ARCHITETTURE DELLE APPLICAZIONI DI RETE

Prima di occuparci della codifica software è necessario disporre di un progetto dettagliato dell'architettura dell'applicazione, completamente diversa dall'architettura di rete (per esempio, l' architettura di Internet a cinque livelli) . Per lo sviluppatore l'architettura di rete è fissata e fornisce alle applicazioni uno specifico insieme di servizi; il suo compito è progettare l'architettura dell'applicazione e stabilire la sua organizzazione sui vari sistemi periferici. Nella scelta lo sviluppatore si baserà probabilmente su una delle due principali architetture attualmente utilizzate: l' architettura client-server o P2P.

Nell'architettura **client-server** vi è un host sempre attivo (IP permanente), chiamato server, che risponde alle richieste di servizio di molti altri host, detti client (client, processi in dispositivi terminali che richiedono servizi, possono essere attivi in determinati momenti e inattivi in altri-> IP dinamici).

Un esempio classico è rappresentato dall'applicazione web, in cui un web server, sempre attivo, risponde alle richieste dei browser in funzione sui client. Quando riceve una richiesta di un oggetto da parte di un client, il server risponde inviandolo. Si osservi che nell'architettura client-server i client non comunicano direttamente tra loro; così, in una applicazione web, i browser non interagiscono direttamente tra loro. Inoltre il server dispone di un indirizzo fisso, diffusamente conosciuto, detto indirizzo IP (che tratteremo pre sto). Il client può quindi contattare il server in qualsiasi momento, inviandogli un pacchetto. Tra le più note applicazioni con architettura client-server, ricordiamo il Web, il trasferimento dei file con FTP, Telnet e la posta elettronica.

Spesso in un'applicazione client-server un singolo host che esegue un server non è in grado di rispondere a tutte le richieste dei suoi client. Per questo motivo nelle architetture client server si usano i data center, che ospitano molti host, per creare un potente server virtuale (un data center può ospitare fino a centinaia di migliaia di server).

In un'**architettura P2P** l'infrastruttura di server in data center è minima o del tutto assente; si sfrutta, invece, la comunicazione diretta tra coppie arbitrarie di host, chiamati peer (ossia pari), collegati in modo intermittente. I peer non appartengono a un fornitore di servizi, ma sono computer fissi e portatili, controllati dagli utenti, che per la maggior parte si trovano nelle abitazioni, nelle università e negli uffici. Dato che i peer comunicano senza passare attraverso un server specializzato, l'architettura viene detta peer-to-peer. Molte tra le applicazioni attualmente più diffuse e con elevata intensità di traffico sono basate su un'architettura P2P. Queste applicazioni includono la condivisione di file (per esempio, BitTorrent), sistemi per aumentare le prestazioni nel trasferimento di file (per esempio, Xunlei), la telefonia su Internet (per esempio, Skype) e IPTV (per esempio, Kankan e PPstream).

Abbiamo accennato che alcune applicazioni presentano un' architettura ibrida, combinando sia elementi client-server sia P2P. Per esempio, per molte applicazioni di messaggistica istantanea, i server sono usati per tenere traccia degli indirizzi IP degli utenti, ma i messaggi tra utenti sono inviati direttamente tra gli host degli utenti, senza passare attraverso server intermedi (es Skype). Uno dei punti di forza dell'architettura P2P è la sua intrinseca scalabilità. Per esempio, in un'applicazione di condivisione dei file P2P, ogni peer, sebbene generi carico di lavoro richiedendo dei file, aggiunge anche capacità di servizio al sistema, rispondendo alle richieste di altri peer (posso identificare quali client hanno a disposizione il contenuto che cerco e non sempre tramite un server, i dispositivi terminali comunicano direttamente tra loro e sono connessi in modo intermittente-> cambiano sempre indirizzo IP).

PROCESSI COMUNICANTI

Prima di costruire un' applicazione di rete dovrete anche conoscere come comunicano tra loro i programmi in esecuzione su diversi sistemi terminali. Nel gergo dei sistemi operativi non si parla in effetti di programmi, ma di processi comunicanti. Si può pensare a un processo come a un programma in esecuzione su un sistema. Processi in esecuzione sullo stesso sistema comunicano utilizzando un approccio interprocesso (interprocess communication). Le regole di questo tipo di comunicazione sono go vernate dal sistema operativo del calcolatore in questione. Ma in questo libro non sia mo interessati a come comunicano i processi all'interno dello stesso host, bensì a co me comunicano i processi in esecuzione su sistemi diversi, che potrebbero anche avere sistemi operativi diversi.

I processi su due sistemi terminali comunicano scambiandosi messaggi attraverso la rete: il processo mittente crea e invia messaggi nella rete e il processo destinatario li riceve e, quando previsto, invia messaggi di risposta. La Figura 2. 1 mostra come i processi comunicano utilizzando il livello di applicazione della pila a cinque livelli dei protocolli Internet.

PROCESSI CLIENT-SERVER

Le applicazioni di rete sono costituite da una coppia di processi che si scambiano messaggi su una rete. Per ciascuna coppia di processi comunicanti, generalmente ne etichettiamo uno come client e l'altro come server.

“Nel contesto di una sessione di comunicazione tra una coppia di processi, quello che avvia la comunicazione (cioè contatta l’altro processo all’inizio della sessione) è indicato come **client**, mentre quello che attende di essere contattato per iniziare la sessione è detto **server**”. Nel Web un processo browser avvia il contatto con un processo web server; quindi il primo è il client e il secondo il server. Nella condivisione di file P2P, quando il per A chiede al peer B di inviare un dato file, il primo rappresenta il client e il secondo il server, nel contesto di questa specifica sessione di comunicazione.

Un processo invia messaggi nella rete e riceve messaggi dalla rete attraverso un'interfaccia software detta **socket**. Una socket è *l'interfaccia tra il livello di applicazione e il livello di trasporto* (es. con TCP come protocollo di trasporto) all'interno di un host.

Come nella posta tradizionale, affinché la consegna possa essere effettuata i processi riceventi devono averne un indirizzo per ricevere i messaggi inviati da un processo in esecuzione su un altro host. Per identificare il processo ricevente, è necessario specificare due informazioni: (1) l'indirizzo dell'host e (2) un identificatore del processo ricevente sull'host di destinazione.

In Internet, gli host vengono identificati attraverso i loro indirizzi IP (un numero di 32 bit che possiamo pensare identifichi univocamente l'host) e oltre a conoscere l'indirizzo dell'host cui è destinato il messaggio, il mittente deve anche identificare il processo destinatario, più specificatamente la socket che deve ricevere il dato. Questa informazione è necessaria in quanto, in generale, sull'host potrebbero essere in esecuzione molte applicazioni di rete. Un numero di **porta** di destinazione assolve questo compito.

SERVIZI DI TRASPORTO DISPONIBILI PER LE APPLICAZIONI

Ricordiamo che una socket è l'interfaccia tra un processo applicativo e il protocollo a livello di trasporto. L'applicazione lato mittente spinge fuori i messaggi tramite la socket; dall'altra parte, lato ricevente, il protocollo a livello di trasporto ha la responsabilità di consegnare i messaggi alla socket del processo ricevente. Molte reti, Internet inclusa, mettono a disposizione vari protocolli di trasporto. Nel progetto di un' applicazione occorre scegliere il protocollo a livello di trasporto. Occorre valutare i servizi resi disponibili dai protocolli a livello di trasporto e scegliere quello che

fornisce i servizi più confacenti all' applicazione. Li possiamo classificare a grandi linee secondo quattro dimensioni: trasferimento dati affidabile, throughput, temporizzazione e sicurezza.
Trasferimento dati affidabile: in una rete di calcolatori i pacchetti possono an dare perduti. Per esempio, un pacchetto può far traboccare un buffer in un router o può essere scartato da un host o da un router in quanto alcuni suoi bit sono corrotti. In alcune applicazioni - quali posta elettronica, messaggistica istantanea, trasmissione di file, accesso a host remoti, invio di documenti web e applicazioni finanziarie - la perdita di informazioni potrebbe causare gravi conseguenze. Quindi, per supportare queste applicazioni occorre garantire che i dati inviati siano consegnati corretti e completi. Se un protocollo fornisce questo tipo di servizio di consegna garantita dei dati, si dice che fornisce un *trasferimento dati affidabile.* (Può essere accettabile che un protocollo a livello di trasporto non fornisca questo servizio ad es per le app multimediali audio/video poiché possono tollerare le perdite).
Throughput: nel contesto di una sessione di comunicazione tra due processi lungo un percorso in rete, è il tasso al quale il processo mittente può inviare i bit al processo ricevente. Dato che altre sessioni condivideranno la banda sul percorso di rete e poiché queste sessioni verranno istituite e rilasciate dinamicamente, il throughput disponibile può fluttuare nel tempo. Queste osservazioni ci guidano verso un altro naturale servizio che un protocollo a li vello di trasporto può fornire, cioè un throughput disponibile garantito. Con tale servizio l' applicazione può richiedere un throughput garantito di r bps e il protocollo di trasporto assicurerà che il throughput disponibile sarà sempre almeno r bps (App sensibili alla banda: telefonia Internet. App elastiche: posta elettronica, trasferimento file, Web).
Temporizzazione: Un protocollo a livello di trasporto può anche fornire garanzie di temporizzazione (timing) che, come quel.l5! per il throughput, possono assumere varie forme. Per esempio, la garanzia potrebbe essere che ogni bit che il mittente invia sulla socket venga ricevuto dalla socket di destinazione non più di 100 millisecondi più tardi. (Utile per app real time come telefonia Internet, teleconferenza, giochi multiutente. App non in tempo reale non hanno grossi vincoli).
Sicurezza: un protocollo a livello di trasporto può fornire a un' applicazione uno o più servizi di sicurezza. Per esempio, nell'host mittente, un protocollo di trasporto può cifrare tutti i dati trasmessi dal processo mittente e, nell'host di destinazione, il protocollo di trasporto può decifrare i dati prima di consegnarli al processo ricevente. Questo tipo di servizio fornirebbe riservatezza tra i due processi, anche se i dati vengono in qualche modo osservati tra il processo mittente e ricevente. Un protocollo a livello di trasporto può fornire altri servizi di sicurezza oltre alla riservatezza, compresi l'integrità dei dati e l'autenticazione end-to-end.

SERVIZI DI TRASPORTO OFFERTI DA INTERNET
Fin qui abbiamo considerato i servizi di trasporto che una rete di calcolatori potrebbe fornire in generale. Entriamo ora nello specifico ed esaminiamo il tipo di supporto alle applicazioni fornito da Internet. Internet (come ogni rete TCP/IP) mette a disposizione delle applicazioni due protocolli di trasporto: UDP e TCP. Quando gli sviluppatori di software realizzano una nuova applicazione di rete per Internet, una delle loro prime decisioni riguarda la scelta tra TCP e UDP: due protocolli che offrono modelli di servizio diversi.
Entrambi non forniscono servizio di temporizzazione e throughput.
Servizi di TCP
Servizio orientato alla connessione: TCP fa in modo che client e server si scambino informazioni di controllo a livello di trasporto prima che i messaggi a livello di applicazione comincino a fluire. Questa procedura, detta di handshaking, mette in allerta client e server, preparandoli alla partenza dei pacchetti. Dopo la fase di

handshaking, si dice che esiste una connessione TCP tra le socket dei due pro cessi. Tale connessione è full-duplex, nel senso che i due processi possono scambiarsi contemporaneamente messaggi sulla connessione. Quando l'applicazione termina l'invio deve chiudere la connessione.
Trasferimento dati affidabile: I processi comunicanti possono contare su TCP per trasportare i dati senza errori e nel giusto ordine quindi tutto ciò che viene inviato da un lato viene ricevuto dall'altro.
Controllo di flusso: il mittente deve assicurarsi che il ricevente non venga inondato di pacchetti, poiché potrebbe esaurire il buffer che contiene i pacchetti che sono in attesa di essere trasferiti al livello di applicazione (altrimenti andrebbero persi).
Controllo di congestione: un servizio che riguarda il benessere generale di Internet e non quello diretto dei processi comunicanti. Una finestra di congestione definisce ogni quanto deve essere inviato un pacchetto, poiché nel caso di rete congestionata significa che i buffer dei router (non dei dispositivi) sono pieni e i pacchetti verrebbero scartati, viene adeguata quindi la velocità di trasmissione.
Sicurezza: può essere arricchito a livello applicativo con SSL per fornire servizi di sicurezza

Servizi di UDP

UDP è un protocollo di trasporto leggero e senza fronzoli, dotato di un modello di servizi minimalista. UDP è senza connessione, non necessita quindi di handshaking, e fornisce un servizio di trasferimento dati non affidabile. Così, quando un processo invia un messaggio tramite una socket UDP, il protocollo non garantisce che questo raggiunga il processo di destinazione. Inoltre i messaggi potrebbero giungere a destinazione non in ordine. UDP non include un meccanismo di controllo della congestione.
(posta elettronica, accesso a terminale remoto, Web e trasferimento file usano tutti TCP. Questo soprattutto perché TCP fornisce un servizio di trasferimento dati affidabile, garantendo che, prima o poi, tutti i dati giungano a destinazione. Poiché le applicazioni di telefonia Internet come Skype possono tollerare perdite, ma hanno requisiti minimi di velocità di trasmissione per essere efficienti, gli sviluppatori scelgono UDP per evitare il controllo di congestione di TCP e i conseguenti messaggi aggiuntivi. Però, siccome molti firewall sono configurati per bloccare la maggior parte del traffico UDP, molte applicazioni di telefonia su Internet sono pro gettate per usare TCP come opzione di riserva nei casi in cui UDP non sia utilizzabile).

PROTOCOLLI A LIVELLO DI APPLICAZIONE

Abbiamo visto come i processi di rete comunichino tra loro inviando messaggi tra socket. Un protocollo a livello di applicazione definisce come i processi di un' applicazione, in esecuzione su sistemi periferici diversi, si scambiano i messaggi. In particolare, un protocollo a livello di applicazione definisce: tipi di messaggi scambiati (richiesta o risposta), la sintassi dei vari tipi di messaggio (quali sono i campi del messaggio), il significato delle info nei campi, le regole per determinare quando e come un processo invia e risponde ai messaggi. Il protocollo a livello di applicazione del Web è HTTP, altri, come quello di Skype sono privati.
(È importante distinguere tra applicazioni di rete e protocolli a livello di applicazione. Un protocollo a livello di applicazione è solo una parte (benché molto importane) di un applicazione di rete. Consideriamo un paio di esempi. Il Web è un'applicazione client-server che consente agli utenti di ottenere su richiesta documenti dai web server. L'applicazione web consiste di molte componenti, tra cui uno standard per i formati di documento (HTML), browser (per esempio: Firefox e Microsoft Internet Explorer), web server (per esempio: Apache, Microsoft IIS) e un protocollo a livello di applicazione. HTTP, il protocollo a livello di applicazione del Web, definisce il formato e la sequenza dei messaggi scambiati tra browser e web server. È, pertanto, solo una

parte dell'applicazione web, benché importante).

Le applicazioni di rete che prendiamo in considerazione cinque: il Web, il trasferimento di file, la posta elettronica, il servizio di directory e le applicazioni P2P. Per prima cosa tratteremo il Web, non solo in quanto rappresenta un' applicazione estremamente diffusa, ma anche perché il suo protocollo a livello di applicazione, HTTP, è diretto e facile da comprendere. Esamineremo quindi FTP che fornisce un'interessante contrapposizione con HTTP, per passare poi alla posta elettronica, la prima fondamentale applicazione Internet, più complessa rispetto al Web in quanto utilizza non uno, ma svariati protocolli a livello di applicazione. Di seguito, tratteremo il DNS che fornisce a Internet un servizio di directory. La maggior parte degli utenti non interagisce direttamente con il DNS, ma lo invoca indirettamente attraverso le proprie applicazioni (tra cui il Web, il trasferimento di file e la posta elettronica). DNS mostra, in modo elegante, come in Internet si possa implementare una funzionalità chiave della rete, come la traduzione da nome di rete a indirizzo di rete a livello applicativo. Vedremo infine varie applicazioni P2P, concentrandoci sulla distribuzione di file e la ricerca di informazioni.

WEB E HTTP

Http è il protocollo a livello di applicazione del web. Questo protocollo è implementato in due programmi, client e server, in esecuzione su due sistemi periferici diversi che comunicano tra loro scambiandosi messaggi HTTP. Il protocollo definisce sia la struttura dei messaggi sia la modalità con cui client e server si scambiano i messaggi. Prima di affrontare in dettaglio HITP soffermiamoci brevemente sulla terminologia web. Una pagina web (web page), detta anche documento, è costituita da oggetti. Un oggetto è semplicemente un file (quale un file HTML, un'immagine JPEG, un'applet Java, una clip video e così via) indirizzabile tramite un URL. La maggioranza delle pagine web consiste di un file HTML principale e diversi oggetti referenziati da esso. Per esempio, se una pagina web contiene testo in HTML e cinque immagini JPEG, allora la pagina nel complesso presenta sei oggetti: il file HTML più le cinque immagini. Il file HTML referenzia gli altri oggetti nella pagina tramite il loro URL. Ogni URL ha due componenti: il nome dell'host del server che ospita l'oggetto e il percorso dell'oggetto. Per esempio, l'URL http : / / www. someSchool.edu/someDepartment/picture.gif ha www.someSchool.edu come nome dell'host e /someDepartment/picture.gif come percorso. Un browser web (come Internet Explorer o Firefox) implementa il lato client di HTTP (quando parliamo di Web useremo le parole browser e client in modo intercambiabile). Un web server, che implementa il lato server di HITP, ospita oggetti web, indirizzabili tramite URL. Tra i più popolari ricordiamo Apache e Microsoft Internet Information Server. Quando l'utente richiede una pagina web (per esempio, cliccando su un collegamento ipertestuale), il browser invia al server messaggi di richiesta HTTP per gli oggetti nella pagina. Il server riceve le richieste e risponde con messaggi di risposta HTTP contenenti gli oggetti. HTTP utilizza TCP (anziché UDP) come protocollo di trasporto. Il client HITP per prima cosa inizia una connessione TCP con il server. Una volta stabilita, i processi client e server accedono a TCP attraverso le proprie socket (l'interfaccia socket è la porta tra un processo e la sua connessione TCP). Il client invia richieste e riceve risposte HTTP tramite la propria interfaccia socket, analogamente il server riceve richieste e invia messaggi di risposta attraverso la propria interfaccia socket. Quando il client ha mandato un messaggio alla sua interfaccia socket, questo non è più in suo possesso, ma si trova "nelle mani" di TCP. Ricordiamo che TCP mette a disposizione di HTTP un servizio di trasferimento dati affidabile; ciò implica che ogni messaggio di richiesta HTTP emesso da un processo client arriverà intatto al server e viceversa. Questo è uno dei grandi vantaggi di un'architettura organizzata a livelli: HITP non si deve preoccupare dei dati smarriti o

di come TCP recuperi le perdite.
É importante notare che il server invia i dati richiesti al client senza memorizzare alcuna informazione di stato a proposito del client. Per cui, in caso di ulteriore richiesta dello stesso oggetto da parte dello stesso client, anche nel giro di pochi secondi, il server procederà nuovamente all'invio, non avendo mantenuto alcuna traccia di quello precedentemente effettuato. Dato che i server HTTP non mantengono informazioni sui client, HTTP è classificato come protocollo senza memoria di stato (stateless protocol). Un web server è sempre attivo, ha un indirizzo IP fisso e risponde potenzialmente alle richieste provenienti da milioni di diversi browser.
In molte applicazioni per Internet, client e server comunicano per un lungo periodo di tempo, a seconda dell'App e del suo impiego la serie di richieste al server da parte del client potrebbero essere effettuate in sequenza, periodicamente a intervalli regolari o in modo intermittente. Gli sviluppatori devono quindi decidere se le varie richieste devono essere inviate su una stessa connessione TCP(**connessione persistente**) o su tutte connessioni TCP separate (**connessione non persistente**).
Connessione non persistente
Supponiamo che la pagina consista di un file HTML principale e di 10 immagini JPEG, e che tutti gli undici oggetti risiedano sullo stesso server. Ipotizziamo che l ' URL del file HTML principale sia: http://www.someSchool.edu/someDepartment/home.index.
Ecco che cosa avviene.
1. Il processo client HTTP inizializza una connessione TCP con il server www.someSchool.edu sulla porta 80, che è la porta di default per HTTP. Associate alla connessione TCP ci saranno una socket per il client e una per il server.
2. Il client HTTP, tramite la propria socket, invia al server un messaggio di richiesta HTTP che include il percorso /someDepartment/home.index.
3. Il processo server HTTP riceve il messaggio di richiesta attraverso la propria socket associata alla connessione, recupera l'oggetto / someDepartmentIhome.index dalla memoria (centrale o d i massa), lo incapsula in un messaggio di risposta HTTP che viene inviato al client attraverso la socket.
4. Il processo server HTTP comunica a TCP di chiudere la connessione. Questo, però, non termina la connessione finché non sia certo che il client abbia ricevuto integro il messaggio di risposta.
5. Il client HTTP riceve il messaggio di risposta. La connessione TCP termina. Il messaggio indica che l'oggetto incapsulato è un file HTML. Il client estrae il file dal messaggio di risposta, esamina il file HTML e trova i riferimenti ai 1 O oggetti JPEG.
6. Vengono quindi ripetuti i primi quattro passi per ciascuno degli oggetti JPEG referenziati.
Ogni connessione TCP viene chiusa dopo l'invio dell'oggetto da parte del server: vale a dire che ciascuna trasporta soltanto un messaggio di richiesta e un messaggio di risposta. Pertanto, in questo esempio, quando l'utente richiede una pagina web, vengono generate 11 connessioni TCP.
Definiamo il round-trip time (RTT), che rappresenta il tempo impiegato da un piccolo pacchetto per viaggiar dal client al server e poi tornare al client. Il RTT include i ritardi di propagazione, di accodamento nei router e di elaborazione del pacchetto.
Consideriamo ora che cosa succede quando un utente fa click con il mouse su un collegamento ipertestuale. Questa azione fa sì che il browser inizializzi una connessione TCP con il web server. Ciò comporta un handshake (letteralmente, "stretta di mano") a tre vie (three-way handshake): il client invia un piccolo segmento TCP al server, quest'ultimo manda una conferma per mezzo di un piccolo segmento TCP. Infine, il client dà anch'esso una con ferma di ritorno al server. Le prime due parti dell'handshake a tre vie richiedono un RTT. Dopo il loro completamento, il client invia un messaggio di richiesta HTTP com binato con la terza parte dell'handshake (la conferma di avvenuta ricezione, o acknowledgement)

tramite la connessione TCP. Quando il messaggio di richiesta arriva al server, quest'ultimo inoltra il file HTML sulla connessione TCP. La richiesta-risposta richiede un altro RTT. Pertanto, il tempo di risposta totale è di due RTT, più il tempo di trasmissione del file HTML da parte del server.

Connessione persistente

Le connessioni non persistenti presentano alcuni limiti: il primo è che per ogni oggetto richiesto occorre stabilire e mantenere una nuova connessione. Per ciascuna di queste connessioni si devono allocare buffer e mantenere variabili TCP sia nel client sia nel server. Ciò pone un grave onere sul web server, che può dover servire contemporaneamente richieste provenienti da centinaia di diversi client. In secondo luogo, come abbiamo appena descritto, ciascun oggetto subisce un ritardo di consegna di due RTT, uno per stabilire la connessione TCP e uno per richiedere e ricevere un oggetto. Nelle connessioni persistenti il server lascia la connessione TCP aperta dopo l'invio

possono essere trasmesse sulla stessa connessione. In particolare, non solo il server può inviare un'intera pagina web (nell'esempio, il file HTML principale e le 10 immagini) su una sola connessione TCP permanente, ma può anche spedire allo stesso client più pagine web. Queste richieste di oggetti possono essere effettuate una di seguito all' altra senza aspettare le risposte delle richieste pendenti (pipelining). In generale, il server HTIP chiude la connessione quando essa rimane inattiva per un dato lasso di tempo (un intervallo configurabile). Quando il server riceve richieste in sequenza invia gli og getti con la stessa modalità. La modalità di default di HTTP impiega connessioni persistenti con pipelining.

Cookies

Abbiamo precedentemente visto che i server HTTP sono privi di stato. Ciò semplifica la progettazione e consente di sviluppare web server ad alte prestazioni, in grado di gestire migliaia di connessioni TCP simultanee. Tuttavia, è spesso auspicabile che i web server possano autenticare gli utenti, sia per limitare l'accesso da parte di questi ultimi sia per fornire contenuti in funzione della loro identità. A questo scopo, HTTP adotta i cookie. I cookie consentono ai server di tener traccia degli utenti. La maggior parte dei siti commerciali usa i cookie. La tecnologia dei cookies presenta 4 componenti: (1) una riga di intestazione nel messaggio di risposta HTTP, (2) una riga di intestazione nel messaggio di richiesta HTTP, (3) un file mantenuto sul sistema dell'utente e gestito dal browser e (4) un database sul sito.

Supponiamo che per la prima volta Susan contatti il sito di Amazon. Quando giunge la richiesta al web server di Amazon, il sito crea un identificativo unico e una voce nel proprio database, indicizzata dal numero identificativo. A questo punto il server risponde al browser di Susan, includendo nella risposta HTTP l'intestazione "Set-cookie" che contiene il numero identificativo. Per esempio, la riga di intestazione potrebbe essere: Set - cookie : 1678 .

Quando il browser di Susan riceve il messaggio di risposta HTTP, vede l'intestazione Set - cookie. Il browser allora aggiunge una riga al file dei cookie che gestisce. Questa riga include il nome dell'host del server e il numero identificativo nell'intestazione Set - cookie : . Si noti che il file di cookie contiene già una voce per eBay, dato che Susan ha già visitato quel sito in passato. Mentre Susan continua a navigare nel sito di Amazon, ogni volta che richiede una pagina web, il suo browser consulta il suo file dei cookie, estrae il suo numero identificativo per il sito e pone nella richiesta HTIP una riga di intestazione del cookie che include tale numero. Più nello specifico, ciascuna delle sue richieste HTIP al server di Amazon include la riga di intestazione: "Cookie:1678" . In tal modo è possibile monitorare l'attività di Susan nel sito. Sebbene esso non ne conosca necessariamente il nome, sa esattamente quali pagine sono state visitate dall'utente 1678, in quale ordine e a quali orari! Se Susan torna nel sito, magari una settimana più tardi, il suo browser continuerà a

inserire la riga di intestazione Cookie : 1678 nei messaggi di richiesta. Amazon può suggerire a Susan dei prodotti sulla base delle pagine web che ha visitato in passato. I cookie possono anche essere usati per creare un livello di sessione utente al di sopra di HTTP che è privo di stato. Per esempio, quando un utente si identifica in un'applicazione di posta elettronica basata su Web, come Hotmail, il browser invia le informazioni del cookie al server, permettendo a quest'ultimo di identificare l'utente attraverso la sessione utente dell'applicazione.

Web Caching

Una web cache, nota anche come proxy server, è un'entità di rete che soddisfa richieste HTTP al posto del web server effettivo. Il proxy ha una propria memoria su disco (una cache) in cui conserva copie di oggetti recentemente richiesti. Il browser di un utente può essere configurato in modo che tutte le richieste HTTP dell'utente vengano innanzitutto dirette al proxy server.

Supponiamo per esempio che un browser stia richiedendo l'oggetto http://www.someschool.edu/campus.gif. Ecco che cosa succede.

1. Il browser stabilisce una connessione TCP con il proxy server e invia una richiesta HTTP per l'oggetto specificato.
2. Il proxy controlla la presenza di una copia dell'oggetto memorizzata localmente. Se l'oggetto viene rilevato, il proxy lo inoltra all'interno di un messaggio di risposta HTTP al browser.
3. Se, invece, la cache non dispone dell'oggetto, apre una connessione TCP verso il server di origine; ossia, nel nostro esempio, www.someschool.edu. Poi, il proxy invia al server una richiesta HTTP per l oggetto. Una volta ricevuta tale richiesta, il server di origine invia al proxy l'oggetto all'interno di una risposta HTTP.
4. Quando il proxy riceve l'oggetto ne salva una copia nella propria memoria locale e ne inoltra un'altra copia, all'interno di un messaggio di risposta HTTP, al browser (sulla connessione TCP esistente tra il browser e il proxy).

Si noti che il proxy è contemporaneamente server e client: quando riceve richieste da un browser e gli invia risposte agisce da server, quando invia richieste e riceve risposte da un server di origine funziona da client. I proxy possono ridurre in maniera sostanziale i tempi di attesa di risposta e il traffico sul Web

GET Condizionale

Sebbene il web caching riduca i tempi di risposta percepiti dall'utente, introduce un nuovo problema: la copia di un oggetto che risiede in cache potrebbe essere scaduta. In altre parole, l'oggetto ospitato nel web server potrebbe esser stato modificato rispetto alla copia nel client (sia esso un proxy o un browser). Fortunatamente, HTTP presenta un meccanismo che permette alla cache di verificare se i suoi oggetti sono aggiornati. Questo meccanismo è chiamato GET condizionale (conditional GET). Un messaggio di richiesta HTTP viene detto messaggio di GET condizionale se (1) usa il metodo GET e (2) include una riga di intestazione "If-modified-since" .

Un client manda un messaggio di richiesta HTTP al proxy per un dato oggetto, il proxy non lo ha in memoria quindi lo richiede al server, inviandolo poi al client e salvandolo in memoria insieme alla data di ultima modifica. Quando dopo un pò di tempo l'oggetto viene richiesto al proxy e lui si accorge di averlo in memoria, allora effettua un controllo di aggiornamento mandando un GET condizionale, verificando che la sua versione sia la più aggiornata: in caso negativo riceverà una risposta dal server contente l'oggetto aggiornato, altrimenti un "Not Modified" sarà notificato al proxy.

TRASFERIMENTO DI FILE: FTP

In una tipica sessione FTP, l'utente utilizza un host locale per trasferire file da o verso un host remoto.Per accedere ed essere autorizzato a scambiare info con il file

system remoto, l'utente deve fornire un nome identificativo e una password, dopo di che può trasferire file tra i due file system. L'utente interagisce con FTP tramite un agente software (FTP user agent). Innanzitutto l'utente fornisce il nome dell'host remoto, in modo che il processo FTP client nell'host locale stabilisca una connessione TCP con il processo FTP server nell'host remoto e fornisce nome identificativo e password, inviate sulla connessione TCP. Una volta ottenuta l'autorizzazione del server può quindi inviare file memorizzati nel file System locale verso quello remoto (o viceversa).
HTTP e FTP sono protocolli di trasferimento file e presentano molte caratteristiche comuni e differenze:utilizzano entrambi TCP, ma la principale differenza è che FTP utilizza due connessioni TCP parallele per trasferire file, una per inviare info di controllo tra gli host(identificativo dell'utente, password, comandi per cambiare directory remota e comandi per inviare e ricevere file) e una per il vero e proprio invio del file.
Quando un utente inizia una sessione PfP con un host remoto, il lato client di FfP (l' utente) come prima cosa inizializza una connessione di controllo TCP con il lato server (l'host remoto) sulla porta 21 del server, dove invia poi l'identificativo dell'u- tente e la password. Inoltre, il lato client di FfP spedisce sulla connessione di con trollo i comandi per cambiare la directory remota. Quando il lato server riceve sulla connessione di controllo un comando di trasferimento file (da o verso l'host remoto), inizializza una connessione dati TCP verso il lato client. FfP invia esattamente un file sulla connessione dati e quindi la chiude. Se, nel corso della stessa sessione, l'utente volesse trasferire un altro file, FfP aprirebbe un'altra connessione dati. Pertanto la connessione di controllo rimane aperta per l'intera durata della sessione utente, ma si crea una nuova connessione dati per ogni file trasferito all'interno della sessione (in altre parole, le connessioni dati sono "non persistenti").

POSTA ELETTRONICA IN INTERNET
In questo paragrafo esamineremo i protocolli a livello applicativo alla base della posta elettronica su Internet.
I mail server costituiscono la parte centrale dell'infrastruttura del servizio di posta elettronica. Ciascun destinatario, come per esempio Bob, ha una casella di posta (mai/box) collocata in un mail server. La mailbox di Bob gestisce e contiene messaggi a lui inviati. Un tipico messaggio inizia il proprio viaggio dallo user agent, giunge al mail server del mittente e prosegue fino al mail server del destinatario utilizzando il protocollo SMTP, dove viene depositato nella sua casella. Per accedere ai messaggi della propria casella Bob deve essere autenticato dal server che lo ospita tramite nome utente e password. SMTP rappresenta il principale protocollo a livello di applicazione per la posta elettronica su Internet. Fa uso del servizio di trasferimento dati affidabile proprio di TCP per trasferire la mail dal server del mittente a quello del destinatario. Così come accade per la maggior parte dei protocolli a livello di applicazione, SMTP presenta un lato client, in esecuzione sul mail server del mittente e un lato server, in esecuzione sul server del destinatario.Entrambi i lati possono essere eseguiti su tutti i server di posta.
Al fine di illustrare le operazioni di base di SMTP, presentiamo uno scenario tipi co. Supponiamo che Alice voglia inviare a Bob un semplice messaggio ASCII.
1. Alice invoca il proprio user agent per la posta elettronica, fornisce l'indirizzo di posta di Bob (per esempio bob@someschool.edu), compone il messaggio e dà istruzione allo user agent di inviarlo.
2. Lo user agent di Alice invia il messaggio al suo mail server, dove è collocato in una coda di messaggi.
3. Il lato client di SMTP, eseguito sul server di Alice, vede il messaggio nella coda dei messaggi e apre una connessione TCP verso un server SMTP in esecuzione sul mail

server di Bob.
4. Dopo un handshaking SMTP, il client SMTP invia il messaggio di Alice sulla connessione TCP
5. Presso il mail server di Bob, il lato server di SMTP riceve il messaggio, che viene posizionato nella casella di Bob.
6. Bob, quando lo ritiene opportuno, invoca il proprio user agent per leggere il messaggio
SMTP può contare sul servizio dati affidabile proprio del TCP. Inoltre nel caso avesse altri messaggi da inviare al server, il client ripeterebbe gli stessi passaggi sulla stessa connessione TCP (SMTP fa uso di connessioni persistenti); viceversa, ordinerebbe a TCP di chiudere la connessione. Se il mail server di Bob è spento, il messaggio rimane nel mail server di Alice e attende un nuovo tentativo. Il messaggio non viene inviato a nessun mail server intermedio.
Tuttavia nel nostro puzzle manca ancora un pezzo. Come fa un destinatario (quale Bob), che esegue uno user agent sul proprio PC locale, a ottenere i messaggi che si trovano nel mail server del suo provider? Osserviamo che lo user agent di Bob non può usare SMTP per ottenere tali messaggi dato che si tratta di un' operazione di pull, mentre SMTP è u protocollo push. Il puzzle viene completato introducendo uno speciale protocollo di accesso alla posta(POP3, IMAP, HTTP), che trasferisce i messaggi dal mail server di Bob al suo PC locale (SMTP è usato per trasferire posta dal server del mittente a quello del destinatario ed è anche utilizzato per trasferire la posta dallo user agent al mail server del mittente. Per trasferire messaggi dal mail server allo user agent del destinatario viene impiegato un protocollo di accesso alla posta, quale POP3).

POP3

POP3 entra in azione quando lo user agent (il client) apre una connessione TCP verso il mail server (il server) sulla porta 1 10. Quando la connessione TCP è stabilita, POP3 procede in tre fasi: autorizzazione, transazione e aggiornamento. Durante la prima fase (autorizzazione) lo user agent invia nome utente e password (in chiaro) per autenticare l'utente. Durante la seconda fase (transazione) lo user agent recupera i messaggi; inoltre, durante questa fase, può marcare i messaggi per la cancellazione, rimuovere i marcatori di cancellazione. La fase di aggiornamento ha luogo dopo che il client ha inviato il comando quit, che conclude la sessione POP3; in questo istante, il server di posta rimuove i messaggi che sono stati marcati per la cancellazione. Durante una sessione tra uno user agent e il mail server, il server POP3 mantiene alcune informazione di stato; in particolare, tiene traccia dei messaggi dell'utente marcati come cancellati. Tuttavia, il server POP3 non trasporta informazione di stato tra sessioni POP3. Questa mancanza di informazione di stato persistente tra sessioni semplifica di molto l'implementazione.

IMAMP

Anche IMAP è un protocollo di accesso alla posta, ma presenta maggiori potenzialità rispetto a POP3 ed è quindi assai più complesso. Di conseguenza risultano molto più complesse anche le implementazioni del lato client e server. Un server IMAP associa a una cartella ogni messaggio arrivato al server. I messaggi in arrivo sono associati alla cartella INBOX del destinatario. Quest'ultimo può poi spostare il messaggio in una nuova cartella creata dall'utente, leggerlo, cancellarlo e così via. Il protocollo IMAP fornisce comandi per consentire agli utenti di creare cartelle e spostare i messaggi da una cartella a un' altra. Fornisce anche comandi che consentono agli utenti di effettuare ricerche nelle cartelle remote sulla base di criteri specifici.
A differenza di POP3, i server IMAMP mantengono le info di stato sull'utente da una sessione ali' altra: per esempio, i nomi delle cartelle e l'associazione tra i messaggi e le cartelle.

Posta basata sul web

Oggi si può ricevere e inviare mail tramite un browser web. Grazie a tale servizio, lo user agent è un semplice browser web e l'utente comunica con la propria casella remota via HTTP. Quando un destinatario, quale Bob, vuole accedere alla propria casella, il messaggio e-mail viene spedito dal server di posta al browser di Bob usando il pro tocollo HTTP anziché POP3 o IMAP. Quando un mittente, quale Alice, vuole inviare un messaggio di posta elettronica, quest'ultimo viene spedito dal suo browser al suo server di posta su HTTP anziché su SMTP. Il server di posta di Alice, in ogni caso, manda ancora i messaggi e li riceve utilizzando SMTP.

Confronto tra HTTP e SMTP

Confrontiamo brevemente SMTP e HTTP. I due protocolli vengono utilizzati per trasferire file da un host a un altro. HTTP trasferisce file (spesso chiamati oggetti) da un web server a un web client (solitamente un browser). SMTP trasferisce file (ossia messaggi di posta elettronica) da un mail server a un altro. Durante il trasferimento sia HTTP persistente sia SMTP utilizzano connessioni persistenti e quindi presentano caratteristiche comuni. Esistono però sostanziali differenze. Innanzitutto HTTP è principalmente un protocollo poli: qualcuno carica informazioni su un web server e gli utenti usano HTTP per attirarle a se (pull) dal server. In particolare, la connessione TCP viene iniziata dalla macchina che vuole ricevere il file. Al contrario, SMTP è sostanzialmente un protocollo push: il mail server di invio spinge (push) i file al mail server in ricezione. In particolare, la connessione TCP viene iniziata dall'host che vuole spedire il file.

Una seconda differenza, cui abbiamo già accennato, è che SMTP deve comporre l'intero messaggio (compreso il corpo) in ASCII a 7 bit. Anche se il messaggio con tiene caratteri che non appartengono ad ASCII a 7 bit (per esempio, caratteri con accenti o dati binari come un'immagine), il messaggio deve essere comunque codificato in ASCII a 7 bit. HTTP non impone tale vincolo. Un' altra importante differenza riguarda la gestione di un documento che contiene testo e immagini (insieme ad altri possibili tipi di media). HTTP incapsula ogni oggetto nel proprio messaggio di risposta HTTP. La posta elettronica colloca tutti gli oggetti in un unico messaggio.

DNS: SERVIZIO DI DIRECTORY DI INTERNET

Gli host possono essere identificati in due modi: con i nomi (www.eurecom.fr) o tramite gli indirizzi IP (indirizzo a 4 byte, Un indi rizzo IP è gerarchico perché, leggendolo da sinistra a destra, otteniamo informazioni sempre più specifiche sulla collocazione dell'host in Internet). Al fine di conciliare i due approcci è necessario un servizio che effettui la traduzione dei nomi degli host nei loro indirizzi IP: DNS. DNS è (1) un database distribuito implementato in una gerarchia di DNS server e (2) un protocollo a livello di applicazione che consente agli host di interrogare il database. Il protocollo DNS utilizza UDP e la porta 53.

DNS viene comunemente utilizzato da altri protocolli a livello di applicazione, tra cui HTTP, SMTP e FTP, per tradurre i nomi di host forniti dall'utente in indirizzi IP. Per esempio, consideriamo che cosa succede quando un browser (ossia un client HTTP) in esecuzione sull'host di un utente richiede l'URL www.someschool.edu/index.html. Affinché l'host dell'utente sia in grado di inviare un messaggio di richiesta HTTP al web server www.someschool.edu, esso deve come prima cosa ottenere il suo indirizzo IP. Ciò avviene come segue:

1 . La stessa macchina utente esegue il lato client dell'applicazione DNS

2. Il browser estrae il nome dell'host, www.someschool.edu, dall'URL e lo passa al lato client dell'applicazione DNS

3. Il client DNS invia una interrogazione contenente l'hostname all'interno di data grammi UDP a un DNS server.

4. Il client DNS prima o poi riceve una risposta, che include l'indirizzo IP corrispondente all'hostname.

5. Una volta ricevuto l'indirizzo IP dal DNS, il browser può dare inizio a una connessione TCP verso il processo server HTTP collegato alla porta 80 di quell'indirizzo IP.
Oltre alla traduzione degli hostname in indirizzi IP, DNS mette a disposizione altri importanti servizi: host aliasing(un host dal nome complicato può avere uno o più sinonimi), Mail Server Aliasing, Distribuzione del carico di rete (distribuire il carico tra server replicati, per esempio dei web server. I siti con molto traffico vengono replicati su più server, e ciascuno di questi viene eseguito su un host diverso e presenta un indirizzo IP differente. Nel caso di web server replicati, va dunque associato a ogni hostname canonico un insieme di indirizzi IP. Il database DNS contiene questo insieme di indirizzi. Quando i client effettuano una query DNS per un nome associato a un insieme di indirizzi, il server risponde con l'intero insieme di indirizzi, ma ne varia l'ordinamento a ogni risposta. Dato che generalmente un client invia il suo messaggio di richiesta HTTP al primo indirizzo IP elencato nell'insieme, la rotazione DNS distribuisce il traffico sui server replicati).
Pertanto, dal punto di vista dell'applicazione nell'host utente, il DNS è una scatola nera che fornisce un servizio di traduzione semplice e diretto. Tuttavia nei fatti la scatola nera è costituita da un gran numero di DNS server distribuiti per il mondo e da un protocollo a livello di applicazione che specifica la comunicazione tra DNS server e host richiedenti. Utilizzare un unico DNS server che contenga tutte le corrispondenze è pressoché impossibile dato l'alto numero di host e l'elevato numero di richieste che seguirebbero, l'unico punto di fallimento, la distanza e la manutenzione. Di conseguenza il DNS è stato progettato in maniera distribuita: utilizza un gran numero di server, distribuiti in tutto il mondo e organizzati in una gerarchia, in cui sono distribuite tutte le corrispondenze. Esistono più classi di DNS server: root server, TLD server, server autoritativi e server locali. Consideriamo un semplice esempio. Supponiamo che l'host cis.poly.edu voglia l'indirizzo IP di gaia.es.umass.edu. Supponiamo, inoltre, che il DNS server locale per cis.poly.edu sia dns.poly.edu, mentre un server autoritativo per gaia.es.umass.edu sia dns.umass.edu. L'host cis.poly.edu dapprima invia un messaggio di richiesta DNS al proprio server locale dns.poly.edu. Il messaggio contiene il nome da tradurre, ossia gaia.es.umass.edu. Il server locale inoltra il messaggio di richiesta a un root server. Quest'ultimo prende nota del suffisso edu e restituisce al server locale un elenco di indirizzi IP per i TLD server responsabili di edu. Il server locale rinvia quindi il messaggio di richiesta a uno di questi ultimi. Il TLD server prende nota del suffisso umass.edu e risponde con l'indirizzo IP del server autoritativo per l'Università del Massachusetts, ossia dns.umass.edu. Infine, il DNS server locale rimanda il messaggio di richiesta direttamente a dns.umass.edu, che risponde con l'indirizzo IP di gaia.es.umass.edu. Si noti che in questo esempio, per ottenere la mappatura di un hostname, sono stati inviati ben otto messaggi DNS: quattro messaggi di richiesta e quattro di risposta.
Ogni volta che il server locale dns.poly.edu riceve una risposta da qualche DNS server, può conservare in cache le informazioni contenute nella risposta, un DNS server locale può, inoltre, memorizzare in cache gli indirizzi IP dei TLD server, consentendogli di aggirare i root server nella catena di richieste.

APPLICAZIONI PEER-TO-PEER

Distribuzione di file P2P

In una distribuzione di file client-server, il server deve inviare una copia del file a ciascun peer, ponendo un enorme fardello sul server e consumandone un'elevata quantità di banda. In una distribuzione di file con P2P ciascun peer può redistribuire agli altri qualsiasi porzione del file abbia ricevuto, aiutando in questo modo il server nel processo di distribuzione. Uno tra i più diffusi protocolli di distribuzione di file

tramite P2P è BitTorrent.
*Confronto P2P- ClientServer:*sia us la banda di upload del collegamento di accesso del server, ui la banda di upload del collegamento di accesso dell'i-esimo peer e di la banda di download del collegamento di accesso dell'i-esimo peer. Siano, inoltre, F la dimensione del file da distribuire (in bit) e N il numero di peer che vuole una copia del file.
Dcs = max {NF/us; F/dmin }

- Il server deve trasmettere una copia del file a N peer, quindi NF bit. Dato che la banda di upload del server è pari a us, il tempo di distribuzione sarà almeno NF/us secondi.
- Sia dmin la minima banda di download di un peer, quest'ultimo non potrà ricevere tutti i bit del file prima di F/dmin secondi.

Dpp = max { NF/us; F/dmin; NF/us+u1+..+un }

- All'inizio della distribuzione solo il server dispone del file. Per trasmetterlo al l'interno della comunità dei peer il server deve inviare ciascun bit del file almeno una volta nel collegamento di accesso. Quindi il minimo tempo di distribuzione è F/u . Diversamente dallo schema client-server, un bit inviato una volta dal server puÒnon dover essere inviato di nuovo, in quanto i peer possono re-distribuire i bit tra loro.
- Come per l'architettura client-server, il peer con la velocità di download più bassa non può ottenere tutti i bit del file in meno di F/dmin secondi.
- Infine, si osservi che la capacità totale di upload del sistema nel suo complesso è uguale alla velocità di upload del server più quella di ciascun peer, cioè utot = u1+..+un, il sistema deve consegnare F bit a ciascuno degli N peer, e questo non potrà essere fatto ad una velocità maggiore di NF/(us+u1+..+un).

BitTorrent
BitTorrent è un diffuso protocollo P2P per la distribuzione di file. Nel gergo di BitTorrent, l'insieme di tutti i peer che partecipano alla distribuzione di un particolare file è chiamato torrent (torrente). I peer in un torrent scaricano chunk (parti) del file di uguale dimensione uno dall altro, con una dimensione tipica di 256 kbyte. Quando un peer entra a far parte di un torrent per la prima volta, non ha chunk del file. Col passare del tempo accumula sempre più pezzi che, mentre scarica, invia agli altri peer. Una volta che un peer ha acquisito l'intero file, può (egoisticamente) lasciare il torrent o (altruisticamente) rimanere nel torrent e continuare a inviare chunk agli altri peer. Inoltre, qualsiasi peer può lasciare il torrent in qualsiasi momento con solo un sottoinsieme dei chunk del file e rientrare a far parte del torrent in seguito.
Illustriamo ora più da vicino come funziona BitTorrent. Ciascun torrent ha un nodo di infrastruttura chiamato tracker. Quando un peer entra a far parte di un torrent, si registra presso il tracker e periodicamente lo informa che è ancora nel torrent. Quando un nuovo peer, Alice, entra a far parte di un torrent, il tracker seleziona in modo casuale un sottoinsieme di peer (diciamo 50) dall'insieme dei peer che stanno partecipando a quel torrent, e invia l'indirizzo IP di questi 50 peer ad Alice. Avendo la lista dei peer, Alice cerca di stabilire delle connessioni TCP contemporanee con tutti i peer della lista. Chiamiamo i peer con i quali Alice riesce a stabilire una connessione TCP "peer vicini" (neighboring peer). Col passare del tempo, alcuni di questi peer possono lasciare il torrent, mentre altri (al di fuori dei 50 iniziali) possono cercare di stabilire una connessione TCP con Alice: quindi i peer vicini a un dato peer cambiano nel tempo. In un certo istante, ciascun peer avrà un sottoinsieme dei chunk di un file e peer diversi ne avranno differenti sottoinsiemi. Periodicamente Alice chiederà a ciascuno dei suoi vicini, tramite le connessioni TCP, la lista dei chunk del file in loro possesso. Tramite questa conoscenza, Alice invierà richieste, di nuovo sulle connessioni TCP, per i chunk del file che ancora le mancano. Di conseguenza, in un dato istante Alice avrà un sottoinsieme dei chunk del file e saprà quali chunk hanno i

suoi vicini. Con queste informazioni, Alice deve prendere due importanti decisioni: in primo luogo quali chunk deve richiedere per primi ai suoi vicini e, secondariamente, a quali vicini dovrebbe inviare i chunk a lei richiesti. Nel decidere quali chunk richiedere, Alice adotta la tecnica del rarest first (il più raro per primo). L'idea è determinare tra i chunk che ancora le mancano, quelli che sono più rari; cioè i chunk con il minor numero di copie ripetute tra i suoi vicini, e richiederli per primi. In questo modo, i chunk più rari vengono ridistribuiti più velocemente, cercando, approssimativamente, di rendere uguale il numero di copie di ciascun chunk nel torrent.
Per determinare a quali richieste Alice debba rispondere, BitTorrent usa un intel- ligente algoritmo di trading (scambio). L'idea di base è che Alice attribuisca priorità ai vicini che le stanno inviando dati in questo momento alla velocità più alta. Speci ficatamente, per ciascuno dei suoi vicini, Alice misura continuamente la velocità alla quale riceve i bit e determina i quattro peer che le stanno passando i bit alla velocità più elevata. Alice poi contraccambia inviando chunk del file a quegli stessi quattro peer. Ogni 10 secondi ricalcola la velocità e può darsi che modifichi l'insieme dei quattro peer. Aspetto importante è che ogni 30 secondi sceglie casualmente un vicino in più e gli invia dei chunk. Chiamiamo Bob il peer scelto casualmente. Dato che Alice sta inviando dati a Bob, potrebbe diventare uno dei quattro peer unchoked di Bob, nel qual caso Bob inizierebbe a inviare dati ad Alice. Se la velocità alla quale Bob manda i dati ad Alice è abbastanza alta, Bob, a sua volta, potrebbe diventare uno dei quattro peer unchoked di Alice. In altre parole, ogni 30 secondi Alice sceglie casualmente un nuovo compagno di scambi e inizia a scambiare chunk con quello. Se i due peer sono soddisfatti degli scambi, l'uno metterà l' altro nella propria lista dei quattro unchoked e continueranno a effettuare scambi tra loro finché uno non trovi un partner migliore. L'effetto è che i peer in grado di inviare dati a velocità compatibili tendono a trovarsi. La selezione casuale dei vicini consente anche a nuovi peer di ottenere chunk del file, in modo che abbiano qualcosa da scambiare.

Tabelle Hash Distribuite (DHT)
Analizzeremo come costruire una versione P2P e distribuita di un database che memorizzi le coppie (chiave, valore) di milioni di peer. Nel sistema P2P ogni peer gestirà solo un piccolo sottoinsieme della totalità delle coppie (chiave, valore). Ogni peer potrà interrogare il database distribuito con una specifica chiave; il database distribuito quindi localizzerà i peer che hanno le coppie (chiave, valore) corrispondenti e le restituirà al peer che ha fatto l'interrogazione. Inoltre, ogni peer potrà inserire nel database nuove coppie (chiave, valore). Tale database distribuito prende il nome di tabella hash distribuita.
Cominciamo ad assegnare un identificatore a ogni peer, dato da un intero nell'intervallo [O, 2^n - I] per un n fissato. Si noti che questo identificatore può essere rappresentato con n bit. Anche ogni chiave deve essere un intero nello stesso intervallo. Per creare numeri interi a partire dalle chiavi useremo una funzione hash che crei la corrispondenza tra le chiavi (per esempio i codici fiscali) e gli interi nell'intervallo [O, 2^n - l]. Una funzione hash è una funzione molti a uno, per cui due input diversi possono avere lo stesso output (stesso numero intero), ma con probabilità estremamente piccola; quindi, quando ci riferiamo alla "chiave", ci stiamo riferendo alla hash della chiave originale.
Consideriamo ora il problema di memorizzare le coppie (chiave, valore) nella DHT. Il problema centrale è definire una regola per assegnare le chiavi ai peer.

Dato che ogni peer ha un identificatore intero e che ogni chiave è un intero nello stesso intervallo, un approccio naturale è quello di assegnare ogni coppia (chiave, valore) al peer il cui identificatore è il più vicino alla chiave. Per implementare questo schema è necessario definire che cosa si intende per "più vicino". Per comodità, definiamo il peer più vicino come il successore più vicino della chiave. Supponiamo ora che un peer, Alice, voglia inserire una coppia (chiave, valore) nella DHT. Ma come fa Alice a determinare il peer più vicino alla chiave? Se Alice tenesse traccia di tutti i peer del sistema (identificatore del peer e corrispondente indirizzo IP) allora potrebbe determinare localmente il peer più vicino. Questo approccio però richiederebbe che ogni peer tenga traccia di tutti gli altri peer nella DHT, schema completamente impraticabile per sistemi su larga scala con milioni di peer.
Per affrontare il problema di scalabilità menzionato prima organizziamo i peer in un cerchio. In questo schema circolare ogni peer tiene traccia solo del suo successore immediato e del suo predecessore immediato (modulo 2"). Questo schema circolare dei peer è un caso speciale di rete di overlay. In una rete di overlay i peer formano una rete logica astratta che risiede sopra la rete sottostante che consiste di collegamenti fisici, router e host. I collegamenti in una rete di overlay non sono connessioni fisiche, ma semplici relazioni virtuali tra coppie di peer.
Supponiamo che un peer voglia sapere chi è il responsabile di una data chiave, allora chiederà al suo successore o predecessore a seconda di chi è più vicino. Se a sua volta il peer non è responsabile per la chiave si limita a inviare il messaggio al suo successore. Questa soluzione però introduce un nuovo problema. Sebbene ogni peer sia consapevole di soli due peer vicini, per trovare il nodo responsabile di una chiave (nel caso peggiore) tutti gli N nodi della DHT devono inoltrare il messaggio sul cerchio; in media vengono inviati N/2 messaggi.
Così nella progettazione di una DHT esiste un compromesso tra il numero di vicini di cui ogni peer deve tenere traccia e il numero di messaggi che la DHT deve trasmettere per risolvere una singola interrogazione. Un modo è tenere come base la copertura circolare, ma aggiungere delle "scorciatoie" in modo che ogni peer non tenga traccia solo del suo successore e predecessore immediato, ma anche di un numero relativamente piccolo di peer disseminati nel cerchio e a lui uniti dalle scorciatoie. Le scorciatoie vengono usate per velocizzare il routing dei messaggi di interrogazione. Quando un peer riceve un messaggio di interrogazione per una chiave, lo inoltra al vicino (successore o tramite scorciatoia) più vicino alla chiave. Per raggiungere un compromesso soddisfacente sia il numero di vicini per peer sia il numero di messaggi per interrogazione sarà O(log N).
Nei sistemi P2P ogni peer può entrare e uscire senza prevviso. Quindi nella proget tazione delle DHT dobbiamo preoccuparci di mantenere la copertura DHT in presen za di questo ricambio continuo dei peer.
Per gestire il ricambio dei peer richiediamo ora che ogni peer tenga traccia (cioè conosca l'indirizzo IP) del suo primo e secondo successore; per esempio, il peer 4 ora tiene traccia sia del peer 5 sia del peer 6. Richiediamo inoltre che

ogni peer verifichi periodicamente che i suoi due successori siano attivi (per esempio inviando periodicamente un messaggio di ping a entrambi e chiedendo una risposta). Vediamo ora come la DHT venga gestita quando un peer improvvisamente la abbandona. Supponiamo, per esempio, che il peer 5 abbandoni improvvisamente la DHT. In questo caso, i due peer che precedono il peer che se ne è andato (4 e 3) apprendono che il 5 ha abbandonato la DHT perché non risponde più ai loro messaggi di ping. Quindi i peer 3 e 4 devono aggiornare i loro successori. Consideriamo di seguito come il peer 4 possa aggiornare il suo stato.
1. Il peer 4 sostituisce il suo primo successore (peer 5) con il suo secondo successore (peer 6).
2. Quindi il peer 4 chiede al suo nuovo primo successore (peer 6) l'identificatore e l'indirizzo IP del suo successore immediato (peer 7). Il peer 4 rende il peer 7 il suo secondo successore.
Stessa cosa farà successivamente il peer 3.
vediamo ora che cosa succede quando un peer vuole unirsi alla DHT. Supponiamo che un peer con identificatore 1 3 voglia unirsi alla DHT e che nel momento in cui si unisce conosca solo l'esistenza del peer 1 . Il peer 13 invierebbe per prima cosa un messaggio al peer 1 chiedendo "quali sono il predecessore e il successore del 13 ?" . Tale messaggio viene inoltrato attraverso la DHT finché raggiunga il peer 12 che sa di essere il predecessore del 13 e che il suo successore attuale, il peer 14, diventerà il successore del 13 . Quindi il peer 12 invia le informazioni sul predecessore e sul successore al peer 13 . Il peer 13 ora può unirsi alla DHT prendendo come successore il peer 14 e notificando al peer 12 di cambiare il suo successore immediato in 13 .