica. La richiesta, in questo caso, include una direttiva max-age=0 che forza ogni cache, lungo il path verso l'origin server, a rivalidare la sua entry, se la possiede, con la cache successiva o con il server; la richiesta iniziale include una condizione di cache-validating con il validator corrente posseduto dal client. Se il client non ha la sua copia locale dell'entry da rivalidare, la rivalidazione è detta end-to-end aspecifica; come prima la richiesta contiene una direttiva max-age=0; la

richiesta iniziale non include una condizione cache-validating: è la prima cache che conserva una entry per la risorsa specificata, lungo il cammino, ad inserirla includendo il suo validator corrente.

In alcuni casi, come quando la connessione della rete è estremamente scadente, un client può esigere che una cache restituisca solo le risposte correntemente memorizzate, senza farne il reload o riconvalidarle con l'origin server. Per ottenere questo risultato, il client può includere nella richiesta la direttiva only-if-cached. Una cache che riceve questa direttiva può rispondere utilizzando una entry "cacheata" che sia comunque consistente con altre restrizioni della stessa richiesta.

Poiché una cache può essere configurata per ignorare l'expiration time specificato dal server e la richiesta di un client può includere una direttiva max-stale (che ha un effetto simile), il protocollo include un meccanismo che permette all'origin server di richiedere la rivalidazione di una entry "cacheata" prima di ogni utilizzo successivo. Se, in una risposta ricevuta da una cache, è presente una direttiva must-revalidate, quella cache non può utilizzare l'entry (dopo che è divenuta stale) per rispondere a richieste successive, senza averla prima riconvalidata con l'origin server. In altre parole, se la risposta "cacheata" è stale in base al valore max-age o al valore Expires dato dall'origin server, la cache deve fare una rivalidazione end-to-end ogni volta.

La direttiva proxy-revalidate ha il medesimo significato della direttiva must-revalidate, eccetto che non si applica a cache non condivise da user agent. Può essere utilizzata come risposta ad una richiesta autenticata, per permettere alla cache dell'utente di salvare e, più tardi, restituire la risposta senza doverla rivalidare (poiché è già stata autenticata una volta da quell'utente), mentre i proxy che servono molti utenti devono continuare a rivalidarla ogni volta (per assicurarsi che ogni utente sia autenticato). Da notare che le risposte autenticate di questo tipo devono avere anche la direttiva cache-control public, per far sì che siano "cacheabili".

Controlli sulla trasformazione delle entity. Per una cache intermedia o un proxy, può essere

utile convertire il tipo di alcuni entity-body; ad esempio, convertire una immagine da un formato ad un altro, per salvare spazio disco. Se un messaggio include la direttiva no-tranform, la cache o il proxy non devono cambiare alcuni campi dell'entity, come content-encoding[40] o content-type[41], questo implica che non possono neanche cambiare alcun aspetto dell'entity-body legato a questi campi.

Estensioni ai sistemi di cache. I campo Cache-Control può essere esteso con l'uso di uno o più token di cache-extension, ognuno con un valore opzionale assegnato. Possono essere aggiunte estensioni informative che non cambiano la semantica di altre direttive, mentre le estensioni comportamentali sono state progettate per lavorare agendo come modificatori delle direttive di cache già esistenti. Le direttive standard e le estensioni sono implementate in modo tale che, le applicazioni che non sono in grado di interpretarle seguiranno di default il comportamento specificato dalle direttive standard, le applicazioni che comprendono le nuove direttive riconosceranno, invece, le modifiche associate. In questo modo, possono essere realizzate estensioni alle direttive Cache-Control senza dover modificare il protocollo di base, e, contemporaneamente, si permette ad una cache di continuare a seguire le direttive cache-control definite per la sua versione nativa del protocollo, ignorando tutte le direttive che essa non comprende. La specifica HTTP/1.1 permette, così, ad una cache di agire in "tunnel mode" quando riceve una richiesta che include un metodo che essa non sa come gestire. Il protocollo è reso, così, estensibile.

3.8.3. Meccanismi di Validazione

I campi If-Match, If-Modified-Since, If-None-Match, If-Range, If-Unmodified-Since sono utilizzati per rendere condizionale un metodo:

- If-Match, un client che possiede una o più entity, precedentemente ottenute da una

[40] usato come modificatore per i media-type.

[41] Indica il media-type dell'entity-body spedito al destinatario.

risorsa, può verificare se una di esse è aggiornata includendo la lista degli entity tag ad essa associati nel campo If-Match del request-header. Lo scopo di questa feature è permettere aggiornamenti efficienti delle informazioni "cacheate" con il minimo overhead di transazione.

- If-Modified-Since, se la variante[42] richiesta non è stata modificata dall'istante specificato in questo campo, il server restituirà una risposta non-modified (304) senza includere il message-body. Questa feature ha la finalità di consentire aggiornamenti efficienti delle informazioni "cacheate" con la quantità minima di overhead di transazione.
- If-None-Match, un client, che possiede una o più entity ottenute da una risorsa, può verificare che nessuna di queste entity è aggiornata, includendo una lista dei loro entity-tag nel campo If-None-Match dell'header.
- Se un client ha una copia parziale di una entity nella sua cache e vorrebbe avere una copia aggiornata dell'intera entity, potrebbe utilizzare il campo Range dell'header della richiesta con una GET condizionale (usando, ad esempio, If-Match). Ma, se la condizione fallisce perché l'entity è stata modificata, il client dovrebbe fare una seconda richiesta per ottenere l'intero entity-body corrente. Il campo If-Range permette ad un client di evitare la seconda richiesta: se l'entity non è cambiata, viene spedita la parte mancante, altrimenti viene spedita interamente. Le richieste di recupero HTTP eseguite con metodi GET condizionali possono richiedere uno o più sub-range di una entity, invece dell'entity intera, utilizzando il campo Range, che si applica all'entity restituita come risultato della richiesta.
- If-Unmodified-Since, se la risorsa non è stata modificata dall'istante specificato, il server dovrebbe eseguire l'operazione richiesta come se questo campo non fosse presente. Se la variante, invece, è stata modificata dopo l'istante specificato, il server non deve eseguire

[42] Una risorsa può avere una o più rappresentazioni, dette varianti.

l'operazione richiesta e deve restituire un codice 412 (Precondition Failed).

- Last-Modified: indica la data e il momento in cui l'origin server presume che la variante sia stata modificata per l'ultima volta. Il significato esatto di questo campo dipende dall'implementazione dell'origin server e dalla natura della risorsa. Per i file, ad esempio, può essere soltanto l'istante di ultima modifica del file system. Per entità che includono parti dinamiche, può essere il più recente di un insieme di istanti di ultima modifica delle componenti.

Per facilitare la comprensione dei meccanismi descritti, inseriamo a titolo d'esempio uno schema dell'algoritmo con cui il proxy cache server Squid[43] calcola la freshness degli oggetti e ne richiede la validazione. I valori dei campi specificati in seguito sono utilizzati per calcolare alcuni valori necessari per il controllo.

- OBJ_DATE è l'istante in cui l'oggetto è stato spedito dall'origin server (derivato dal campo Date del response-header).
- OBJ_LASTMOD è l'istante di ultima modifica dell'oggetto (derivato dal campo Last-Modified del response-header).
- OBJ_AGE indica l'età dell'oggetto rispetto al momento in cui è stato ricevuto ed è dato da:
- OBJ_AGE = NOW - OBJ_DATE (dove NOW è l'istante in cui si sta facendo il calcolo).
- LM_AGE indica quanto era "vecchio" l'oggetto quando è stato ritrovato:
- LM_AGE = OBJ_DATE - OBJ_LASTMOD
- LM_FACTOR = OBJ_AGE / LM_AGE
- CLIENT_MAX_AGE è l'età massima (opzionale) entro cui il client accetta l'oggetto, ricavato dal campo Cache-Control dell'header della richiesta HTTP/1.1.

[43] di pubblico dominio

- EXPIRES è l'explicit expiration time (opzionale), ricavato dal campo Expires dell'header di risposta del server.

Questi valori sono confrontati con i seguenti parametri di refresh:

- CONF_MIN: il tempo (in minuti) entro il quale un oggetto senza un explicit expiration time dovrebbe essere considerato fresh. Il valore raccomandato è zero, ogni valore più alto può essere far sì che oggetti non "cacheabili", come ad esempio applicazioni dinamiche, siano erroneamente "cacheati".
- CONF_PERCENT: una percentuale dell'age, entro cui un oggetto senza explicit expiration time sarà considerato fresh.
- CONF_MAX: un limite superiore su quanto a lungo gli oggetti senza explicit expiration time saranno considerati fresh.

Nel seguito viene schematizzato l'algoritmo:

```
if (EXPIRES è presente ed è <= NOW)
     return STALE
 else
    return FRESH
if (CLIENT_MAX_AGE è presente ed è < OBJ_AGE)
    return STALE

 if (OBJ_AGE > CONF_MAX)
   return STALE
 if (OBJ_DATE > OBJ_LASTMOD)
{
 if (LM_FACTOR < CONF_PERCENT)
   return FRESH
 else
   return STALE
 }
 if (OBJ_AGE <= CONF_MIN)
    return FRESH
 return STALE
```

3.8.4. WCIP: Web Cache Invalidation Protocol

Se un contenuto web non è soltanto popolare, ma anche soggetto a modifiche frequenti (si dice che ha contenuto dinamico), mantenere la consistenza della copia salvata all'interno della cache diviene abbastanza difficile. D'altra parte, i contenuti dinamici sono divenuti velocemente

una percentuale significativa del traffico web, basta considerare le news, le quotazioni di borsa o i prezzi di un catalogo di shopping, etc. Poiché il contenuto cambia velocemente, la cache deve fare frequenti richieste all'origin server per controllarne la freshness e, nonostante questo, potrà rispondere agli utenti con dati stale. Per risolvere questo problema, il content provider setta un *expiration time* molto breve oppure marca questi contenuti dinamici come non-"cacheabili", anche quando potrebbero essere "cacheati" (almeno per un certo intervallo di tempo). Il protocollo WCIP (Web Cache Invalidation Protocol) cerca di risolvere questo problema, in modo da fornire una freshness garantita per in certo intervallo, utilizzando meccanismi di invalidazione e di aggiornamento per mantenere *fresh* gli oggetti "cacheati". Usando il WCIP, un server web può segnalare aggiornamenti e/o invalidazioni degli oggetti alle cache, in modo che esse possano cominciare a "cacheare" oggetti altrimenti non "cacheabili".

3.9. Problemi Generali della Tecnologia

Il principale problema legato all'utilizzo di informazioni "cacheate" e alla consistenza dei contenuti è stato ampiamente trattato nei paragrafi precedenti. Nel seguito accenniamo brevemente ad altre problematiche della tecnologia.

Innanzi tutto, c'è un problema di sicurezza dei dati che non riguarda solo i sistemi di caching, ma tutti i sistemi di rete. Una possibile soluzione è l'utilizzo di tecnologia a chiave pubblica, applicata ai contenuti web. I siti potrebbero "firmare" crittograficamente le pagine web, in modo che il client che le riceve possa verificare sia l'integrità della versione ricevuta, sia l'identità del server.

Un secondo problema è legato alla privacy. Le cache possono agevolare il controllo delle attività di navigazione web degli utenti ed altri controlli sui contenuti cui gli utenti possono accedere poiché offrono un singolo punto di controllo. Gli sforzi per creare una gerarchia di cache mondiale (il cosiddetto "global mesh") potrebbero, potenzialmente, aumentare questo tipo di rischi. Comunque, nel caso di due cache coinvolte in una richiesta, spedita attraverso il global mesh, la seconda cache sa da dove proviene la richiesta, ma non può procurarsi informazioni addizionali sull'utente da cui ha avuto origine. Anche questo problema non riguarda soltanto le cache: per ogni richiesta HTTP, esiste sempre un punto che tiene traccia della richiesta, se non è una cache è, ad esempio, l'origin server.

Le politiche sulla privacy delle informazioni dei siti web non hanno un effettivo valore se le cache intermedie hanno il controllo dei dati d'accesso (che possono includere informazioni significative sui siti e sugli utenti). Tali informazioni potrebbero essere vendute per ricerche di marketing, per scopi investigativi o per essere inserite in database commerciali. Anche se un sito web ha policy molto restrittive riguardo all'informazione raccolta sui siti o fornita dagli utenti, non è garantito che le informazioni "cacheate" provenienti da quel sito siano trattate con

la stessa policy [6].

Infine, le cache possono impedire ad un sito di stimare accuratamente i visitatori che accedono alle pagine, poiché non possono fare valutazioni sugli hit delle cache. È probabile che i siti più popolari siano pesantemente "cacheati" e per questo le statistiche del numero di accessi possono essere fortemente alterate verso il basso, a causa delle "deviazioni" delle richieste verso le cache.

4. Sperimentazione della tecnologia di Transparent Web Caching

Nella progettazione di un servizio di rete, la sperimentazione di nuove tecnologie consente di verificare l'applicabilità di soluzioni innovative ad uno specifico contesto e supporta la fase di analisi costi/benefici, fondamentale per un attento indirizzo negli investimenti.

Nel seguito introduciamo brevemente il servizio di web caching del CNR, quello del GARR, riorganizzazione di recente e, infine, motiviamo e descriviamo la nostra sperimentazione.

4.1. Il Servizio Cache Server del CNR

L'architettura di cache server attualmente utilizzata nel CNR è costituita da una gerarchia a due livelli [11] di proxy cache server omogenei: tutti i nodi utilizzano Squid, in diverse versioni e/o su piattaforme eterogenee. Nel livello più alto della gerarchia troviamo i cache server di Bologna e Roma (top level), correlati mediante una reciproca relazione di sibling. I rimanenti server sono tutti di secondo livello e riferiscono i due top level cache server, come mostrato in Figura 4.1.

Ogni server di secondo livello riferisce entrambi i server di primo livello ma con pesi diversi. Questo consente di avere una configurazione con un server primario e un server di back-up, che sarà utilizzato nel caso che il parent primario sia temporaneamente irraggiungibile o in caso di congestionamento delle linee. Nell'algoritmo di selezione del parent utilizzato da Squid sono privilegiati i server con pesi maggiori.

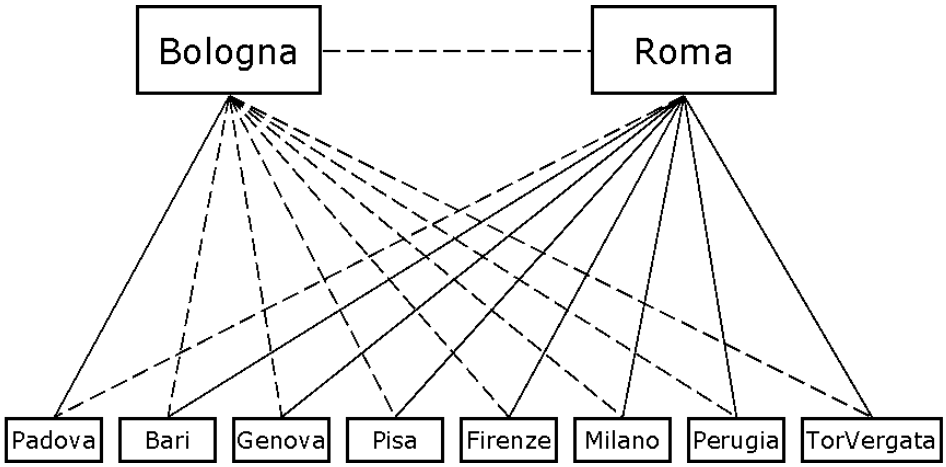

Figura 4.1 - Schema del servizio Proxy Cache Server del CNR

Nella Figura 4.1, le relazioni tra i server di primo e secondo livello sono evidenziate con linea continua (parenting) e linea tratteggiata (sibling).

Sono inoltre attive relazioni di sibling con le Università, non evidenziate dalla figura (ad esempio, Pisa riferisce come sibling il proxy.unipi.it) e delle relazioni tra i Top Level cache server ed i server dell'NLANR [44] per le richieste originate per i TLD .com, .edu, .org (global mesh).

4.2. Evoluzione del Servizio di Web Caching del GARR

Nel 1999 il GARR ha pubblicato un bando per affidare la progettazione e lo sviluppo di un servizio di web caching e Mirroring a livello nazionale. Nel 2000, il progetto è stato affidato al CILEA (Consorsio Interuniversitario Lombardo per L'elaborazione Automatica) che ne è risultato vincitore. Il progetto ha come obiettivo primario "la più ampia diffusione dell'uso dei sistemi di caching in ambito GARR sia a livello dei gestori sia di utenti finali" [5].

Questo ha portato a:

- Creazione di un centro di supporto (help-desk) diretto ai gestori delle cache di secondo livello (per indirizzare la scelta, il dimensionamento e la configurazione dei server).
- Progettazione e attuazione di una campagna informativa capillare per la diffusione di cache di secondo livello.

In questa visione, una migrazione della architettura di web caching del CNR è necessaria per una efficace integrazione con la dorsale cache offerta dal GARR.

Lo scopo della nostra sperimentazione è valutare la tecnologia di transparent web caching per stimare la sua efficacia. Tale soluzione, infatti, sembra interessante, perché elimina il carico amministrativo richiesto per la configurazione dei client, consentendo, al tempo stesso, di modificare la configurazione e/o il servizio, in modo del tutto trasparente agli utenti. Dato che, come è stato detto nel § 3.1, la redirezione del traffico verso la cache può portare a rallentamenti (del resto è tutto relativo, perché dipende dalla capacità della cache di gestire migliaia di connessioni per secondo), un punto fondamentale della nostra sperimentazione è verificare se, nel caso peggiore quando la richiesta di un oggetto genera un miss (quindi, la richiesta va inoltrata verso l'origin server), la qualità del servizio degrada in modo percettibile per l'utente.

4.3. I Test

La sperimentazione comprende due parti:

1. Il primo esperimento effettua una redirezione statica del traffico web generato dalla rete dello IAT (via router Cisco), verso il cache server Squid (attivo su un PC dedicato).
2. Il secondo esperimento si basa su una redirezione dinamica del traffico (via router) e un

---

[44] National Laboratory for Applied Network Research http://www.nlanr.net/

cache server commerciale di fascia bassa, perché la versione 2 del protocollo WCCP, al momento, non è ancora supportata da Squid e la versione 1 (supportata) comporta noti problemi di efficienza sui router.

Prima di passare alle sperimentazioni, è necessario introdurre una breve descrizione dell'ambiente in cui esse sono state effettuate.

4.3.1. I Client

Sulla LAN dello IAT sono interconnessi più di 100 computer, anche se meno della metà sono utilizzati dal personale per navigare in Internet. Nella rete, infatti, sono presenti sia un certo numero di server dedicati a servizi di rete per la comunità CNR (posta elettronica, web server, list server, ldap server, etc.), sia un elevato numero di PC utilizzati per accedere e gestire database locali utilizzati dalla Registration Autority Italiana. Naturalmente, tutto il traffico web locale, se pur elevato, non attraversa il cache server e quindi non è registrato nei file di log e nelle statistiche. Il numero medio di client che hanno acceduto il server nel periodo monitorato è risultato 34.

4.3.2. La Rete

La rete dell'Area della Ricerca CNR di Pisa è un buon esempio di costruzione di una rete con tecnologia ATM. Il cuore di questa complessa infrastruttura di rete è una dorsale composta da 6 switch ATM interconnessi da link di fibra ridondante a 155 Mbps, a copertura di tutti e tre gli edifici dell'Area.

Le LAN dei 15 Istituti sono costruite con un numero variabile di switch Ethernet 10/100 con interfacce ATM attestate al backbone. L'interoperabilità tra i due ambienti è ottenuta tramite LAN Emulation. Al vertice della gerarchia c'è un singolo router interconnesso alla rete attraverso una interfaccia ATM. Il router stesso fornisce l'accesso alla rete Internet attraverso un

link a 4Mbps verso la rete GARR[45]. La rappresentazione logica della rete è mostrata in Figura 4.2

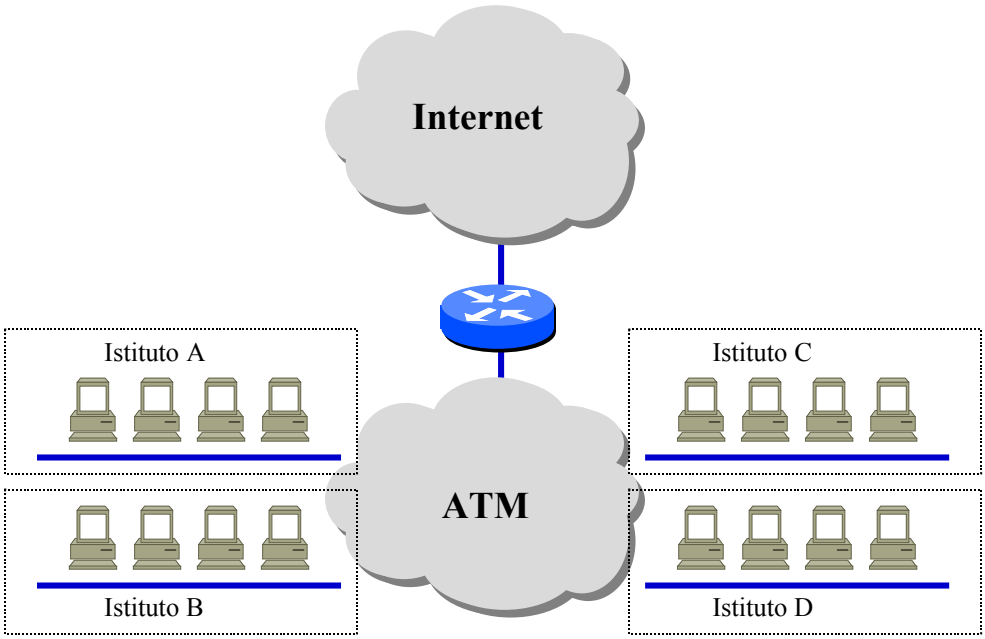

Figura 4.2 - Schema logico della rete dell'Area di Ricerca del CNR di Pisa

4.4. Primo Esperimento: Redirezione Statica

La redirezione statica del traffico si può ottenere tramite una opportuna configurazione del router. Il traffico HTTP generato dai client residenti sulla rete locale dello IAT viene redirezionato (staticamente) dal router verso il cache server dal router. La redirezione è stata effettuata utilizzando la feature "policy based" del Cisco IOS (la terminologia Cisco è "policy

[45] Nel Febbraio 2001 è stato fatto un upgrade a 8 Mbps

based routing"). La Figura 4.4 mostra un esempio di configurazione del router.

Il maggiore svantaggio di questo tipo di configurazione è il fatto di essere soggetta a failure. Il router non conosce lo stato del cache server, quindi continuerà a redirezionare i pacchetti verso di esso, anche se il server sarà down o se starà funzionando temporaneamente in modo non appropriato. Inoltre la soluzione non è scalabile e non offre bilanciamento di carico.

```
interface Ethernet2/4
description e2/4---->Squid Cache
ip address 146.48.127.X
ip route-cache flow
…….
interface ATM3/0.65 multipoint
description ATM3/0.65---> LAN IAT
……
ip policy route-map TPROXY
route-map TPROXY permit Y
match ip address 111
set ip next-hop 146.48.127.Y
……..
access-list 111 deny tcp host 146.48.65.Z eq www   !eliminato dalla redirezione
access-list 111 permit tcp 146.48.65.0    0. 0.0.255 any eq www
```

Figura 4.3 - Configurazione del router (un esempio)

4.4.1. Criteri di Valutazione

Il nostro obiettivo è studiare gli effetti dell'introduzione della tecnologia di transparent web caching da molteplici punti di vista: performance, risparmio nella banda utilizzata, qualità del servizio percepito dall'utente, accettazione del servizio da parte della comunità di utenti ed

applicabilità del servizio nel rispetto della normativa italiana relativa alla privacy[46]. La prima parte del test è stata effettuata con tecnologia a basso costo. Questa scelta ci permette di sperimentare la tecnologia senza investimenti aggiuntivi. L'analisi dei dati raccolti consentirà di verificare l'efficacia del servizio e, in caso di risultati positivi, di definire i requisiti del sistema per lo specifico ambiente operativo. Questo approccio è utile perché consente di evitare investimenti in apparecchiature di rete non necessarie.

4.4.2. Il Cache Server

Il nostro test è cominciato in ambiente a basso costo: un Pentium III-450 Mhz, con 512 KB di cache, 512 MB di RAM, e un disco U/Dma di 14 GB. Sul sistema è stato installato il sistema operativo Linux Red Hat 6.1, kernel version 2.2.12-20 e il proxy cache software Squid, release 2.2-STABLE5.

Il cache server ha una porta Ethernet 10/100 direttamente connessa al router tramite un cavo incrociato TP[47]. La velocità di questa LAN dedicata è limitata a 10Mbps, dato che il router è dotato di porte Ethernet standard. La topologia di transparent caching implementata è delineata in Figura 4.3. Le FAQ[48] di Squid [38] forniscono dettagliate informazioni su come configurare il server per meglio adattarlo alle specifiche esigenze. Come primo passo, abbiamo applicato la configurazione di default presente nel file di configurazione *squid.conf*, modificando soltanto le opzioni relative alle dimensioni della memoria e del disco dedicati alla cache. In particolare, abbiamo dedicato 8 GB di spazio disco e 180MB di memoria per Squid. Nel prossimo

---

[46] L'accesso a dati che rivelano informazioni "sensibili" relative a persone, come lo stato di salute, la religione, le opinioni politiche, ecc. sono protette in Italia e nella Comunità Europea da numerose leggi (Legge 31 dicembre 1996 n. 675). Nello sviluppo di un sistema di caching, l'amministratore deve porre attenzione nel cancellare o rendere anonimi i file di log.

[47] Indicato con TP, abbreviazione di twisted-pair, il cavo che comprende due o più coppi di fili isolati intrecciati insieme

paragrafo, descriveremo alcune delle caratteristiche principali di Squid.

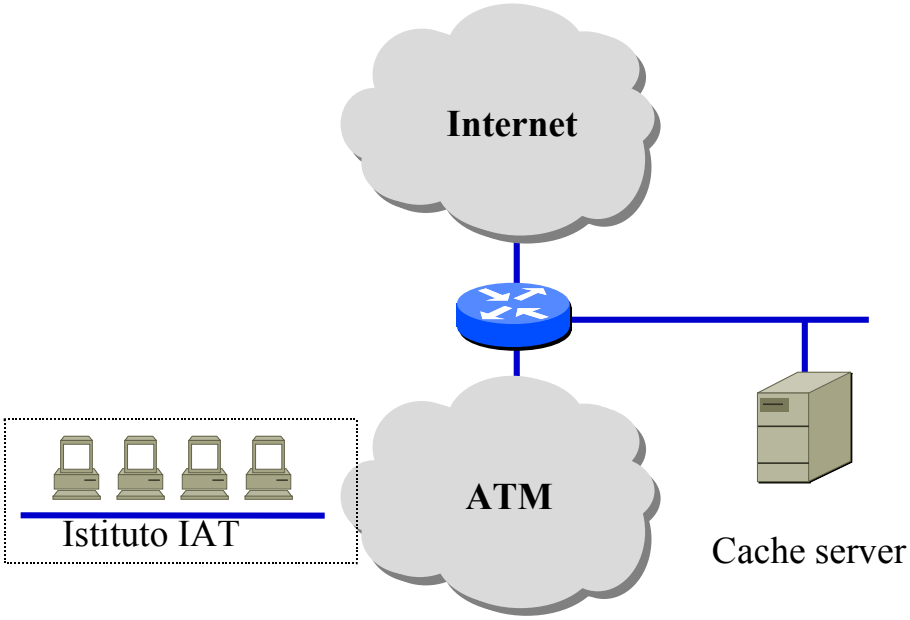

Figura 4.4 - Schema logico dell'architettura di Transparent Web Caching

4.4.2.1. Squid

Squid è un high-performance proxy cache server che supporta il caching di oggetti FTP, Gopher[49], e HTTP [38]. Il suo codice è scritto in C ed i sorgenti, liberamente distribuiti[50], evolvono principalmente grazie al lavoro dedicato dai laboratori dell'NLANR, ma, in parte, anche grazie al contributo di tutta la comunità scientifica internazionale.

Squid è multipiattaforma: è disponibile per pressoché tutte le versioni di Unix (AIX, FreeBSD, HP-UX, Linux, OSF/1, Solaris, SunOS, etc.) e per OS/2. È possibile compilare e

---

[48] Frequently Asked Questions

[49] Il protocollo Internet Gopher per la ricerca di documenti distribuiti; è un sistema che precede il World Wide Web nell'organizzazione e la visualizzazione dei file sui server Internet. Un server Gopher presenta i propri contenuti come una lista gerarchicamente strutturata di file.

[50] accessibili dalla URL http://squid.nlanr.net/Squid/.

utilizzare Squid anche su sistemi Windows NT, ma, allo stato attuale, il software non è ancora completamente affidabile su tale piattaforma. Supporta completamente i protocolli ICP v.2 e Cache Digest e supporta parzialmente HTCP e CARP. È possibile utilizzare Squid in soluzioni di transparent caching, anche grazie al supporto del protocollo WCCP v.1 [14]. È un prodotto molto flessibile, che offre la possibilità di configurare una serie di funzionalità molto interessanti, inclusa la possibilità di utilizzarlo come HTTPd-accelerator per ridurre il carico del server web e della rete locale.

La distribuzione del prodotto consiste in un server primario: *Squid*, un programma per il DNS lookup (*dnsserver)*, dei programmi opzionali (riscrittura delle richieste, attivazione dell'autenticazione, etc.) e degli strumenti di gestione. In particolare, il prodotto offre la possibilità di gestione remota via web (mediante moduli di tipo CGI) o via SNMP[51] (Simple Network Management Protocol). Alla partenza, Squid produce un numero considerevole di processi dnsserver, ognuno dei quali può eseguire un singolo DNS lookup. In questo modo, si riduce il tempo di attesa della cache per queste operazioni e inoltre, essendo processi separati, i dnsserver possono bloccarsi senza causare un blocco di Squid.

Squid può essere configurato anche in presenza di un firewall. Se il server si trova dietro il firewall, non si possono fare connessioni dirette verso il mondo esterno, così "bisogna" utilizzare una cache parent. Si può usare l'opzione inside_firewall in squid.conf per specificare una lista di domini interni al firewall. Con questa direttiva, gli oggetti che non fanno parte della lista di domini saranno considerati oltre il firewall.

Squid cerca di mantenere la dimensione del disco relativamente uniforme, facendo in modo che gli oggetti siano rimossi nella stessa percentuale in cui sono aggiunti. Per questa operazione di rimpiazzamento degli oggetti "cacheati", il server utilizza un algoritmo Least Recently

[51] Simple Network Management Protocol: un insieme di protocolli per amministrare reti complesse.

Used[52]. La soglia utilizzata per la rimozione (LRU age) è calcolata dinamicamente, in base alla quantità di spazio disco utilizzato. Quando la cache ha un maggiore carico di lavoro, la LRU age si abbassa, cosicché saranno rimossi più oggetti. La memoria disco di una cache Squid è implementata come una hash table, con un certo numero di hash bucket[53]. Squid scandisce un bucket alla volta e ogni oggetto, nel bucket, che non è stato acceduto per questa quantità di tempo sarà rimosso dalla cache per far posto a nuovi oggetti. Il ritmo di scansione è regolato in modo da prendere approssimativamente ventiquattro ore per la scansione dell'intera cache.

Squid offre, inoltre, un estensivo controllo degli accessi e mantiene il log dell'intera richiesta. Le richieste HTTP possono essere anonimizzare, per evitare che sia memorizzato l'indirizzo del client che ha fatto la richiesta.

Il file di configurazione squid.conf contiene i valori di default per varie opzioni ed è suddiviso in cinque sezioni:

- opzioni di rete (numeri delle porte da cui ascoltare e verso cui mandare richieste e risposte, etc.);
- opzioni riguardanti la selezione delle cache *neighbor* (vicine), ad esempio quali sono le altre cache in gerarchia, i domini per certe richieste, i timeout etc.;
- opzioni sulla dimensione della cache e del processo Squid;
- opzioni sui pathname dei file di log e delle directory della cache;
- opzioni per programmi di supporto esterni (ftp, processo del dnslookup).

4.4.3. Sorgenti dei Dati

Il tipo di analisi effettuato ha richiesto la raccolta di dati relativi sia al cache server sia al

[52] LRU least recently used: la regola che seleziona un oggetto da rimpiazzare se è stato usato (in lettura o scrittura) meno recentemente di ogni altro.

router. Queste multiple sorgenti di dati hanno consentito di collezionare tutti i dati necessari e validare la consistenza dei valori di una stessa variabile raccolti da sorgenti differenti.

4.4.3.1. I File di Log di Squid

I file di log di Squid includono importanti informazioni che possono essere elaborate con vari strumenti. Al fine di rispettare la privacy degli utenti, abbiamo configurato Squid in modo da rendere anonimi i file di log.

In particolare, abbiamo usato il sofware *Calamaris*[54] per estrarre i seguenti dati:

- Il traffico totale in numero di richieste e in byte
- hit/miss rate delle richieste
- hit/miss rate in termini di volume di traffico in byte
- La percentuale di richieste destinate al TLD (Top Level Domain) .it.

Nel file *access.log* sono conservate le informazioni riguardanti tutte le richieste degli utenti verso il proxy. Da questo file si può ricavare quanti client utilizzano la cache, quali pagine sono più popolari, etc. Esistono due formati di base, a seconda della configurazione adottata: *common* e *native*. Per default, Squid utilizza il formato native, contenente i seguenti campi:

time elapsed remotehost code/status bytes method URL rfc931 peerstatus/peerhost type

dove:

- time è il timestamp della richiesta (UTC, Universal Coordinated Time), espresso in secondi, con una precisione al millisecondo.
- elapsed è il tempo, in millisecondi, trascorso durante la connessione.
- remotehost è l'indirizzo IP del client che ha fatto la richiesta.
- code è un codice che descrive il risultato della richiesta: un miss, un hit o una richiesta di

---

[53] Un bucket è definito come un certo sottoinsieme della sezione di hash assegnata alla cache.

validazione con l'origin server. I codici che iniziano per "TCP_", si riferiscono alle richieste HTTP (ad esempio, TCP_HIT indica che una copia valida dell'oggetto richiesto è stata trovata nella cache; TCP_MISS indica che l'oggetto richiesto non è presente nella cache; TCP_REFRESH_HIT indica che l'oggetto richiesto è stato trovato nella cache, ma era stale e la successiva richiesta If-Modified-Since inviata all'origin server ha dato come risultato "not_modified"), mentre i codici che iniziano per "UDP_" si riferiscono a richieste ICP (ad esempio, UDP_HIT, UDP_MISS).

- status codice di stato HTTP, preso dalla prima linea del response-header (ad esempio, 200 "success", 400 "bad request"). Per richieste ICP, è sempre 000.
- bytes, per richieste TCP è il numero di byte recapitato al client; per richieste ICP è la dimensione della richiesta, in byte.
- method, per le richieste HTTP è il metodo della richiesta, GET, POST, etc.; per le richieste ICP, il metodo ICP_QUERY.
- URL è la URL richiesta.
- rfc931 è lo username associato alla connessione del client, è il risultato di un Ident lookup (RFC 931 [34]); l'Ident lookup è disabilitato per default e viene sostituito da un "-".
- peerstatus descrive come e da dove l'oggetto richiesto è stato acquisito:
  1. da una cache peer (ad esempio, SIBLING_HIT se l'oggetto è stato richiesto ad una cache sibling che ha risposto con un UDP_HIT)
  2. direttamente dall'origin server (DIRECT).
- peerhost è l'host verso cui è stata spedita la richiesta, cioè da dove arriva l'oggetto.
- type è il content-type dell'header della risposta.

---

[54] È un tool per analizzare il file access.log di Squid; genera report in formato ASCII o HTML.

Il file *store.log* mostra cosa sta accadendo sul disco, in relazione agli oggetti salvati o rimossi dalla cache. Ha il formato:

time action file-number status datehdr lastmod expires type sizes method key

dove:

- time è il momento in cui l'entry è stata registrata.
- action è una delle azioni effettuate sull'oggetto:
  1. RELEASE l'oggetto è stato rimosso dalla cache;
  2. SWAPOUT l'oggetto è stato salvato su disco;
  3. SWAPIN l'oggetto esisteva sul disco ed è stato messo in memoria;
  4. CREATE sembra essere inutilizzato.
- file-number il numero di file per la memorizzazione dell'oggetto. Ad esempio, un file-number uguale a FFFFFFFF denomina oggetti "memory only" e ogni azione su un tale file-number si riferisce ad un oggetto che esiste solo in memoria e non su disco.
- status è il codice di stato di risposta HTTP, se disponibile, 555 altrimenti; per le richieste ICP è sempre 0.
- datehdr il valore del campo Date del response-header.
- lastmod il valore del campo Last-Modified del response-header.
- expires il valore del campo Expires del response-header
- type il valore del campo HTTP Content-Type del response-header, unknown se non può essere determinato.
- sizes, questa colonna consiste di due campi separati:
  1. il valore del campo Content-Length del response-header, è uguale a 0 se questo campo non è determinabile;
  2. il num. di byte del contenuto letto effettivamente.

- method il metodo HTTP richiesto;
- key la chiave di accesso ad un oggetto nella cache, spesso è la URL.

Il file *cache.log* contiene i messaggi di debug e di errore generati da Squid.

4.4.3.2. MRTG

MRTG (Multi Router Traffic Grapher) [31] è uno strumento per la raccolta di dati da dispositivi di rete tramite SNMP. I dati (le variabili MIB, Management Information Base) sono raccolti in file (log) e rappresentati come grafi immersi in pagine web. Tipicamente campiona dati ogni cinque minuti e crea giornalmente, settimanalmente, mensilmente e annualmente grafi, che possono essere utilizzati per monitorare il cache server pressappoco in real-time e per osservare i trend di utilizzo/performance della cache a lungo termine. Per poter fare query SNMP a Squid, per prima cosa il sorgente deve essere compilato con l'opzione enable-snmp. Il secondo passo è configurare Squid per accettare richieste SNMP (il valore di default è negare l'accesso SNMP) [24]. Un esempio dell'utilizzo di MRTG nella sperimentazione è dato dalle Figure 4.5 e 4.6.

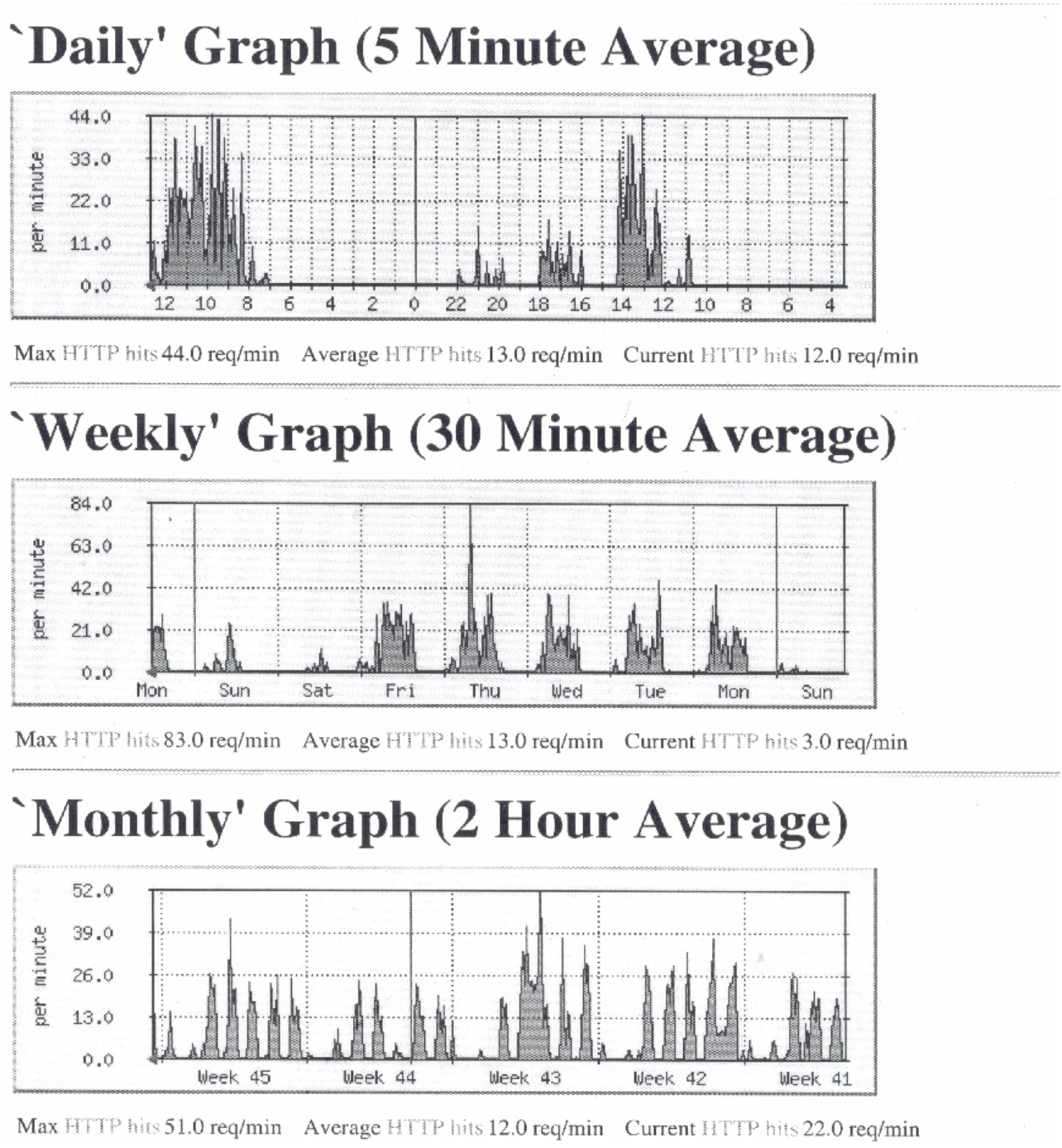

Figura 4.5 - Statistiche del Proxy Cache: hit HTTP

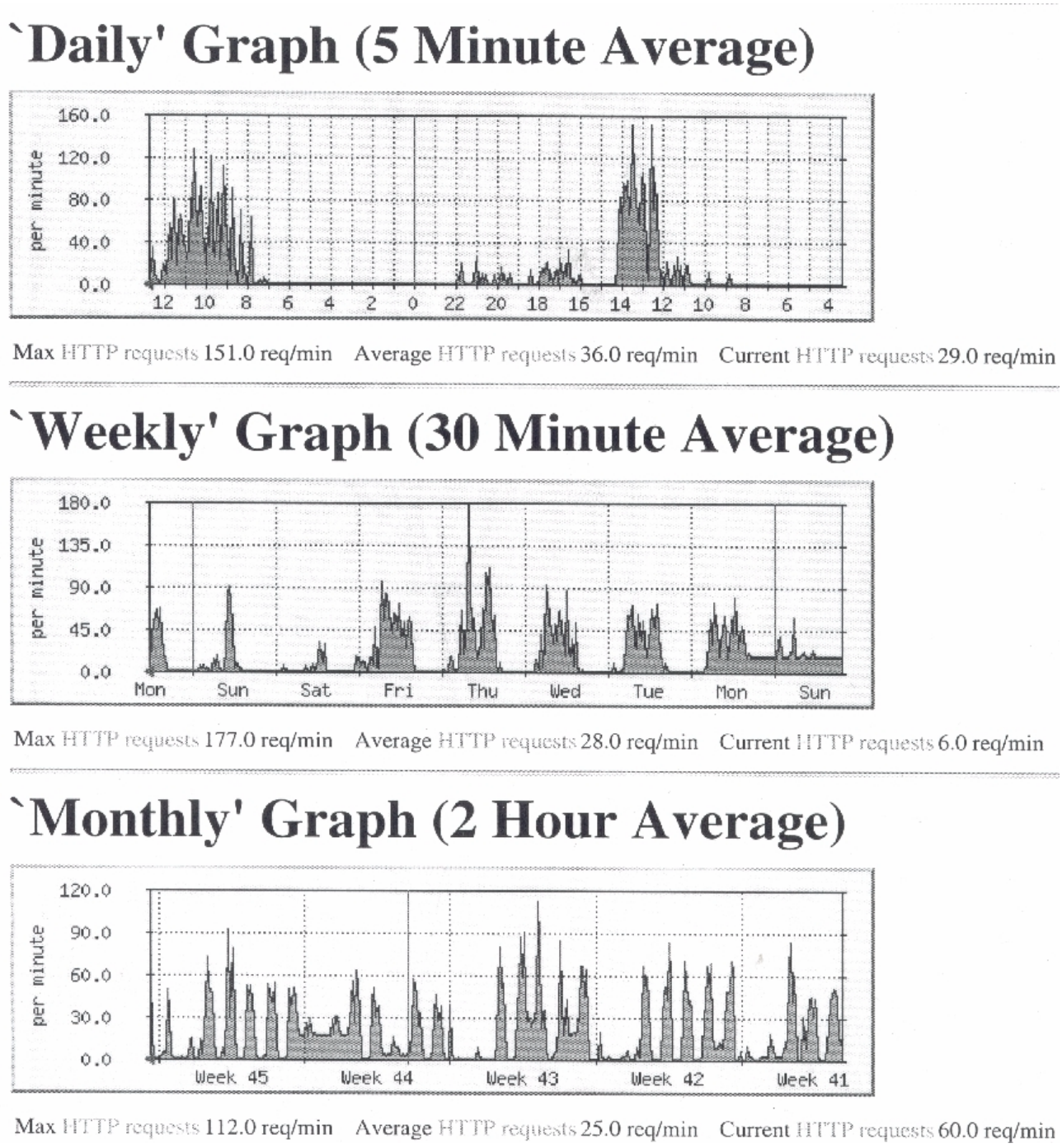

Figura 4.6 - Statistiche del Proxy Cache: richieste HTTP

4.4.3.3. NetFlow

NetFlow è una feature per switching IP ad alte prestazioni, fornita dal Cisco IOS che, tra l'altro, raccoglie un esteso insieme di statistiche sul traffico IP nella rete. I dati raccolti sono

flussi di traffico (*traffic flows*), cioè sequenze unidirezionali di pacchetti tra una data sorgente ed una destinazione finale condividenti lo stesso protocollo e le stesse informazioni a livello di trasporto (transport-layer). Le statistiche raccolte, quindi, sono esportate dai router e possono essere utilizzate per molteplici compiti (monitoraggio della rete, analisi e pianificazione, contabilità, rilevamento di attacchi, etc.). Per raccogliere ed analizzare i flussi esportati, è stato utilizzato Cflowd [1] un pacchetto sviluppato da CAIDA[55]. È chiaro che la fine granularità delle informazioni collezionate richiede la memorizzazione di una grande quantità di dati. Anche se i flussi sono archiviati in un formato compatto, i file contenenti le statistiche giornaliere raggiungono in media i 50Mbyte; questi file mantengono traccia di ogni singolo byte entrato o uscito dalla rete, permettendo qualsiasi tipo di preciso calcolo sul traffico. Comunque, le utility per la visualizzazione e il post-processing del contenuti dei file hanno caratteristiche di filtraggio e aggregazione limitate e non possono essere utilizzate per estrazioni di dati complesse. Noi abbiamo utilizzato questi dati per avere l'ammontare totale dei dati generati e ricevuti dalla rete dello IAT e l'ammontare del traffico HTTP servito sia dalla cache sia direttamente dai server web (origin server).

4.4.4. L'Analisi dei Dati

4.4.4.1. La Percentuale di Hit

Nel periodo osservato, il traffico web è variato da 90 a 180 MB nei giorni lavorativi, con valori inferiori durante i week-end e giorni festivi. Il valore medio del traffico nei giorni feriali nel periodo di 8 settimane è stato di circa 118 MB al giorno. Il diagramma in Figura 4.7 mostra le statistiche di Aprile 2000. Possiamo osservare che l'object hit rate varia dal 30% al 50%, con una media del 41,19%. Il byte hit rate si è mantenuto in un range tra il 12% e il 23%, con un valore medio di 19,23%. Questo mostra che la maggior parte degli oggetti mantenuti nella cache

[55] Cooperative Association for Internet Data Analisys http://www.caida.org/

sono di piccole dimensioni.

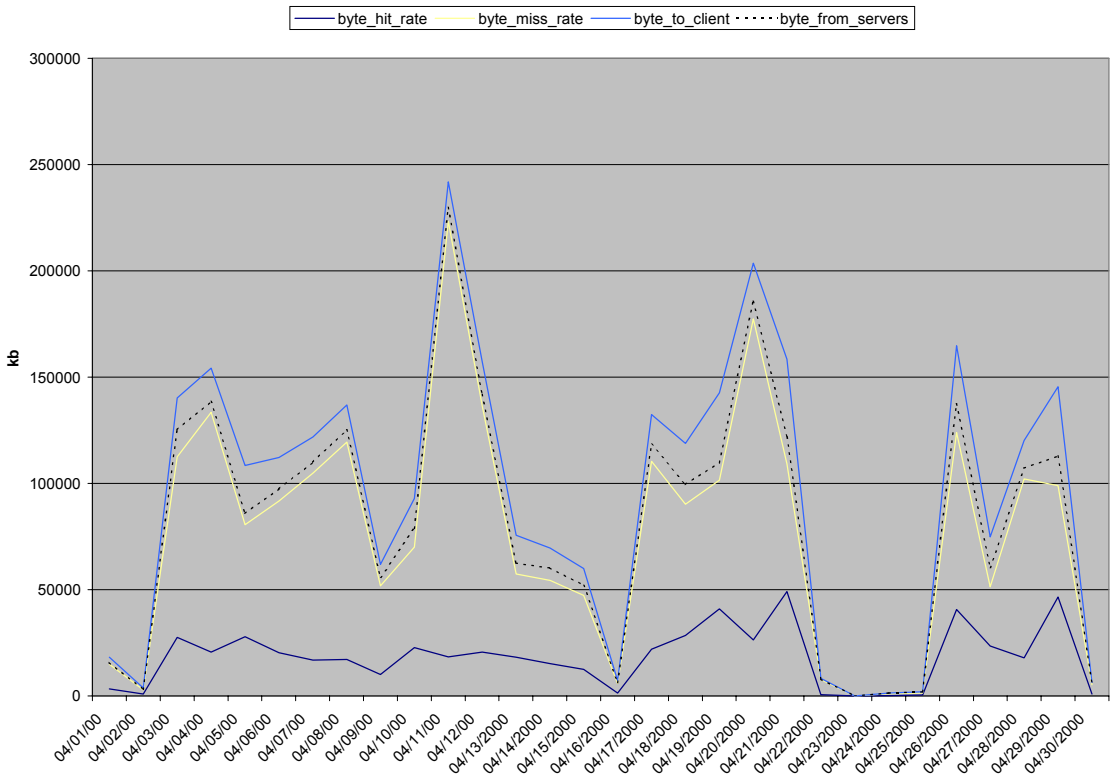

Figura 4.7 - Statistiche del Proxy Cache, Aprile 2000

4.4.4.2. Variazione della Dimensione degli Oggetti

Come accennato in precedenza, come primo passo della sperimentazione, abbiamo applicato la configurazione di default di Squid, in cui il valore della dimensione degli oggetti memorizzabili nella cache è 4 MB (configurando il parametro maximum_object_size).

Sulla base dei risultati dei primi due mesi di sperimentazione, abbiamo deciso di raddoppiare il valore di tale parametro da 4MB a 8MB, al fine di aumentare il valore del byte hit rate cercando un giusto compromesso tra salvare la banda e aumentare la velocità.

A Maggio e Giugno abbiamo potuto osservare che il valore medio del byte hit rate è cresciuto fino al 20,70 % con un object hit rate di 42,45%. È comunque importante notare che

durante l'intero esperimento la cache stava popolandosi, allargando quindi l'insieme degli oggetti memorizzati nella cache e potenzialmente disponibili per richieste di altri utenti. Il processo di popolamento ha richiesto 5 mesi: dal 2 Maggio al 2 Agosto la cache è passata dal 34% al 90%.

4.4.4.3. Modifiche nell'Utilizzo della Banda

La Figura 4.8 mostra il traffico totale in ingresso nella nostra rete. Si vede che la curva del traffico web (Mbytes_to_clients) è piuttosto bassa rispetto al traffico totale entrante.

Non ci sono anomalie in questo: il principale server di posta elettronica del CNR, che effettua l'hosting del servizio di e-mail per diversi altri istituti CNR, risiede sulla rete dello IAT. In aggiunta, sullo stesso sistema sono presenti altri servizi come mailing list, fax gateway e conversioni di formato di documenti; questo, comunque, non invalida i benefici del web caching.

Sebbene la banda di rete salvata sia significativa, ciò non porta ad una riduzione dei costi perché il CNR paga un costo fisso per il suo accesso ad Internet.

4.4.4.4. Caratterizzazione del Traffico

L'analisi dei file di log di Squid ci ha aiutato a meglio capire gli interessi della comunità di utenti dell'Area e il volume di dati scaricati dai TLD. Il TLD più frequentemente visitato è risultato il .it, seguito dal .com. Un totale del 40% dei byte richiesti, corrispondenti al 54% di oggetti richiesti, era indirizzato al TLD .it.

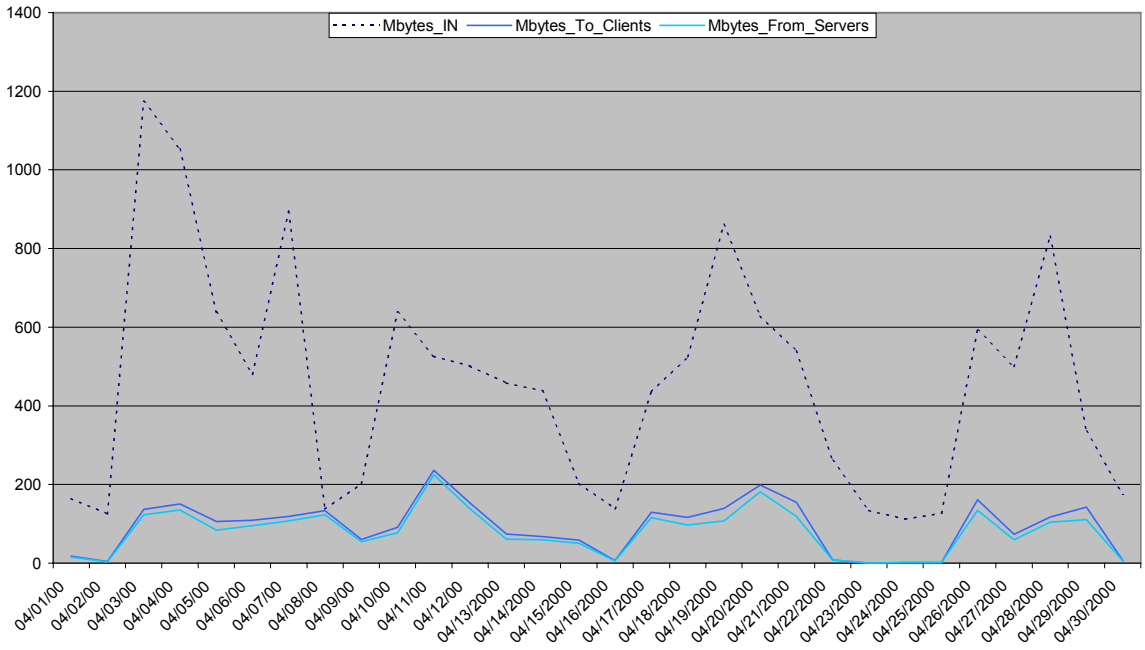

Figura 4.8 - Confronto fra traffico totale di rete e traffico Web

4.4.4.5. La Latenza Introdotta dal Router e dal Cache Server

Nel paragrafo precedente abbiamo mostrato come l'utilizzo del cache server abbia un effetto positivo sul consumo di banda. Abbiamo anche verificato che il tempo medio di scaricamento di oggetti web (request hit rate) decresce, riducendo significativamente la latenza e migliorando la qualità del servizio per l'utente. Comunque, nel caso sfortunato in cui l'oggetto richiesto non sia disponibile nella cache e debba essere recuperato dall'origin server, la latenza aumenta poiché i pacchetti relativi a quella connessione HTTP devono viaggiare due volte attraverso il router, il segmento di rete cache-router e il cache server stesso. Abbiamo provato a calcolare e/o stimare questa latenza e i suoi effetti sulla qualità del servizio percepito dagli utenti.

Il router ha un ruolo centrale nella nostra topologia di transparent caching: con la feature "policy routing" attiva, esso redirige il traffico TCP destinato alla porta 80 verso il cache server. Sebbene la latenza introdotta dal router sia misurabile e definita, alcune considerazioni

dimostrano che essa rappresenta solo una piccola frazione del tempo di viaggio dei pacchetti:

- I router Cisco con IOS 11.3 o versioni successive eseguono policy routing via fast switched path (le precedenti versioni le eseguono con il più lento processo switched path, via processore, in questo modo si rallenta molto l'elaborazione del pacchetto). Non c'è degradazione di performance quando è configurata la feature policy routing.
- Il nostro router è un high-end serie Cisco7505 con switching bus e interfacce di rete veloci.

Anche la latenza introdotta dal segmento di rete cache-router è irrilevante. Il RTT (Round-Trip Time) medio tra il cache server (10Mbps Ethernet) e i client (100Mbps Fast Ethernet) è un ms. In condizioni ottimali il RTT è veramente basso (0.0004 sec) raggiungendo i valori massimi di pochi ms quando la rete è carica. Purtroppo la porta Ethernet del router non è full duplex, altrimenti il trasferimento sarebbe stato ancora più veloce.

La latenza introdotta dal cache server dipende da vari fattori come la velocità della CPU, le dimensioni della memoria, la performance dei dischi di I/O, il sistema operativo, etc. Invece di eseguire il monitoraggio e il calcolo di dati complessi, abbiamo misurato un solo parametro: il tempo trascorso tra la richiesta originaria del client e quella effettuata dal cache server al fine di richiedere direttamente e memorizzare l'oggetto. Sulla base dell'assunzione che la latenza introdotta dal router e dal segmento di rete cache-router possa essere trascurata (come descritto sopra), questo "switching path" nel cache server rappresenta la latenza globale[56]. In definitiva, rispetto al tempo che si otterrebbe se i client andassero a recuperare direttamente gli oggetti dagli origin server, il tempo di download degli Internet object viene aumentato da questo valore (latenza).

[56] Questo è vero solo per connessioni http persistenti. Con http 1.0 ogni singola connessione http è ritardata da questo valore.

Per misurare questo tempo, sono stati utilizzati due metodi: *tcpdump*[57] e netflow[58]. I dati raccolti con tcpdump hanno una maggiore accuratezza (pochi microsecondi) ma sono in formato meno leggibile[59]. Quelli collezionati da netflow sono pronti per l'uso, ma meno accurati e il tempo di creazione dei flussi è espresso in centinaia di secondi.

I valori misurati variano tra 2 e 70 ms. Se confrontiamo questi valori con il tempo medio di download degli oggetti ritrovati dall'origin server (680ms) possiamo dedurre che la latenza introdotta dal cache server non ha impatto sulla qualità del servizio e non è percepibile da un utente.

4.5. Secondo Esperimento: Redirezione Dinamica

La seconda parte dell'esperimento testa una configurazione di web caching con redirezione dinamica, via router, del traffico web verso un Cache Engine commerciale (Cisco CE 505).

La redirezione dinamica del traffico via router può essere attivata utilizzando il protocollo WCCP[60]. Il sistema operativo Cisco IOS offre il supporto per il WCCP, fornendo una soluzione dinamica di transparent web caching. Con questo protocollo i client spediscono le richieste web alla URL del web server desiderato, il router, intelligentemente, intercetta le richieste HTTP e, in modo trasparente, le redireziona verso la cache. Quando un router abilitato per il WCCP riceve un pacchetto IP, determina se è una richiesta da mandare alla cache (se la porta di destinazione è la 80), se lo è fa la redirezione, altrimenti il pacchetto segue il normale percorso verso la rete. Attraverso comunicazioni con le cache, i router che eseguono il protocollo WCCP hanno la possibilità di essere consapevoli della disponibilità delle cache: in questo modo, la

---

[57] Il tcpdump restituisce l'header dei pacchetti su una interfaccia di rete compatibili con una data espressione booleana

[58] § 4.4.3.3

[59] Per facilitare l'analisi dei dati ottenuti da tcpdump è stata utilizzata l'utility EtherPeek di ag group,inc.

[60] vedi § 4.5.1

redirezione è effettuata solo se c'è almeno una cache operativa.

Nei prossimi paragrafi, approfondiremo il funzionamento del protocollo WCCP, già introdotto nel § 3.6.2 e descriveremo, quindi, le funzionalità del Cache Engine.

4.5.1. Il Protocollo WCCP

WCCP [14] [15] è un acronimo per Web Cache Coordination Protocol. Il protocollo WCCP permette di associare uno (nel caso della versione 1 del protocollo) o più (nella versione successiva) router ad una o più web cache (più web cache costituiscono una cache farm o "cluster"), per redirezionare il traffico HTTP, consentendo, contemporaneamente, un bilanciamento del carico. La cache con indirizzo IP minore (detta *designed web cache*) ha il compito di decidere in che modo il router deve distribuire il traffico nel cluster.

In caso di guasto di una delle cache, il router è in grado di accorgersene e può provvedere automaticamente ad escludere quella cache dal cluster, interrompere l'invio delle richieste verso di essa e continuare ad utilizzare le cache rimaste. Se la designed web cache smette di funzionare, il suo posto viene preso da una delle cache rimanenti. Nel caso di una singola cache, i dati vengono inviati direttamente verso il sito Internet di destinazione (fail-safe policy).

L'intercettazione e la redirezione trasparenti del traffico, non sono nulla di nuovo, infatti, erano già state realizzate con altre tecniche, come il routing "policy-based" oppure gli switch. L'idea nuova del WCCP è di fornire l'abilità di intercettare il traffico in modo trasparente, redirezionandolo verso le cache, cercando simultaneamente di rimediare alle mancanze di molte delle metodologie di intercettazione e redirezione alternative:

- Scalabilità: il traffico rediretto può essere distribuito automaticamente in un cluster, composto fino a 32 cache comunicanti con un router, senza riconfigurazione addizionale router/switch.
- Meccanismi di fail-safe integrati: la redirezione continua soltanto se ci sono cache

operative.

Nel WCCP v.1 può essere intercettato e rediretto soltanto il traffico TCP destinato alla porta 80. La funzione hash "quale-traffico-spedire-a-chi" è definita dalla designed web cache (che esegue il compito per l'intero cluster). Oltre a questa limitazione, la versione 1 del protocollo presenta altri problemi:

- Soltanto un singolo router può comunicare con un cluster di cache.
- Le richieste inviate verso una cache non possono essere rimandate indietro verso il cammino originario, se la cache non è in grado di servirle.
- Ci sono problemi di performance dovuti al crescente uso della CPU da parte del router.

Con l'intento di ovviare ad alcuni di questi problemi, è stata sviluppata una nuova versione del protocollo. Il WCCP v.2 permette ad uno o più router abilitati per la redirezione trasparente di connettersi con una o più web cache. Dopo aver stabilito la connessione, i router e le cache formano dei *Service Group* (gruppi di servizio) che gestiscono la redirezione del traffico, secondo alcune regole che sono parte della definizione del gruppo. Il protocollo fornisce i mezzi per negoziare lo specifico metodo utilizzato per la distribuzione del carico tra le web cache e il metodo usato per trasportare il traffico tra router e cache. Recentemente, la versione 2.0 del protocollo è stata resa di pubblico dominio e sottoposta alla comunità in forma di Internet Draft [15], in modo da consentire il supporto per il WCCP v.2 anche in prodotti di altre aziende o in software freeware (come Squid che implementa già il supporto per la versione 1).

4.5.1.1. Overview

- Supporto Multi-Router. La relazione Cache-to-Router è ora molti-a-molti: il protocollo permette ad una farm di web cache di essere collegata a più router.
- Supporto Multicast. WCCP v.2 supporta il multicasting dei messaggi del protocollo tra web cache e router, in questo modo, né le cache né i router necessitano di avere esplicitamente configurati altri indirizzi IP, ma possono comunicare attraverso il comune

servizio multicast.

- Aumento della sicurezza. C'è la possibilità (opzionale) di autenticare i pacchetti del protocollo ricevuti da web cache e router.
- Supporto per la redirezione del traffico non HTTP. Può essere intercettato e rediretto qualsiasi traffico IP.
- Restituzione dei pacchetti. WCCP v.2 permette ad una web cache di rifiutarsi di servire un pacchetto rediretto e di restituirlo al router perché possa rispedirlo (ad esempio, per metodi non conosciuti)[61].
- Metodi d'assegnamento multipli. La designed web cache in un Service Group può negoziare il metodo con cui i pacchetti sono distribuiti tra le web cache del gruppo. I pacchetti possono essere assegnati utilizzando uno schema hash o una maschera. Con un assegnamento hash, ogni router utilizza un hash table di redirezione con 256 bucket per distribuire il traffico tra le cache di un Service Group. La responsabilità di assegnare la hash table di redirezione del router è della designed web cache, che comunica l'assegnamento agli altri membri del Service Group. Se viene utilizzata una maschera, ogni router si serve di una maschera e di una tabella di valori per distribuire il traffico per un Service Group. Anche in questo caso, la designed web cache ha la responsabilità di assegnare ogni insieme maschera/valore del router, all'interno del Service Group.

4.5.1.2. Informazioni di Stato e Comandi

Il protocollo include un meccanismo che permette alle cache di passare un comando ai router nel Service Group. Lo stesso meccanismo (che è implementato tramite la componente "Command Extension" nei messaggi WCCP2_HERE_I_AM e WCCP2_I_SEE_YOU) può

[61] § 3.8.2

essere impiegato dai router per passare informazioni di stato alle cache in un Service Group.

Una cache si congiunge e mantiene la sua membership ad un Service Group trasmettendo un messaggio WCCP2_HERE_I_AM verso ogni router nel gruppo, ad intervalli di HERE_I_AM_T secondi. Nel messaggio WCCP2_HERE_I_AM, la componente "Web Cache Info" identifica la cache tramite l'indirizzo IP, la componente "Service Info" identifica e descrive il Service Group al quale la cache vorrebbe partecipare. Un router risponde con un messaggio WCCP2_I_SEE_YOU. La componente "Router Identity" di questo messaggio include una lista di cache cui il pacchetto è indirizzato. Se una cache non è nella lista scarterà il messaggio.

Un Service Group è identificato dal tipo di servizio e da un identificatore (Service ID). I servizi sono di due tipi: i "Well Known Service", conosciuti sia dai router sia dalle cache, che non richiedono altra descrizione oltre ad un Service ID, ed i "Dynamic Service", che devono essere descritti al router. Un router configurato per partecipare ad un particolare dynamic Service Group riceve la descrizione del traffico ad esso associato tramite il messaggio WCCP2_HERE_I_AM della prima cache che si congiunge al Service Group. Nella componente Service Info ci sarà le descrizione del Service Group che, ad esempio, contiene il numero della porta su cui fare la redirezione. Una volta che il Dynamic Service è stato definito, il router scarterà ogni seguente messaggio WCCP2_HERE_I_AM che descrive un Service Group per cui esso non è configurato.

La componente "Router Identity Info" del messaggio WCCP2_HERE_I_AM contiene un campo (Receive ID), mantenuto separatamente da ogni Service Group e incrementato ogni volta che il router manda un messaggio WCCP2_I_SEE_YOU al Service Group. Il router controlla il valore del campo Receive ID in ogni messaggio WCCP2_HERE_I_AM ricevuto da un membro del Service Group, se il valore non corrisponde a quello spedito nell'ultimo WCCP2_I_SEE_YOU, il messaggio viene scartato. Una cache è considerata un membro

operativo di un Service Group solo dopo che il router ha ricevuto un Receive ID valido.

Il WCCP v.2 mette a disposizione una *security component* in ogni messaggio del protocollo, per permettere una semplice autenticazione. Sono possibili due opzioni: "No Security" (default) e "MD5[62] password security". Quest'ultima richiede che ogni router e ogni cache che appartengono ad un Service Group siano configurati con una password. Ogni pacchetto spedito da un router o ad una delle cache per quel Service Group, nella *security component,* dovrà contenere la password del Service Group e il checksum del messaggio.

4.5.1.3. Considerazioni sulla Sicurezza

Il WCCP v.2 fornisce un meccanismo per l'autenticazione dei messaggi. Questo meccanismo dipende da una password conosciuta da tutti i router e le cache in un Service Group. La password è parte della configurazione del Service Group ed è usata per calcolare i checksum dei messaggi che possono essere verificati dagli altri membri del gruppo. Se qualche host venisse a conoscenza della password, potrebbe cercare di intralciare il lavoro del Service Group, sarebbe possibile per quell'host imitare i messaggi WCCP ed apparire come uno dei router o una delle web cache del Service Group. Per porsi come router in un Service Group, un host dovrebbe avvertire della propria presenza gli altri membri del gruppo con un messaggio I_SEE_YOU. Se fosse accettato come parte del Service Group, l'host riceverebbe la configurazione per il gruppo in un messaggio HERE_I_AM della designated cache. Questa situazione non dovrebbe presentare una qualche minaccia per l'operatività del Service Group, poiché l'host non sarebbe capace di effettuare alcuna redirezione dei pacchetti e tutti i pacchetti fluirebbero normalmente. Per porsi come una web cache all'interno di un Service Group, l'host dovrebbe segnalare la propria presenza in un messaggio HERE_I_AM. L'accettazione dell'host come parte del

[62]Un algoritmo creato nel 1991 da Ronald Rivest utilizzato per creare firme digitali. L'algoritmo prende come input un messaggio di lunghezza arbitraria e produce come output una stringa fissata di cifre, detta "message digest" (rappresentazione di testo in forma di una singola stringa di numeri).

Service Group sarebbe decisa dalla designated cache e dovrebbe essere soggetta ad un controllo di sicurezza addizionale non specifico del WCCP [15].

4.5.2. Il Cache Engine Cisco 505

Nella seconda parte dell'esperimento, è stato utilizzato un cache server commerciale che implementa la versione 2 del protocollo WCCP: il Cache Engine Cisco modello CE505. È un sistema dedicato con CPU modello AMD-K6 a 266 Mhz, disco di capacità 9,18 GB, 128 MB di RAM e due porte Ethernet 10/100 Mbps. Una delle porte è stata collegata al router con una linea half duplex a 10Mbps. Il software utilizzato dal sistema è il CE 2.20(0), mentre il protocollo di comunicazione abilitato è il WCCP v.2. Il disco è partizionato in un file system DOS e un file system dedicato alla cache. La partizione DOS (2 GB) contiene i file immagine, l'interfaccia utente ed è utilizzata dal sistema per l'output.

4.5.2.1. Caratteristiche Generali

Il Cache Engine può essere configurato per operare in una delle tre modalità: Web Cache (l'utilizzo fatto nell'esperimento), Reverse Proxy[63] e Custom Web Cache [4]. Il servizio che permette al router di redirigere il traffico HTTP verso il Cache Engine, su una porta diversa dalla porta 80, è detto "Custom Web Cache". L'implementazione del transparent web caching [Figura 4.9] via WCCP risulta semplificata rispetto ad altre implementazioni (routing policy-based e switch), non comporta degradazione di performance e non interferisce con le normali operazioni di routing.

[63] § 2.4.1

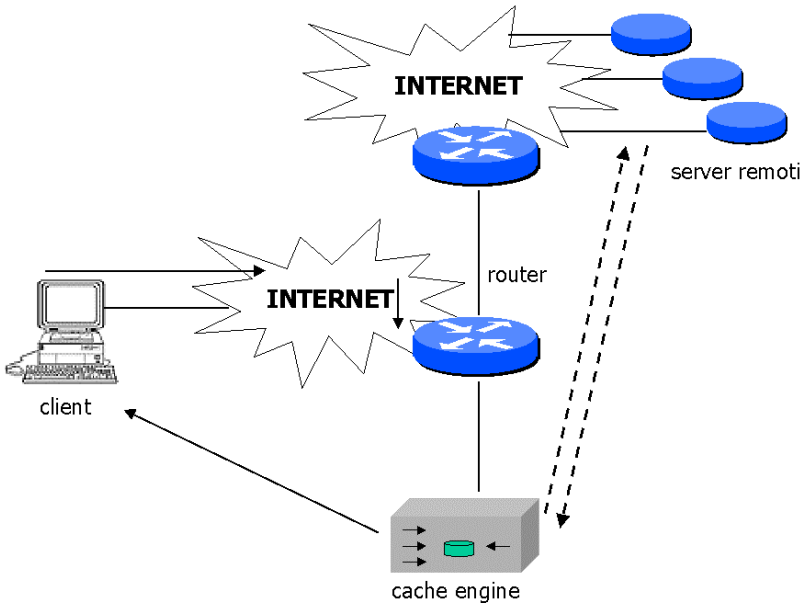

Figura 4.9 - Uso del Cache Engine in uno schema di Transparent Web Caching

Le principali caratteristiche offerte dal Cache Engine sono:

- È una soluzione di caching "integrato" nella rete. Il Cache Engine è stato progettato dalla Cisco come una estensione di rete, per questo può essere gestito come un qualsiasi elemento di rete e questa caratteristica permette di inserirlo in rete in modo trasparente, non soltanto nel senso che la redirezione non è percepibile dagli utenti, ma anche che il router opera nella modalità usuale per il traffico non rediretto. Può adattarsi dinamicamente per prevenire i problemi che possono essere associati alle cache tradizionali, ad esempio, determinando quando permettere al traffico di bypassarlo, per prevenire problemi di overload e di autenticazione.
- È di facile utilizzo, grazie alla possibilità di gestirlo tramite le capability del sistema operativo Cisco IOS oppure, direttamente, tramite una interfaccia grafica, accessibile via web. Proprio per la presenza dell'interfaccia grafica, presenta una maggiore semplicità di utilizzo, rispetto alla prima soluzione di transparent caching adottata.

- Il Cache Engine può operare in "proxy mode" in modo da essere compatibile con reti che utilizzano proxy server. In altre parole, se abilitato ad operare come un proxy, può essere utilizzato anche in ambienti in cui non è abilitato il WCCP o in cui client sono stati precedentemente configurati per usare un proxy server. È inoltre possibile spedire tutto il traffico riguardante HTTP miss ad un proxy cache parent.
- Bilanciamento di carico. È possibile realizzare, insieme con altri Cache Engine, un cluster [Figura 4.10], permettendo di bilanciare e scalare, in modo lineare, il carico del traffico di richieste-risposte tra i componenti del cluster. Questa scalabilità lineare è realizzata grazie al modo in cui il router, in cui è abilitato il WCCP, redirige il traffico verso i Cache Engine. Il router esegue una funzione hash sull'indirizzo IP di destinazione delle richieste in arrivo, mappando le richieste in 256 raggruppamenti discreti (bucket). Statisticamente, questa funzione hash distribuisce equamente le richieste in arrivo tra tutti i bucket, che sono allocati tra tutti i componenti del cluster: il router fa in modo che ci sia sempre un particolare Cache Engine a servire le richieste per un certo indirizzo IP in Internet. In pratica, l'algoritmo di distribuzione basato sull'indirizzo IP di destinazione consente una distribuzione del carico tra tutti i componenti del cluster. Quando viene aggiunto un nuovo Cache Engine al cluster, il router rialloca i 256 bucket, prendendo in considerazione la nuova presenza. Ad esempio, l'installazione più semplice implica un router e un singolo Cache Engine, con assegnati tutti i 256 bucket. Se viene aggiunto un altro Cache Engine, il router farà la redirezione dei pacchetti verso entrambi: su ognuno dei Cache Engine saranno allocati 128 bucket. Se qualcuno dei Cache Engine del cluster fallisce, il cluster continua ad operare con un componente di meno e il router ridistribuisce equamente il carico di quel Cache Engine tra i rimanenti.

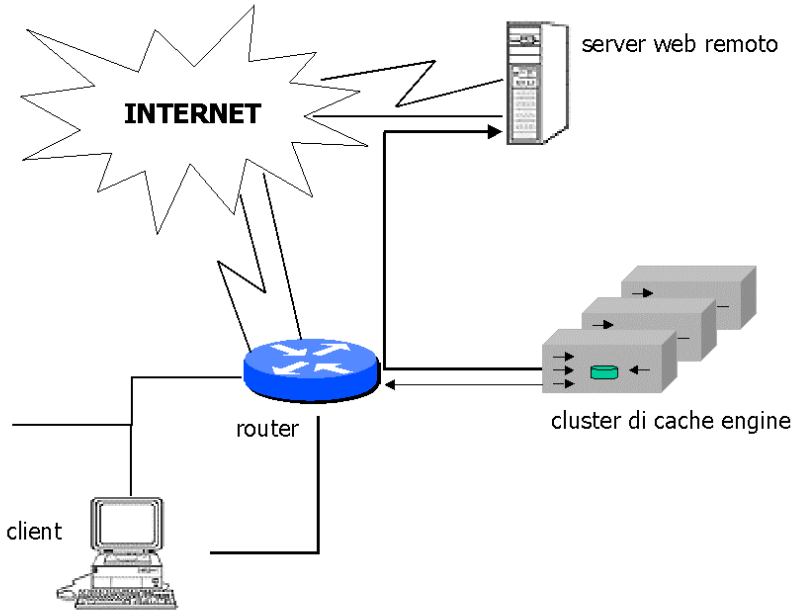

Figura 4.10 - Schema di Cluster composto da tre Cache Engine

- Load bypass. Se si verifica una inaspettata ondata di traffico, un cluster di Cache Engine potrebbe risultare sovraccarico. Per gestire questa situazione, ogni Cache Engine nel cluster può capire se è sovraccarico e, in questo caso, rifiuta richieste successive (*load bypass*) verso gli origin server, che risponderanno direttamente ai client. Lo stesso Cache Engine sarà, poi, in grado di riconoscere quando ha nuovamente le risorse necessarie per ricominciare ad accettare le richieste successive e, quindi, smettere di bypassarle. L'attivazione e disattivazione del load bypass sono determinate automaticamente dal carico della CPU e del file system. In questo modo, anche in condizioni di traffico inusuali, il cluster non introduce latenza eccessiva e mantiene l'accessibilità alla rete: se uno dei Cache Engine non risponde più a causa del sovraccarico, il router può escluderlo dal cluster e riallocare i bucket.
- Authentication bypass. Il Cache Engine ha una feature di bypass dinamico del client (*authentication bypass*) che gli permette di generare una lista di accessi dinamica per

queste coppie client-server e di connettere direttamente il client con l'origin server. Praticamente, se un client fa una richiesta di questo tipo, inizialmente, questa viene rediretta verso un Cache Engine, che, poiché non ha l'oggetto richiesto, deve fare la richiesta all'origin server. Quando riceve la richiesta, che necessita di un'autenticazione, il server spedisce in risposta un codice di errore, tra: 401 (unauthorized request), 403 (forbidden) o 503 (service unavailable). Come conseguenza, il Cache Engine attiverà il bypass dinamico e spedirà un messaggio HTTP al client, dicendogli di riprovare. Inoltre, conserverà (per un certo numero di minuti, che può essere configurato) l'informazione relativa alla coppia indirizzo IP di destinazione/indirizzo IP del client, cosicché i futuri pacchetti con questi due indirizzi IP saranno direttamente bypassati. Alla richiesta di reload del client, il Cache Engine scoprirà che la richiesta corrisponde ad una delle entry della lista di accessi di bypass, quindi la rifiuterà e la spedirà al router che invierà la richiesta (contenente l'indirizzo IP del client) direttamente all'origin server. L'authentication bypass è anche propagato, nel caso di caching gerarchico, nella gerarchia, verso l'alto e verso il basso [4].

4.5.2.2. Consistenza degli Oggetti

Il Cache Engine determina per quanto tempo mantenere gli oggetti "cacheati" nel disco, utilizzando dei fattori che determinano "freshness" di un oggetto:

- **Maximum TTL** Pone una soglia per gli expiration-time stimati. Il valore "Maximum Time-to-Live" configura il Cache Engine per considerare stale gli oggetti testuali (file HTML) e gli oggetti binari (embedded object), separatamente, dopo un dato numero di giorni (oppure ore, minuti o secondi), indipendentemente dalla data in cui l'oggetto è stato modificato l'ultima volta. Se un oggetto ha un esplicito expiration-time, questo ha la precedenza sul TTL configurabile.
- **Age Multiplier** Il Cache Engine può supporre la durata di vita di un oggetto

moltiplicando il tempo trascorso dalla data dell'ultima modifica dell'oggetto per una data percentuale (il valore assegnato a questo fattore, ad esempio, 30% per gli oggetti di testo e 60% per tutti gli altri), per derivare un expiration-time approssimato. Dopo questa data, l'oggetto è considerato stale.

La cache può, inoltre, rispondere ad un richiesta If-Modified-Since, servendola direttamente, senza rivalidarla con l'origin server, se l'oggetto richiesto ha una età minore di una percentuale (configurabile) del valore presente nell'header max-age dell'oggetto. I valori di default sono 80% per gli oggetti binari e 50% per gli oggetti testuali.

Quando l'oggetto scade, in base al suo valore di TTL, il Cache Engine fa una richiesta IMS (If-Modified-Since) all'origin server, la volta successiva in cui l'oggetto viene richiesto.

4.5.2.3. Configurazione

La notifica al router dell'esistenza del Cache Engine avviene automaticamente per mezzo di comunicazioni tra i due device [2]. Per questo è necessario abilitare il supporto del WCCP sul router, assicurandosi che la release del Cisco IOS includa il supporto per la versione 2 del protocollo. Il servizio può quindi essere abilitato specificando pochi semplici comandi:

```
ip wccp webcache
ip web-cache redirect out
```

A parte l'anonimizzazione dei file di log (opzione *sanitize*), per il rispetto della legge sulla privacy[64], è stata mantenuta la configurazione di default del sistema.

Nelle prime quattro settimane dell'esperimento, è stata abilitata l'opzione "Revalidate every request with origin server" nel menu "HTTP Freshness", che consente di verificare la validità di ogni oggetto richiesto (inserendo il campo If-Modified-Since nel request-header) [Figura 4.11].

[64] Ved. nota 46.

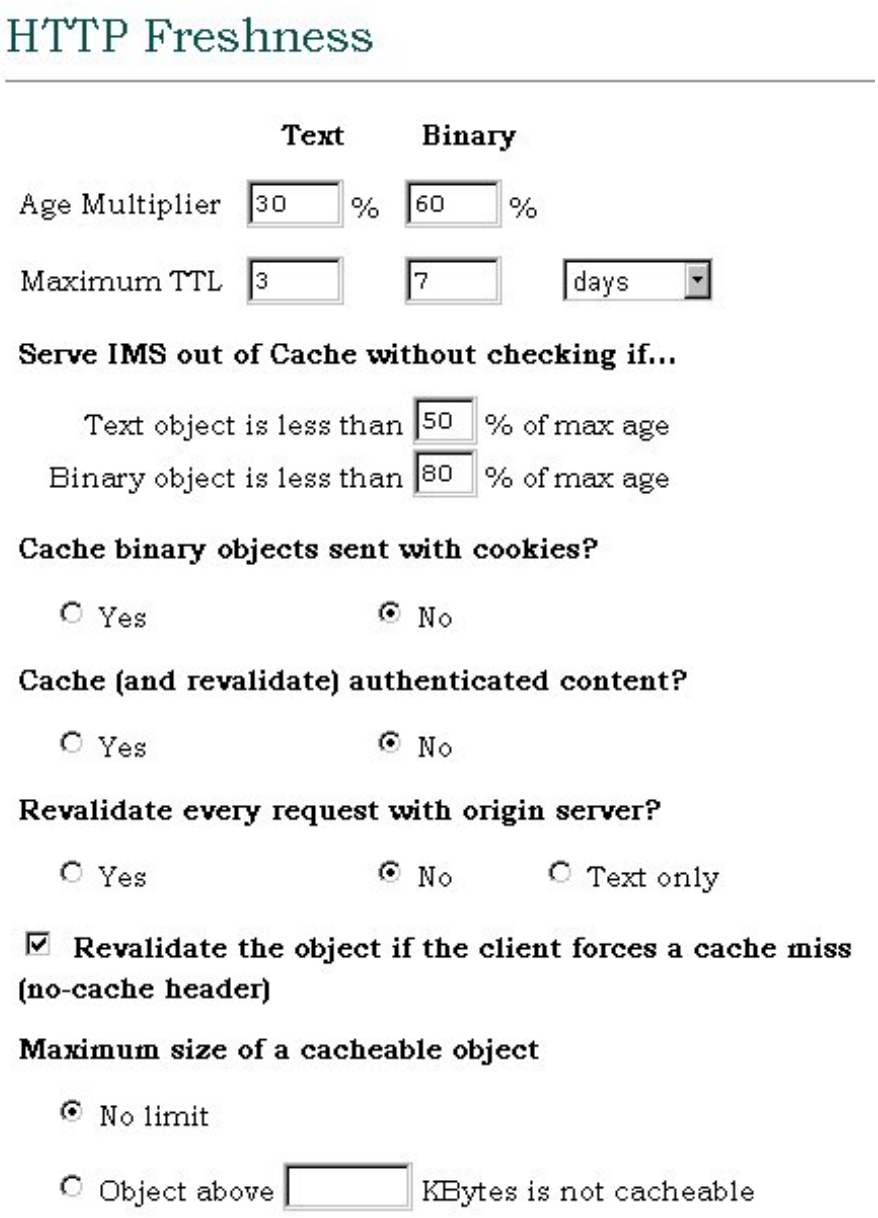

Figura 4.11 - Interfaccia di Configurazione del CE 505 (menu HTTP Freshness)

4.5.3. Analisi dei Dati

Le misurazioni sono state effettuate dopo un periodo di quattro settimane dall'attivazione del servizio, e, successivamente, dopo altre quattro. Grazie all'interfaccia grafica presente, le statistiche sono direttamente disponibili, senza la necessità di elaborare i file di log che, in ogni

caso, possono essere esportati su altri server (fino a 4) per essere analizzati in un momento successivo.

Nell'analisi dei dati, va considerato il fatto che il Cache Engine era inizialmente vuoto, per cui un basso numero di hit iniziale dipende dal periodo di transizione in cui la cache è andata popolandosi. Dopo le prime 5/6 settimane, la cache si è stabilizzata con un 89% di disco pieno [Figura 4.12], per cui le successive misurazioni sono state più realistiche rispetto ad un'attività a regime.

Disk Stats

Current Sun Jan 14 07:10:46 2001 GMT
Uptime 6 weeks, 2 days, 21 hours, 44 minutes, 48 seconds

volume /c0t0d0s3

Data Bytes Max	7006158 KB			
Data Bytes used	6282636 KB (89% full)			
Disk Wraps	1 Read errors	0	Write errors	0
Inode load errors	0 Inode Hits	429947	Inode misses	928648
Attribute load errors	0 Object truncations	36144	Truncated object flushes	36144

Figura 4.12 - Statistiche del disco del Cache Engine dopo sei settimane

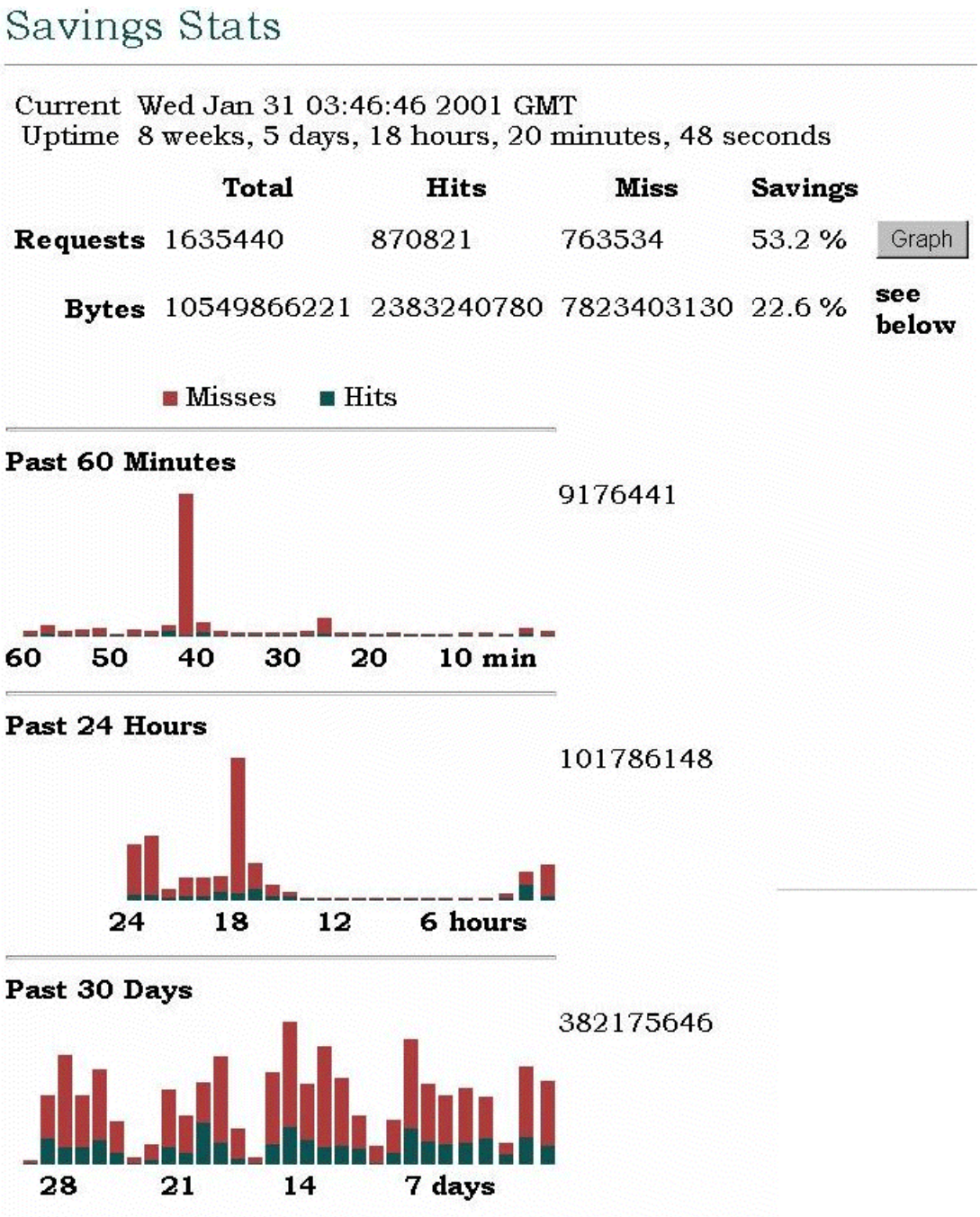

Figura 4.13 - Statistiche del Cache Engine, Byte Hit Rate

Dopo le prime quattro settimane di sperimentazione il traffico totale rediretto verso il Cache Engine è risultato 4,7GB, di cui 865MB sono stati serviti direttamente dalla cache, con una percentuale di byte hit rate del 18,4%, corrispondente al 46,8% di richieste. Dopo altre 4

settimane di attività e un maggiore popolamento del disco, la percentuale di byte hit rate è salita al 22,6% [Figura 4.13] con il 53,2% di richieste servite direttamente dalla cache [Figura 4.14]. I valori in alto a destra nei grafici delle Figure 4.13 e 4.14 indicano i valori massimi in termini di byte salvati e richieste servite in intervalli di 60 minuti, 24 ore e 30 giorni.

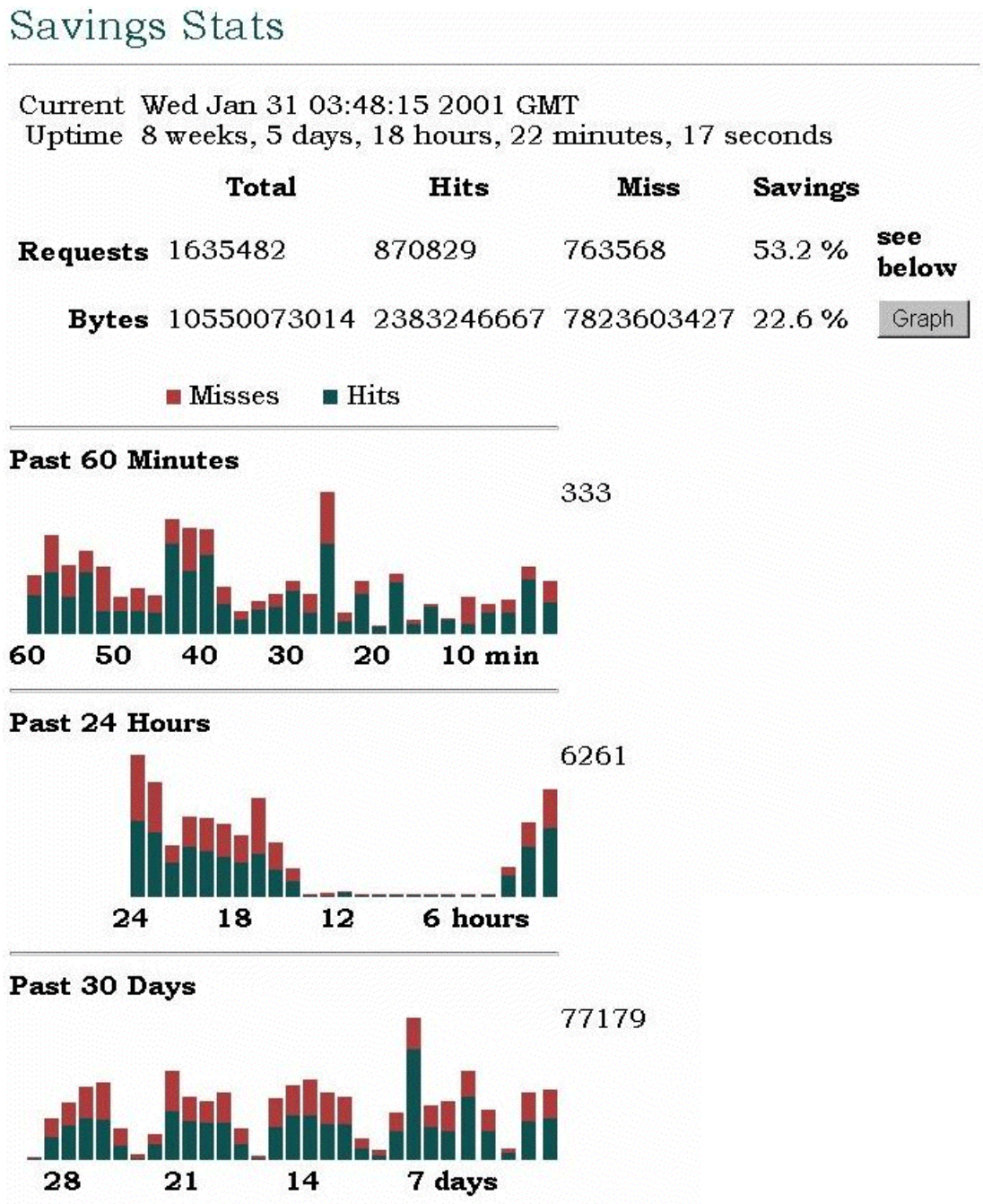

Figura 4.14 - Statistiche del Cache Engine, Richieste HTTP

Rispetto al primo esperimento, è stato rilevato un aumento del traffico web dovuto alla presenza di nuovi utenti e ad una effettiva crescita del traffico (rispetto al numero di utenti). Nella configurazione non è stato posto alcun limite sulla dimensione degli oggetti "cacheabili", mentre nella configurazione di Squid il limite era stato fissato a 8 MB.

4.6. Confronto dei Risultati

Nel seguito viene effettuato un breve confronto tra i risultati ottenuti nei due esperimenti precedentemente descritti.

4.6.1. Efficacia del Servizio

Se confrontiamo i risultati dei due esperimenti in termini di numero di richieste e di byte serviti direttamente dalla cache, possiamo notare delle differenze. Un confronto esatto tra le due soluzioni non è possibile, perché i cache server utilizzati sono diversi, sia in termini di risorse, sia nelle configurazioni. Comunque le opzioni di configurazione adottate permettono di interpretare correttamente i risultati.

Nella prima parte dell'esperimento, con una redirezione statica e con l'utilizzo del cache server Squid, è stato ottenuto il 41,19% di richieste servite dalla cache, corrispondenti al 19,23% di byte hit rate, nel periodo di popolamento della cache. Successivamente, a regime, è stato osservato il 42,45% di request hit rate corrispondenti al 20,70% di byte hit rate.

Con l'utilizzo della redirezione dinamica e del Cisco Cache Engine, è stato ottenuto il 46,80% di request hit rate e il 18,40% di byte hit rate, nella fase di popolamento ed il 53% di richieste servite direttamente dalla cache, corrispondenti al 22,6% di byte, a regime.

La seconda configurazione, rispetto alla prima, ha un byte hit rate minore nella fase di popolamento e maggiore nel seguito. In realtà questo comportamento non è anomalo, ma dipende dalle impostazioni nella configurazione del Cache Engine. Nelle prime quattro

settimane di attività, infatti, il Cache Engine è stato configurato per richiedere una validazione degli oggetti ad ogni richiesta. Questo corrisponde ad una GET condizionale (viene aggiunto il campo If-Modified-Since dell'header), quindi comporta un trasferimento di dati (nel caso di modifica della risorsa viene restituita l'intera entità, altrimenti vengono restituiti i soli metadati). Questa configurazione influisce sul numero di byte richiesti e ricevuti dall'origin server, ma garantisce di restituire al client copie consistenti degli oggetti richiesti. Nei risultati a regime, invece, è stata applicata la configurazione di default del Cache Engine, che utilizza un algoritmo di refresh, basato sul tempo trascorso dalla data dell'ultima modifica dell'oggetto, per derivare un expiration-time approssimato[65]. Naturalmente, decidendo di utilizzare i valori di default del Cache Engine (senza validazione ad ogni richiesta) si avrà un incremento del byte hit rate, al costo della possibilità di restituire oggetti stale ai client. Il controllo della consistenza su ogni oggetto, di contro, utilizza maggiore banda. Considerando anche il fatto che, nella seconda fase dell'esperimento, il traffico rediretto era maggiore, i risultati ottenuti sono quindi paragonabili.

Duska et al. [20] hanno osservato che (1) gli oggetti più piccoli tendono ad avere hit rate superiore a quello di oggetti di maggiori dimensioni e che (2) in una gerarchia di cache il numero di byte hit tende a crescere ulteriormente, dovuto al maggior numero di oggetti condivisi. Per questo, ci aspettiamo ulteriori benefici quando (e se) la cache sarà inserita nella gerarchia del servizio nazionale di proxy cache server del GARR e la redirezione del traffico sarà estesa al servizio dell'intera Area di Ricerca di Pisa.

4.6.2. Problemi

I limiti della soluzione statica, già noti, sono stati verificati anche in prima persona. La redirezione statica non è fail-safe, e, infatti, durante la sperimentazione abbiamo avuto un fermo del cache server perché, in seguito ad una mancata erogazione della corrente, il sistema non è

[65] § 4.5.2

ripartito. Una semplice impostazione del BIOS ci ha permesso di superare il problema, ma gli utenti hanno perso la connettività HTTP per un certo intervallo di tempo che, se pur piccolo, ha portato ad un disservizio. Considerando che il cache server (nel nostro caso un semplice PC) può essere soggetto a guasti in qualsiasi momento, l'adozione di tale soluzione non è raccomandabile. Tra l'altro, l'utilizzo di un protocollo di comunicazione tra cache e router (come nella redirezione dinamica) comporta anche tutti gli altri vantaggi descritti nei paragrafi precedenti (scalabilità, bilanciamento di carico, etc.).

Per completezza, dobbiamo dire che tutti i vantaggi della soluzione dinamica sono ugualmente ottenibili utilizzando uno switch "intelligente", ma nel nostro specifico contesto, l'implementazione di tali funzionalità a livello di router ha eliminato la necessità di investimenti in device aggiuntivi.

Durante la prima fase della sperimentazione, è stato incontrato un secondo problema, legato all'utilizzo di Real Player. Esaminando i file di log del cache server è stato notato un elevato numero di richieste che generavano il codice di errore NONE/400 (client bad request) [App. A]. Queste richieste erano originate da un processo di Real Player che girava in background e, ad intervalli regolari di pochi secondi, inviava richieste HTTP per una URL non esistente (per aggiornare le headlines). Il problema è stato risolto installando la nuova versione del prodotto e i file di log sono stati "ripuliti" da queste entry per non falsare le statistiche.

Uno dei principali inconvenienti nella redirezione statica è la gestione del bypass del traffico di client che accedono a servizi ad accesso controllato, come library on-line. Se un client deve accedere ad un servizio che richiede un'autenticazione basata su indirizzi IP, il server che ospita il sito web deve vedere l'indirizzo IP del client e non quello di una eventuale cache interposta tra essi. Mascherando la reale identità del client, un transparent cache server impedisce l'autenticazione. In questo caso, il traffico originato da client autorizzati deve essere instradato direttamente verso l'origin server. Nella prima parte dell'esperimento abbiamo dovuto

esplicitamente escludere il flusso di dati di queste coppie client-server dalla redirezione del traffico, inserendo "a mano" le impostazioni nella configurazione del router. Con la seconda soluzione, il Cache Engine automaticamente crea e gestisce ACL (Acces Control List) temporanee[66], che dopo un certo intervallo di tempo saranno rimosse, cosicché i client possano continuare ad utilizzare la cache al di fuori di queste attività ad accesso controllato. In tal modo il vantaggio di una gestione automatica delle liste (che in grande organizzazioni può comportare un considerevole carico amministrativo), è unito ad un utilizzo più ampio del servizio.

4.6.3. Grado di Accettazione del Servizio

A dispetto di qualche resistenza iniziale, il servizio è stato ampiamente accettato. Solo pochi utenti si sono preoccupati che le loro transazioni di rete potessero essere controllate, ma queste ritrosie sono state facilmente superate garantendo che i file di log sarebbero stati resi anonimi e gli indirizzi IP delle stazioni mascherati.

4.6.4. Strumenti di Amministrazione del Servizio

Il Cache Engine offre delle interfacce via web che consentono di semplificare la configurazione, gestione e monitoraggio quotidiano del servizio. Questo, sicuramente, comporta dei vantaggi, specie per persone meno esperte, poiché le interfacce semplificano la gestione delle configurazioni, le statistiche sono già disponibili, etc. Da un altro punto di vista, comunque, essendo le interfacce limitate nelle scelte che il form propone, possono risultare meno flessibili sia nelle opzioni di configurazione sia nella elaborazione delle statistiche.

È comunque possibile una configurazione da linea di comando che offre un maggiore range di opzioni.

---

[66] § 4.5.2.1

5. Conclusioni e Sviluppi Futuri

Questo lavoro di tesi si basa su una sperimentazione della tecnologia di transparent web caching svolta presso l'Istituto per le Applicazioni Telematiche (IAT) del CNR.

Come fase preliminare, il lavoro ha richiesto uno studio approfondito della tecnologia web, con particolare attenzione verso il protocollo HTTP, ed un approfondimento della tecnologia di web caching nelle sue configurazioni, architetture, protocolli e algoritmi per la consistenza e validazione degli oggetti.

La sperimentazione comprende la redirezione **via router** del traffico HTTP, generato dalla rete locale dello IAT, verso un cache server, utilizzando due differenti configurazioni:

- redirezione **statica** del traffico HTTP verso un PC su cui è attivo Squid, un cache server di pubblico dominio sviluppato dai laboratori NLANR (primo esperimento);
- redirezione **dinamica** (secondo esperimento) del traffico verso un Cache Engine della Cisco (CE 505), un elemento di rete dedicato a queste funzionalità, utilizzando la versione 2 del protocollo WCCP (Web Cache Coordination Protocol).

Non è stato possibile utilizzare Squid, anche nel secondo esperimento, perché ad oggi, esso supporta solo la versione 1 del WCCP che presenta problemi di efficienza lato router e, per questo, non può essere utilizzata in un ambiente operativo (altrimenti tutte le comunicazioni Internet sarebbero penalizzate). Come conseguenza, i risultati dei due esperimenti non sono immediatamente confrontabili, ma devono essere analizzati alla luce delle differenti caratteristiche dei cache server utilizzati.

Entrambe le redirezioni sono state attivate abilitando le rispettive configurazioni sul Cisco 7505 presente nella LAN dell'Area di Ricerca del CNR di Pisa.

L'obiettivo primario del lavoro è stato sperimentare e valutare la tecnologia in un ambiente di produzione. Sebbene la comunità dello IAT sia piuttosto piccola ed omogenea, il che ha avuto effetto su alcune delle variabili misurate, l'analisi dei dati e il feedback degli utenti ha confermato l'utilità del servizio.

I risultati[67] hanno mostrato che:

- La qualità del servizio è migliorata con una metà degli Internet object serviti dalla cache. In generale la latenza e il tempo di download delle pagine web sono stati ridotti risultando in un miglioramento percepibile del servizio. Non è stata osservata alcuna variazione rilevante nella qualità del servizio per oggetti (miss) scaricati dagli origin server.
- È stato ottenuto un piccolo ma comunque importante risparmio, nella banda corrispondente, nella configurazione più restrittiva, ad un quinto del traffico HTTP rediretto.
- L'utilizzo del servizio ha ridotto il traffico non necessario inviato in Internet, diminuendo così il carico sui web server più popolari, ed anche contribuendo a limitare la congestione della rete Internet.

La sperimentazione è stata effettuata nelle normali condizioni di traffico della rete locale dello IAT, quindi non è stato possibile analizzare le caratteristiche dei cache server in condizioni di "stress". Per un approfondimento sulle caratteristiche dei vari cache server, commerciali e freeware, si rimanda alle interessanti valutazioni (Cache-offs http://cacheoff.ircache.net/)[68] dei laboratori dell'NLANR.

---

[67] § 4.6

[68] "Cache-offs caching products competition" effettua il testing di prodotti simili di diversi costruttori, sotto le medesime condizioni, in un breve periodo di tempo e, usualmente, nella stessa locazione, in modo da confrontare i risultati.

Dal punto di vista della tecnologia utilizzata, la redirezione dinamica è certamente preferibile a quella statica, risultando affidabile, scalabile (con bilanciamento di carico) e flessibile (load bypass e authentication bypass)[69].

Gli inconvenienti incontrati nel primo esperimento sono sostanzialmente legati all'utilizzo della redirezione statica (mancato restart del cache server, gestione manuale delle lista di authentication bypass), anche se uno specifico problema si è verificato per l'utilizzo di una versione di Real Player contente un bug[70]. Nessun problema è stato osservato durante il secondo esperimento.

Dopo il primo momento di diffidenza, il servizio è stato largamente accettato dagli utenti, dietro assicurazione che i file di log sarebbero stati anonimizzati nel rispetto della legge sulla privacy.

I risultati di questo lavoro saranno utilizzati per progettare la ristrutturazione del servizio di web caching del CNR, nell'ottica del "Servizio Web Cache Nazionale GARR"[71]. Una integrazione dei cache server di secondo livello con quelli della dorsale fornita dal GARR consentirebbe, infatti, di migliorare il servizio in termini di crescita di hit rate e di diminuzione della latenza.

Anche se la velocità delle linee di comunicazione è in continua crescita ed il GARR ha in previsione la migrazione a connessioni ad altissima velocità (dell'ordine dei Gbps), la tecnologia rimane valida, poiché "avvicinando" i contenuti web agli utenti, la latenza diminuisce e migliora la qualità del servizio. Naturalmente, in un simile scenario, è necessario dotarsi di sistemi in grado di servire richieste con un'adeguata velocità, altrimenti si rischia di creare un collo di bottiglia nella rete.

---

[69] § 4.5.2.1

[70] § 4.6.2

[71] § 4.2

D'altro canto, l'utilizzo della tecnologia di caching (HTTPd accelerator)[72] all'interno delle Content Delivery Network[73], è una ulteriore conferma dell'importanza della distribuzione di copie di contenuto nell'ottica sia del risparmio delle risorse di rete e dei server, sia per il miglioramento della qualità del servizio fornito agli utenti.

[72] § 2.4.1

[73] § 2.5

6. Appendice A

6.1. Codici di risposta HTTP

```
Status-Code    =
 "100" ; Continue
| "101"  ; Switching Protocols
| "200"  ; OK
| "201"  ; Created
| "202"  ; Accepted
| "203"  ; Non-Authoritative Information
| "204"  ; No Content
| "205"  ; Reset Content
| "206"  ; Partial Content
| "300"  ; Multiple Choices
| "301"  ; Moved Permanently
| "302"  ; Found
| "303"  ; Other
| "304"  ; Not Modified
| "305"  ; Use Proxy
| "307"  ; Temporary Redirect
| "400"  ; Bad Request
| "401"  ; Unauthorized
| "402"  ; Payment Required
| "403"  ; Forbidden
| "404"  ; Not Found
```

```
        | "405"  ; Method Not Allowed
        | "406"  ; Not Acceptable
        | "407"  ; Proxy Authentication Required
        | "408"  ; Request Time-out
        | "409"  ; Conflict
        | "410"  ; Gone
        | "411"  ; Length Required
        | "412"  ; Precondition Failed
        | "413"  ; Request Entity Too Large
        | "414"  ; Request-URI Too Large
        | "415"  ; Unsupported Media Type
        | "416"  ; Requested range not satisfiable
        | "417"  ; Expectation Failed
        | "500"  ; Internal Server Error
        | "501"  ; Not Implemented
        | "502"  ; Bad Gateway
        | "503"  ; Service Unavailable
        | "504"  ; Gateway Time-out
        | "505"  ; HTTP Version not supported
        | extension-code
extension-code = 3DIGIT
```

7. Bibliografia

[1] **AA. VV.** *CAIDA Cflowd* - Febbraio 2001- http://www.caida.org/tools/measurement/cflowd/

[2] **AA. VV.** *Cisco Cache Engine User Guide, Version 2.1.0* - Cisco Systems Inc.- 2001 - http://www.cisco.com/univercd/cc/td/doc/product/iaabu/webcache/ce21/ver21/

[3] **AA. VV.** *Navigator Proxy Auto-Config File Format* - Marzo 1996 - http://home.netscape.com/eng/mozilla/2.0/relnotes/demo/proxy-live.html

[4] **AA. VV**. *Release Notes for Cisco Content Engine Software, Release 2.2.0* - Cisco Systems, Inc. - 2000 - http://www.cisco.com/univercd/cc/td/doc/product/iaabu/webcache/rn_ce220.htm

[5] **AA. VV**. *Servizio Web Cache Nazionale GARR - 2001* - http://www.cache.garr.it/

[6] **AA. VV.** *Web Caching - Issues* - Ottobre 1997 - http://www.ircache.net/Cache/FAQ/ircache-faq-7.html

[7] **L. Abba, M. Buzzi, D. Pobric M. Ianigro** - *Introducing Transparent Web Caching in a Local Area Network* - Computer Mesurament Group (CMG) , Orlando USA - Dicembre 2000 -

[8] **L. Abba, M. Buzzi, F. Gennai, M. Ianigro** - *Free Software per Web Caching, Convegno Annuale AICA* – Abano Terme (PD) 26-29 Settembre 1999, pp. 572-580.

[9] **T. Berners-Lee, R. Fielding, UC Irvine, H. Frystyk** - *HyperText Transfer Protocol - HTTP/1.0, RFC 1945* - Maggio 1996 - http://www.ietf.org/rfc/rfc1945.txt

[10] **T. Berners-Lee R. Fielding U.C. Irvine L. Masinter** - *Uniform Resource Identifiers (URI): Generic Syntax, RFC 2396* - Agosto 1998 - http://www.ietf.org/rfc/rfc2396.txt

[11] **M. Buzzi** - *Il servizio Proxy Cache Server del CNR* – Febbraio 1999 - http://www.iat.cnr.it/~cachemgr/CacheServer/

[12] **B. Cain, G. Tomlinson, S. Thomas** - *CDN Peering Architectural overview*, *Internet Draft* - Novembre 2000 - http://www.ietf.org/internet-drafts/draft-green-cdnp-gen-arch-02.txt

[13] **L. Cherkasova, M. DeSouza, S. Ponnekanti** - *Performance Analysis of "Content-Aware" Load Balancing Strategy FLEX: Two Case Studies* - Proceedings of the 34th Hawaii International Conference on System Sciences - 2001

[14] **M. Cieslak, D. Forster** - *Web Cache Coordination Protocol V1.0, Internet draft* - Giugno 1999 - http://www.wrec.org/Drafts/draft-ietf-wrec-web-pro-00.txt

[15] **M Cieslak, D Forster** - *Web Cache Coordination Protocol V2.0, Internet draft* - Luglio 2000 - http://www.wrec.org/Drafts/draft-wilson-wrec-wccp-v2-00.txt

[16] **M. Conti, E. Gregori, F. Panzieri** - *QoS-based Architectures for Geographically Replicated Web Server*s - Cluster Computing Journal, Baltzer (to appear).

[17] **I. Cooper I. Melve G. Tomlinson** - *Internet Web Replication and Caching Taxonomy, RFC 3040* -Gennaio 2001 - ftp://ftp.rfc-editor.org/in-notes/rfc3040.txt

[18] **P. Dazing** - *NetCache architepture and deployment* - Computer Network and ISDN Systems - Novembre 1998 http://wwwcache.ja.net/events/workshop/01/NetCache-3_2.pdf

[19] **J. Dilley, I. Cooper** - *Known HTTP Proxy/Caching Problems, Internet Draft* - Novembre 2000 -http://search.ietf.org/internet-drafts/draft-ietf-wrec-known-prob-03.txt

[20] **B.M. Duska, D. Marwood, M. J. Feeley** - *The Measured Access Characteristics of World-Wide-Web Client Proxy Caches - USENIX Symposium on Internet Technologies and Systems* - Dicembre 1997 - http://www.usenix.org/publications/library/proceedings/usits97/full_papers/duska/duska_html/duska.html

[21] **K. Egevang P. Francis** - *The IP Network Address Translator (NAT) - RFC 1631*- Maggio 1994 - http://www.ietf.org/rfc/rfc1631.txt

[22] **R. Fielding, U.C. Irvine, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee** - *Hypertext Transfer Protocol - HTTP/1.1, RFC 2616* - Giugno 1999 - http://www.ietf.org/rfc/rfc2616.txt

[23] **P. Gauthier, J. Cohen, M. Dunsmuir, C. Perkins** - *Web Proxy Auto-Discovery Protocol, Internet Draft* - Luglio 1999 - http://www.wrec.org/Drafts/draft-ietf-wrec-wpad-01.txt

[24] **M. Grabnar** - *Squid and MRTG: to SNMP or not SNMP*? - Febbraio 1999 http://www.terena.nl/d2-workshop/d2cache2000/lecture.html

[25] **M. Hamilton, A. Rousskov, D. Wessels** - *Cache Digest specification, version 5* - Dicembre 1998 - http://squid.nlanr.net/Squid/CacheDigest/cache-digest-v5.txt

[26] **Johnson et al.** - *The Measured Performance of Content Distribuition Networks* - 5th Internetional Web Caching and Content Delivery Workshop, 22-24 Maggio 2000

[27] **Kangasharju et al.** - *Performance Evaluation of Redirection Schemes in Content Distribuition Networks* - 5th Internetional Web Caching and Content Delivery Workshop, 22-24 Maggio 2000

[28] **D. Li P. Cao M. Dahlin** - *WCIP: Web Cache Invalidation Protocol*, *Internet Draft* - Novembre 2000 - http://search.ietf.org/internet-drafts/draft-danli-wrec-wcip-00.txt

[29] **I. Melve** - *Inter Cache Communication Protocol, Internet draft* - Novembre 1998 - http://www.wrec.org/Drafts/draft-melve-intercache-comproto-00.txt

[30] **J. Niedertst** - *Web Design in a Nutshell* - Gennaio 1999 - ed. ÒReilly

[31] **T. Oetiker et al.** - *Multi Router Traffic Grapher* - http://ee-staff.ethz.ch/~oetiker/webtools/mrtg/mrtg.html

[32] **J. Palme A. Hopmann** - *MIME E-mail Encapsulation of Aggregate Documents, such as HTML (MHTML) RFC 2110* - http://www.ietf.org/rfc/rfc2110.txt

[33] **W. R. Stevens** - *TCP/IP Illustrated, Volume 1 The Protocols* - ed. Addison-Wesley 1994

[34] **M. StJohns** - *Authentication Server, RFC 931* - Gennaio 1985 - http://www.ietf.org/rfc/rfc0931.txt

[35] **A. S. Tanenbaum** - *Computer Networks* - ed. Prentice-Hall 1998

[36] **V. Valloppillil, K. W. Ross** - *Cache Array Routing Protocol v1.0, Internet draft* - Febbraio 1998 - http://www.ietf.org/proceedings/99mar/I-D/draft-vinod-carp-v1-03.txt

[37] **P. Vixie, S. Thomson, Y. Rekhter, J. Bound** - *Dynamic Updates in the Domain Name System (DNS UPDATE), RFC 2136* - Aprile 1997 - http://www.ietf.org/rfc/rfc2136.txt

[38] **D. Wessels** - *Squid Frequently Asked Questions* - 2001 - http://squid.nlanr.net/Squid/FAQ/FAQ.html

[39] **D. Wessels, K. Claffy** - *Application of Internet Cache Protocol (ICP), version 2, RFC 2187* - Settembre 1997 - http://squid.nlanr.net/Squid/rfc2187.txt

[40] **D. Wessels, K. Claffy** - *Internet Cache Protocol (ICP), version 2* - Settembre 1997 - http://squid.nlanr.net/Squid/rfc2186.txt

[41] **D. Wessels P. Vixie,** - *Hyper Text Caching Protocol (HTCP/0.0), Internet draft* - Settembre 1998 - http://www.wrec.org/Drafts/draft-vixie-htcp-proto-04.txt

[42] **B. Williams** - *Transparent Web Caching Solution* - Alteon Networks - Aprile 1998 - http://wwwcache.ja.net/events/workshop/33/cachpaper.html

[43] **C. E. Wills M. Mikhailov** - *Examining the Cacheability of User-Requested Web Resources , WCW99* - San Diego, USA