Reti di calcolatori e internet

Appunti

Alessandro Ferro

Introduzione

Questi appunti, presi dal libro "Reti di Calcolatori e Internet - un approccio Top Down" (Kurose, Ross, 7th edition) sono parziali in quanto una parte sono presi a penna. La numerazione dei capitoli, sezioni e paragrafi segue quella del libro.

Capitolo 1

Reti di calcolatori e Internet

[Le sezioni 1.1 e 1.2 sono scritte a penna sul quaderno]

1.3 Il nucleo della rete

1.3.2 Commutazione di circuito

Per spostare i dati in una rete esistono due approcci: commutazione di pacchetto e commutazione di circuito.

Nelle reti a commutazione di circuito, le risorse necessarie sono allocate per l'intera durata della comunicazione tra i sistemi periferici. Nelle reti a commutazione di pacchetto, invece, tali risorse non sono riservate e sono allocate in funzione della necessità della rete.

Le reti telefoniche sono esempi di rete a commutazione di circuito in quanto il canale resta aperto anche in assenza di comunicazione.

Quando la rete stabilisce un circuito, viene riservata anche una certa larghezza di banda, pertanto, il mittente può trasferire i dai a una velocità costante garantita.

D'inverso, quando un host invia un pacchetto a un altro host su una rete a commutazione di pacchetto, esso viene immesso nella rete senza che vengano ad esso riservate risorse.

Multiplexing nelle reti a commutazione di circuito

In una rete a commutazione di circuito può essere implementato il multiplexing, ovvero l'utilizzo dello stesso mezzo trasmissivo da più utenti.

Il multiplexing viene implementato tramite:

- Suddivisione di frequenza: lo spettro di frequenza di un collegamento viene suddiviso tra le connessioni stabilite. Ciò, ad esempio, è il caso di walkie-talkie che comunicano su un circuito a una frequenza diversa da altri walkie-talkie che comunicano sullo stesso circuito.

- Suddivisione di tempo: quando la rete stabilisce una connessione a commutazione di circuito, assegna degli slot temporali per ogni sessione.

Confronto tra commutazione di pacchetto e commutazione di circuito

I denigratori della commutazione di pacchetto sostengono che quest'ultimo mal si adatti alle applicazioni in tempo reale a causa dei suoi ritardi variabili e non deterministici. I suoi sostenitori, invece, affermano che la commutazione di pacchetto non soltanto offra una migliore condivisione della larghezza di banda, ma è anche più semplice, efficiente ed economica.

La commutazione di pacchetto risulta essere più efficiente perché la commutazione di circuito pre-alloca l'uso del collegamento trasmissivo indipendentemente dalla richiesta necessaria, con collegamenti garantiti ma potenzialmente non necessari. D'altro canto la commutazione di pacchetto alloca le risorse su richiesta.

1.3.3 Una rete di reti

Gli ISP si distinguono per la copertura geografica: gli ISP di accesso, che forniscono Internet direttamente all'utente finale, sono connessi agli ISP regionali che a loro volta sono connessi agli ISP di primo livello (cosiddetti ISP globali). Tale configurazione permette a tutti i nodi della rete di essere in comunicazione.

Per ridurre i costi una coppia di ISP vicini e di pari livello gerarchico può fare uso di peering, cioè connettere direttamente le loro reti in modo tale che il traffico passi direttamente tra di loro anziché passare per un intermediario.

1.4 Ritardi, perdite e throughput nelle reti a commutazione di pacchetto

Idealmente, vorremmo che i servizi Internet siano in grado di spostare dati tra due sistemi periferici istantaneamente e senza nessuna perdita dati. Ciò, ovviamente, non è fattibile nella realtà: il throughput è necessariamente limitato, ci possono essere ritardi e si potrebbero perdere pacchetti.

1.4.1 Panoramica del ritardo nelle reti a commutazione di pacchetto

Un pacchetto parte da un host, passa una serie di router e conclude il viaggio in un altro host. In ciascun nodo lungo il percorso (inclusi sorgente e destinazione) il pacchetto subisce vari tipi di ritardo. I principali ritardi sono dovuti al ritardo di elaborazione, ritardo di accodamento, ritardo di trasmissione e ritardo di propagazione che complessivamente formano il ritardo totale di nodo.

La figura 1.16 mostra un ottimo schema che riassume i tipi di ritardo.

Ritardo di elaborazione

Il tempo richiesto per esaminare l'intestazione del pacchetto e per determinare dove dirigerlo, nonché il tempo richiesto per controllare eventuali errori al suo interno fa parte del ritardo di elaborazione.

Ritardo di accodamento

Una volta in coda, il pacchetto subisce un ritardo di accodamento mentre attende la trasmissione sul collegamento. Ne abbiamo già parlato in 1.3.1 "ritardi di accodamento e perdita di pacchetti"

Ritardo di trasmissione

Sia L la lunghezza del pacchetto in bit e R la velocità di trasmissione in bps dal router A al router B, il ritardo di trasmissione è $\frac{L}{R}$ secondi e risulta essere il tempo richiesto per trasmettere tutti i bit del pacchetto sul collegamento.

Ritardo di propagazione

Una volta immesso sul collegamento, il pacchetto deve propagarsi fino al successivo nodo della rete. Questo tempo si chiama ritardo di propagazione. Il ritardo di propagazione è dato da d/v dove d è la distanza tra i due router e v è la velocità di propagazione nel collegamento (in m/s). Questo ritardo è puramente fisico: indica quanto tempo impiega un singolo bit a "viaggiare" dal mittente al destinatario una volta che è stato trasmesso. Questo ritardo solitamente è trascurabile.

Confronto tra ritardi di trasmissione e di propagazione

Il ritardo di trasmissione è la quantità di tempo impiegata dal router per trasmettere in uscita il pacchetto ed è funzione della lunghezza del pacchetto e della velocità di trasmissione sul collegamento, ma non ha niente a che fare con la distanza tra i due router. Come detto poc'anzi, è il tempo richiesto per trasmettere tutti i bit del pacchetto sul collegamento.

Il ritardo di propagazione, invece, è il tempo richiesto per la propagazione di un solo bit da un router a quello successivo ed è funzione della distanza tra i due router e della velocità di propagazione nel mezzo trasmissivo ma non ha niente a che fare con la dimensione del pacchetto o la velocità di trasmissione.

Nota come velocità di trasmissione $\neq$ velocità di propagazione.

	Nome (simbolo)	Significato	Unità tipica
1	**Velocità di propagazione** (v)	Velocità con cui **un segnale elettrico/ottico** si propaga fisicamente lungo il mezzo (rame, fibra, aria, ecc.).	metri/secondo (m/s)
2	**Velocità di trasmissione** (o *data rate*) (R)	Velocità con cui **i bit vengono trasmessi** sul collegamento, cioè quanti bit al secondo vengono "iniettati" nel canale.	bit/secondo (bps)

Table 1.1: Confronto tra velocità di propagazione e velocità di trasmissione.

Formula

Siano d_{elab}, d_{acc}, d_{tra}, d_{prop}, i ritardi, rispettivamente di elaborazione, accodamento, trasmissione e propagazione. Allora, il ritardo totale di un nodo è dato da:

$$d_{elab} + d_{acc} + d_{tra} + d_{prop}$$

1.4.2 Ritardo di accodamento e perdita di pacchetti

A differenza degli altri tre ritardi, quello di accodamento può variare da pacchetto a pacchetto. Ciò vuol dire che mentre elaborazione, trasmissione e propagazione sono costanti per ogni pacchetto che viaggia su quella specifica tratta, il ritardo di accodamento è variabile. Per tale motivo, nel caratterizzare il ritardo di accodamento si fa uso di misure statistiche quali il ritardo di accodamento medio e la sua varianza.

Denotiamo con a la velocità di arrivo di pacchetti nella coda espressa in pacchetti al secondo, R la velocità di trasmissione alla quale i bit vengono trasmessi in uscita alla coda espressa in bit al secondo e supponiamo che tutti i pacchetti abbiano dimensione L bit. Pertanto arrivano al nodo La bit/s.

Con La/R si denota l'intensità di traffico e il risultato è un numero adimensionale (es. $\frac{500bps}{250bps} = 2$)

- **(1)** Se $La/R > 1$ vuol dire che stanno entrando più pacchetti di quanti il nodo può trasferirne in uscita e la coda cresce verso ∞ bit.

  Questo perché $\frac{La}{R} > 1$ implica che $La > R$

- **(2)** Se $La/R \leq 1$ succede quanto segue: Innanzitutto nota che se ogni pacchetto ha dimensione L vuol dire che il nodo impiega $\frac{L}{R}$ secondi per smaltirlo. Ora ipotizziamo che N pacchetti giungano simultaneamente ogni $N(\frac{L}{R})$ secondi. Se ogni pacchetto viene smaltito ogni $\frac{L}{R}$ secondi, in $N(\frac{L}{R})$ secondi ne vengono smaltiti esattamente N.

  In questo caso il primo pacchetto inizia a uscire dalla coda immediatamente, il secondo pacchetto deve aspettare L/R secondi prima che il suo primo bit esca dalla coda, il terzo deve aspettare $2 * \frac{L}{R}$, il quarto deve aspettare $3 * \frac{L}{R}$ secondi e così via. Più in generale, il pacchetto N° deve aspettare $(N-1) * \frac{L}{R}$.

  Per rendere più chiaro il concetto facciamo un esempio. Si supponga che arrivino 10 pacchetti all'istante 0 ognuno grande 50bit con una velocità di trasmissione R pari a 1000bit/s. Il primo pacchetto inizia a uscire dalla coda subito, ma il primo bit del secondo pacchetto deve aspettare $\frac{L}{R} = \frac{50}{1000} = 0.05s$ prima che l'intero pacchetto precedente sia completamente uscito. Il decimo pacchetto deve aspettare $(9) * 0.05 = 0.45s$ e, nel momento in cui il suo trasferimento sul collegamento sarà terminato, a tempo 0.5, ne subentrano altri 10, di cui il primo non deve aspettare nulla prima che inizia a essere trasferito.

  Ciò dimostra che la condizione $La/R \leq 1$, non garantisce assenza di attese.

Perdita di pacchetti

La quantità di pacchetti perduti aumenta in proporzione all'intensità di traffico in quanto la coda potrebbe non essere abbastanza capiente per soddisfare la richiesta. Pertanto, le prestazioni di un nodo sono misurate non solo in termini di ritardo ma anche dalla probabilità di perdita di pacchetti.

1.4.3 Ritardo end to end

Nella sottosezione 1.3.1 (paragrafo "trasmissione store e forward") abbiamo detto che, supponendo di avere N-1 nodi tra sorgente e destinazione (e quindi N collegamenti) il ritardo end-to-end è

$$d_{e2e} = N(d_{trasmissione}) = N(\frac{L}{R})$$

ma prendevamo in considerazione solo il ritardo di trasmissione. La formula generale è

$$d_{e2e} = N(d_{elab} + d_{accodamento} + d_{trasmissione} + d_{prop})$$

Traceroute

Traceroute è un programma eseguibile su qualsiasi host di Internet. Quando l'utente specifica il nome di un host di destinazione, il programma invia pacchetti speciali verso di essa, che, durante il loro percorso, passano attraverso una serie di router. Quando uno di questi router riceve questo pacchetto speciale, invia un messaggio che torna alla destinazione che contiene il nome e l'indirizzo del router.

Il funzionamento è il seguente: Supponiamo di avere N-1 router dalla sorgente alla destinazione. La sorgente invia N pacchetti ognuno dei quali contiene l'indirizzo della destinazione. Quando l' n-esimo router riceve il pacchetto marcato come n, anziché indirizzarlo verso la destinazione risponde al mittente come poc'anzi descritto. L'N-esimo pacchetto, che raggiunge la destinazione, invita anche quest'ultima a rispondere.

In questo modo l'origine può ricostruire il percorso intrapreso dai pacchetti ed è inoltre in grado di determinare i ritardi per ogni nodo.

Sistemi periferici, applicazioni e altri ritardi

Oltre ai ritardi descritti nelle sezioni precedenti, si potrebbero manifestare ulteriori ritardi nei sistemi periferici. A titolo esemplificativo, un sistema periferico può volontariamente ritardare la sua trasmissione in quanto condivide il mezzo trasmissivo con altri sistemi periferici.

Un altro esempio è nel VoIP: In VoIP il mittente deve prima di tutto riempire il pacchetto con conversazione digitalizzata prima di inviarlo su Internet. Questo tempo per riempire un pacchetto è detto ritardo di pacchettizzazione.

1.4.4 Throughput nelle reti di calcolatori

Un'altra misura critica delle prestazioni in una rete di calcolatori è il throughput end-to-end.

Consideriamo un trasferimento di file da A a B. Il throughput istantaneo in ogni istante di tempo è la velocità di bit/s alla quale B sta ricevendo il file. È misurato in B e non in A in quanto è l'unico modo per individuare la velocità effettiva di trasferimento dopo tutti i ritardi e colli di bottiglia che intercorrono nel percorso.

Se il file consiste di F bit e il trasferimento richiede T secondi affinché la destinazione riceva tutti i bit, allora il throughput medio è F/T bit/s.

Supponiamo che A e B si stiano trasferendo un file e che tra di loro ci sia un router C. Supponiamo R_1 essere il throughput da A a C e R_2 il throughput da C a B. Se $R_1 \leq R_2$ allora la velocità di collegamento da A a B (throughput end to end) è R_1 ma se $R_1 > R_2$ la velocità di collegamento è R_2 in quanto il router fa da collo di bottiglia. Quindi il throughput end to end è $MIN(R_1, R_2)$ o più in generale $MIN(R_1, ..., R_{n+1})$ se ci sono n router.

1.5 Livelli dei protocolli e loro modelli di servizio

1.5.1 Architettura a livelli

Un'architettura a livelli consente di discutere e analizzare una parte specifica e ben definita di un sistema complesso. Ciò permette di introdurre un ulteriore vantaggio: la modularità, che rende molto più facile cambiare l'implementazione di un servizio fornito da un determinato livello. Fino a quando il livello fornisce lo stesso servizio allo strato superiore e utilizza gli stessi servizi dello strato inferiore, la parte rimanente del sistema (ovvero gli altri livelli) rimane invariata al variare dell'implementazione del livello.

Stratificazione dei protocolli

I protocolli, nonché l'hardware e il software che li implementano, sono organizzati in livelli. Ciascun protocollo appartiene a uno dei livelli. Ogni livello fornisce il suo servizio (1) effettuando determinate azioni all'interno del livello stesso e (2) utilizzando i servizi del livello immediatamente inferiore (se c'è).

Un livello può essere implementato via hardware, software o con una combinazione di essi.

L'insieme dei protocolli di tutti i livelli viene chiamato pila di protocolli. Internet è formato da una pila a 5 livelli. Questi livelli sono: fisico, collegamento, rete, trasporto e applicazione. In ambito accademico si studia anche la pila a 7 livelli denominata ISO/OSI.

Un protocollo, per sua natura, serve per scambiare messaggi. Pertanto, un protocollo "vive" su più sistemi di rete.

Degli svantaggi dell'architettura a strati sono la possibilità che un livello duplichi le funzionalità di un livello inferiore e che un livello possa prelevare informazioni da altri livelli bypassandone i servizi esposti. Ciò viola lo scopo della separazione tra livelli.

Livello di applicazione

Il livello di applicazione è la sede dei protocolli usati dalle applicazioni di rete. Alcuni dei protocolli a questo livello sono HTTP, SMTP e FTP. I pacchetti di informazione del livello di rete vengono denominati messaggi.

Livello di trasporto

Il livello di trasporto trasferisce i messaggi del livello di applicazione. Vi sono due protocolli di trasporto: TCP e UDP.

- TCP: fornisce alle applicazioni un servizio orientato alla connessione (cioè l'utente deve stabilire una connessione, usarla e quindi rilasciarla), garantisce la consegna dei messaggi di applicazione e il loro ordine. Gestisce il controllo di flusso (ovvero la corrispondenza tra le velocità del mittente e destinatario) e ha inoltre un controllo di congestione della rete.
- UDP: questo protocollo è molto semplice. Non è orientato alla connessione e non garantisce affidabilità, controllo di flusso e controllo della congestione.

I pacchetti a livello di trasporto sono denominati segmenti.

Livello di rete

Il livello di rete si occupa di trasferire i pacchetti a livello di rete, detti datagrammi, da un host a un altro.

Il livello di trasporto passa al livello di rete il proprio segmento e un indirizzo di destinazione.

Il livello di rete comprende il protocollo IP.

Livello di collegamento

Per trasferire un pacchetto da un nodo a quello successivo sul percorso, il livello di rete si affida ai servizi del livello di collegamento. Esempi di protocolli a livello di collegamento includono Ethernet e WiFi.

Un datagramma potrebbe essere gestito da differenti protocolli a livello di collegamento lungo le diverse tratte che costituiscono il suo percorso.

Chiameremo frame i pacchetti a livello di collegamento.

Livello fisico

Mentre il compito del livello di collegamento è spostare frame tra nodi adiacenti, il ruolo del livello fisico è spostare i singoli bit. I protocolli di questo livello dipendono dall'effettivo mezzo trasmissivo.

1.5.2 Incapsulamento

Dalla sorgente, i dati scendono lungo la pila dei protocolli e vi risalgono nella destinazione. Nei commutatori, i dati scendono e salgono fino a un determinato livello: per i commutatori a livello di collegamento fino al livello 2, per i router fino al livello 3. Gli host implementano tutti e cinque i livelli.

In altre parole, presso un host mittente, un messaggio a livello di applicazione viene passato al livello di trasporto. Questo livello prende il messaggio e gli concatena informazioni aggiuntive (le informazioni di intestazioni) che verranno utilizzate dal protocollo di trasporto nell'host ricevente. Messaggio livello applicazione + intestazioni del protocollo di trasporto costituiscono il segmento.

Il protocollo di trasporto passa il segmento al livello di rete, il quale gli concatena le proprie intestazioni, formando il datagramma e così via scendendo nella pila.

Quindi a ciascun livello il pacchetto ha due tipi di campi: quello di intestazione e quello di payload (il carico utile). Il payload è tipicamente un pacchetto proveniente dal livello superiore.

Capitolo 2

Livello di applicazione

Le applicazioni sono la ragion d'essere delle reti di calcolatori. Senza di esse predisporre una rete sarebbe inutile.

2.1 Principi delle applicazioni di rete

I programmi che fanno uso del livello Applicazione sono eseguiti sui sistemi periferici che comunicano tra loro via rete. Esempi di programmi sono i browser e i web server. I programmatori delle applicazioni a livello di rete non hanno bisogno di scrivere software che viene eseguito sui commutatori

2.1.1 Architettura delle applicazioni di rete

Per lo sviluppatore di applicazioni l'architettura di rete è fissa e fornisce alle applicazioni uno specifico insieme di servizi. Il compito dello sviluppatore è progettare l'applicazione. Egli deve scegliere tra utilizzare l'architettura client-server o P2P (peer-to-peer).

Nell'architettura client-server vi è un host sempre attivo, chiamato server, che risponde alle richieste di servizi da altri host, detti client. I client non si connettono direttamente fra loro.

In un'architettura P2P si sfrutta invece la comunicazione diretta di host, chiamati peer, che a differenza di un server centralizzato possono (e spesso lo sono) collegati in maniera intermittente. I peer non appartengono a un fornitore di servizi ma appartengono agli utenti. Uno dei punti di forza dell'architettura P2P è la sua intrinseca scalabilità e sono anche economicamente convenienti perché non richiedono server prestanti nè una banda elevata.

Alcune applicazioni presentano un'architettura ibrida, combinando sia elementi P2P che client-server.

2.1.2 Processi comunicanti

I processi su due sistemi terminali comunicano scambiandosi messaggi attraverso la rete. Il processo mittente crea e invia messaggi sulla rete e il processo destinatario li riceve e, quando previsto, invia messaggi di risposta.

Processi client e server

Per ciascuna coppia di processi comunicanti ne etichettiamo uno come client e l'altro come server. In alcune applicazioni un processo può essere sia client che server (come ad esempio in un programma P2P). Nonostante ciò possiamo comunque etichettarli basandoci sul contesto di una specifica sessione.

Un processo che richiede il servizio o le informazioni è indicato come client mentre quello che eroga il servizio o procura le informazioni è detto server.

L'interfaccia tra il processo e la rete

Ogni messaggio inviato da un processo a un altro remoto deve passare attraverso la rete sottostante. Il processo presuppone l'esistenza di un'infrastruttura esterna che trasporterà il messaggio attraverso la rete.

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 quello di trasporto all'interno di un host. Viene chiamata anche API tra l'applicazione e la rete.

Il progettista di un'applicazione può (1) scegliere il protocollo di trasporto e (2) talvolta determinare alcuni parametri a livello di trasporto.

Indirizzamento

I processi riceventi devono poter avere un indirizzo per ricevere i messaggi inviati da un processo in esecuzione su un altro host. Per identificare un processo ricevente è necessario specificare due informazioni: (1) l'indirizzo dell'host e (2) un identificatore del processo ricevente sull'host di destinazione. Il punto (2) serve perché su un host potrebbero esserci più processi in esecuzione simultaneamente.

In Internet gli host vengono identificati attraverso un indirizzo IP, composto da 32 bit, che per il momento, possiamo pensare che identificano univocamente un host. Il processo mittente deve anche identificare il processo destinatario, più specificatamente, la socket che deve ricevere il dato. Un numero di porta di destinazione assolve questo compito. Alle applicazioni più note sono stati assegnati numeri di porta specifici.

2.1.3 Servizi di trasporto disponibili per le applicazioni

Il protocollo a livello di trasporto ha la responsabilità di consegnare i messaggi alla socket del processo ricevente. I protocolli di trasporto lo possiamo classificare in 4 dimensioni: trasferimento dati affidabile, throughput, temporizzazione e sicurezza

Trasferimento dati affidabile

In alcune applicazioni la perdita di informazioni potrebbe causare gravi conseguenze. Dunque, per supportare tali applicazioni occorre garantire che i dati inviati siano consegnati corretti e completi. Se un protocollo fornisce ciò si dice che fornisce un trasferimento dati affidabile.

In questo caso il processo mittente può passare alla socket i messaggi e sapere con assoluta certezza che verranno recapitati al processo ricevente.

Invece, le applicazioni che tollerano le perdite sono le applicazioni audio/video.

Throughput

Un protocollo a livello di trasporto potrebbe fornire un throughput garantito. Con tale servizio l'applicazione potrebbe chiedere che il throughput sia almeno di r bps.

Le applicazioni che hanno requisiti di throughput si dicono applicazioni sensibili alla banda, che si oppongono alle cosiddette applicazioni elastiche che possono funzionare anche senza garanzia di throughput.

Temporizzazione

Un protocollo a livello di trasporto potrebbe anche fornire garanzie di temporizzazione, ovvero garantire che ogni bit che il mittente invia venga consegnato al destinatario in non più di t secondi.

Sicurezza

Un protocollo a livello di trasporto può fornire a un'applicazione servizi di sicurezza, per esempio cifratura e decifratura dei dati così come integrità e autenticazione.

2.0.1 2.1.4 Servizi di trasporto offerti da internet

Nelle sezioni precedenti abbiamo dato una panoramica dei servizi che un protocollo a libello di trasporto potrebbe fornire in teoria, ma Internet non fornisce tutti e 4.

In Internet ci sono due protocolli di trasporto: TCP e UDP.

Servizi di TCP

TCP prevede un servizio orientato alla connessione e il trasporto affidabile dei dati.

- Servizio orientato alla connessione: TCP fa in modo che client e server si scambino informazioni di controllo prima che i messaggi a livello di applicazione comincino a fluire. Questa procedura, detta handshaking, prepara gli host alla trasmissione dei pacchetti. Dopo la fase di handshaking c'è la vera e propria connessione TCP tra le socket che è full-duplex, ovvero possono scambiarsi contemporaneamente messaggi. L'applicazione deve chiudere la connessione quando termina di inviare messaggi.
- Servizio di trasferimento affidabile: i processi comunicanti possono contare su TCP per trasportare dati senza errori e nel giusto ordine.

TCP prevede anche un meccanismo di controllo della congestione che beneficia l'intero Internet, non solo i processi comunicanti.

Servizi di UDP

UDP è un protocollo di trasporto leggero senza connessione e dunque non necessita di handshaking. Non offre alcun tipo di affidabilità: il protocollo non garantisce che un certo messaggio arrivi al destinatario, e se lo fanno, potrebbero non giungere in ordine. Non include neanche un meccanismo di controllo della congestione.

Servizi non forniti dai protocolli di trasporto di Internet

Garanzie di throughput e temporizzazione non sono forniti dagli odierni protocolli di trasporto di Internet.

2.1.5 Protocolli a livello di applicazione

Un protocollo a livello di applicazione definisce come i processi di un'applicazione, in esecuzione su sistemi periferici diversi, si scambiano messaggi.

Un protocollo a livello di applicazione definisce:

- I tipi di messaggio scambiati (es. richiesta o risposta)
- La sintassi dei messaggi
- La semantica dei messaggi
- Le regole per determinare quando e come un processo invia e risponde ai messaggi

È importante distinguere tra applicazioni di rete e protocolli a livello di applicazioni di rete. Un protocollo è solo una parte di un'applicazione. Il Web è un'applicazione di rete che consiste in uno standard per i formati dei documenti, il browser, il web server e infine il protocollo a livello di applicaione: HTTP.

2.2 Web e HTTP

2.2.1 Panoramica di HTTP

HTTP è un protocollo a livello di applicazione del Web. Questo protocollo è implementato in due programmi: client e server.

Una pagina Web, detta anche documento, è costituita da oggetti. Un oggetto è semplicemente un file (es. una pagina HTML o un'immagine) indirizzabile tramite URL. La maggior parte delle pagine web consiste in un file HTML principale e diversi oggetti referenziati da esso.

Ogni URL ha almeno due componenti: il nome dell'host del server che ospita l'oggetto e il percorso dell'oggetto.

$$\underbrace{www.school.edu}_{Nome\ dell'host}\ /\ \underbrace{chemistry/lesson1.jpg}_{percorso\ dell'oggetto}$$

Un browser implementa il lato client di HTTP e un web server implementa il lato server di HTTP che ospita oggetti web indirizzabili tramite URL.

HTTP definisce in che modo i client web richiedono le pagine ai web server e come quest'ultimi le trasferiscono ai client.

HTTP utilizza TCP come protocollo di trasporto. In questo modo HTTP non si deve preoccupare dei dati smarriti, di perdite e riordinamento dei pacchetti.

Il client invia richieste e riceve risposte HTTP tramite la propria interfaccia socket. Analogamente, il server riceve richieste e invia risposte tramite la propria interfaccia socket.

Il server invia i file richiesti al client senza memorizzare alcune informazione di stato a proposito del client. Per questo motivo viene detto protocollo senza memoria di stato (stateless).

2.2.2 Connessioni persistenti e non persistenti

Ciascuna coppia richiesta/risposta deve essere inviata su una connessione TCP separata o devono essere inviate tutte sulla stessa connessione? Nel primo caso si dice che l'applicazione usa connessioni non persistenti, nel secondo caso usa connessioni persistenti.

HTTP con connessioni non persistenti

Ecco cosa avviene quando HTTP usa connessioni non persistenti, supponendo che il client chieda la seguente pagina Web: www.school.edu/chemistry/index.html

1. Il processo client HTTP inizializza una connessione TCP con il server sulla porta 80.
2. Il client HTTP invia al server un messaggio di richiesta che include il percorso /chemistry/index.html
3. Il processo server recupera l'oggetto /chemistry/index.html, lo incapsula in un messaggio HTTP e viene inviato al client attraverso la socket
4. Quando il server si è assicurato che il messaggio è arrivato alla destinazione, comunica a TCP di chiudere la connessione.
5. Il client riceve il messaggio di risposta e la connessione TCP termina

Come abbiamo visto, ciascuna connessione TCP trasporta solo un messaggio di richiesta e uno di risposta. Se la pagina index.html avesse contenuto 10 oggetti al suo interno, avrebbe generato 11 connessioni TCP, ognuna per ciasun oggetto.

Definiamo essere Round Trip Time (RTT) il tempo che intercorre tra l'invio di un pacchetto e la ricezione della risposta. Quando il client vuole ottenere una risorsa, apre una connessione TCP con il server il quale risponde con una conferma. Il tempo per fare ciò è RTT. Dopodiché il client manda un messaggio al server che contiene (1) a sua volta una conferma di avvenuta ricezione e (2) la richiesta HTTP della risorsa. A questo punto il server risponde con il file HTML. Il tempo per fare ciò è un altro RTT.

Il processo richiede 2 RTT (prima fase e seconda fase) + il tempo per trasfrerire la pagina HTML (tempo non trascurabile).

HTTP con connesioni persistenti

Le connessioni non persistenti presentano alcuni limiti: il primo è che per ogni oggetto richiesto occorre stabilire e mantenere una nuova connessione. In secondo luogo ciascun oggetto subisce un ritardo di 2 RTT. Il primo per stabilire la connessione TCP il secondo per richiedere e ricevere un oggetto.

Con HTTP a connessione persistente il server lascia la connessione TCP aperta dopo l'invio di una risposta, per cui le richieste e le risposte successive useranno la stessa connessione. In questo modo il server può anche spedire più oggetti contemporaneamente. Le richieste possono infatti essere fatte una di seguito all'altra senza aspettare prima il completamento delle richieste precedenti (pipelining). Il server HTTP chiude la connessione TCP quando essa rimane inattiva per un dato lasso di tempo. La modalità di default di HTTP impiega connessioni persistenti con pipelinig.

2.2.3 Formato dei messaggi HTTP

Le specifiche HTTP includono la definizione dei due formati HTTP: richiesta e risposta.

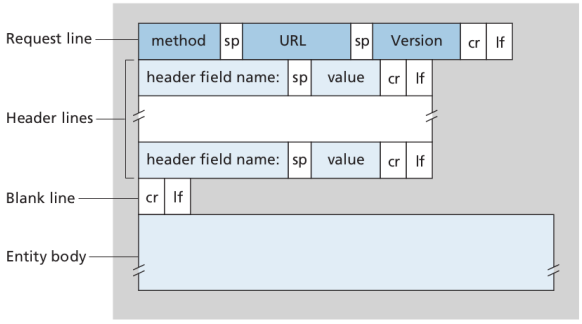
Messaggio di richiesta HTTP

Un messaggio di richiesta HTTP è

```
GET /somedir/index.html HTTP/1.1 \r\n
Host:  www.school.edu \r\n
Connection:  close \r\n
User-agent:  Mozilla/5.0 \r\n
Accept-language:  it \r\n
```

Notiamo fin da subito che il testo è scritto in codifica ASCII. Consiste di righe, ognuna delle quali con dei ritorni a capo. Un messaggio di richiesta può essere costituito da un numero indefinito di righe, anche una sola (la prima è obbligatoria). La prima riga è definita riga di richiesta mentre le altre sono righe di intestazione (header).

- La prima riga presenta tre campi: il campo metodo, il campo URL e il campo versione di HTTP. Il campo metodo è GET quando viene richiesto un oggetto, POST quando l'utente riempie un form (l'utente sta ancora richiedendo una pagina web al server, ma i contenuti specifici della pagina dipendono da ciò che l'utente ha inserito). Si può usare GET anche per riempire un form, includendo i dati nell'URL (query string). Poi vi è il metodo HEAD, e quando un server riceve un metodo di questo tipo risponde ma tralascia gli oggetti richiesti. PUT consente di inviare un oggetto a un percorso specifico e DELETE permette di cancellare un oggetto su un server. Nota: queste sono solo convenzioni.
- Host specifica l'host su cui risiede l'oggetto. Si potrebbe pensare sia seperflua dato che già in corso una connessione TCP con l'host ma quest'informazione serve per la cache dei proxy
- Con la terza linea il browswer specifica che vuole che il server chiuda la connessione dopo aver inviato l'oggetto richiesto.
- User-agent specifica quale browser sta effettuando la richiesta
- Accept-language specifica la lingua richiesta.



Formato generale dei messaggi di richiesta HTTP

Il corpo è (solitamente, per convenzione) vuoto nelle richieste GET.

Messaggio di risposta HTTP

Un esempio di messaggio di risposta HTTP è

```
HTTP/1.1 200 OK
Connection:  close
Date:  18 Aug 2025
Server:  Apache Last-Modified:  11 Jun 2025 Content-Length:  3145
Content-Type:  text/html
<html>...</head>
```

Troviamo la riga di intestazione, un numero variabile di intestazioni e il corpo.

- La riga di stato presenta tre campi: la versione del protocollo, un codice di stato e un messaggio relativo al codice di stato
- Connection: close sta ad indicare che il server ha intenzione di chiudere la connessione TCP dopo l'invio del messaggio
- Date indica la data di invio del messaggio
- Server indica il tipo di server che sta fornendo il messaggio
- Last-Modified indica quando l'oggetto è stato modificato per l'ultima volta
- Content-Length indica quanti byte è il corpo
- Content-Type indica il tipo del corpo, in questo caso un file HTML.

2.2.4 Interazione utente-server: i cookie

Abbiamo detto che HTTP è stateless. Tuttavia, è spesso auspicabile che i web server possano tener traccia degli utenti per limitare l'accesso ad alcune pagine a taluni o per fornire contenuti in funzione della loro identità.

A questo scopo HTTP adotta i cookie, che consentono ai server di tener traccia degli utenti.

Per poter utilizzare i cookie sono necessari 4 componenti:

- Una riga di intestazione nel messaggio di risposta
- Una riga di intestazione nel messaggio di richiesta
- Un file mantenuto sul sistema dell'utente gestito dal browser
- Un database sul server

Il server crea un identificativo univoco e lo inserisce all'interno del proprio database. A questo punto il server risponde al browser inserendo nel messaggio HTTP un header chiamato Set-cookie che contiene il numero identificativo. Quando il browser riceve il messaggio, aggiunge una riga nel proprio file. Questa riga contiene il nome del server e il numero identificativo.

Ogni volta che l'utente richiede una pagina web, il suo browser consulta il suo file dei cookie, estrae il suo numero identificativo per il sito e lo pone nella richiesta HTTP. In tal modo il server può monitorare l'attività dell'utente nel sito.

2.2.5 Web caching

Una web cache è un'entità di rete che soddisfa richieste HTTP al posto del web server effettivo. Il proxy ha una propria memoria in cui conserva copie di oggetti recentemente richiesti.

Ecco cosa succede:

1. Il browser stabilisce una connessione TCP con il proxy serber e gli invia una richiesta HTTP per l'oggetto specificato.
2. Il proxy controlla la presenza di una copia dell'oggetto memorizzato localmente. Se l'oggetto viene rilevato, il proxy lo inoltra al client
3. Se la cache non dispone dell'oggetto, apre una connessione con il server di origine, richiedendo l'oggetto in questione. Il server gli risponde
4. Il proxy salva l'oggetto nella propria cache e lo manda al client.

Il proxy è contemporaneamente sia client che server a seconda dei casi.

I proxy possono (1) ridurre i tempi di risposta ai client e (2) riducono il carico dei web server e dei loro collegamenti. Ciò implica che l'intero Internet ne trae beneficio.

GET condizionale

L'oggetto ospitato nel web server potrebbe essere stato modificato rispetto alla copia presente nel proxy. HTTP presenta un meccanismo che permette alla cache di verificare se i suoi oggetti sono aggiornati. Questo meccanismo è chiamato GET condizionle.

Un messaggio di richiesta HTTP viene chiamato messaggio di GET condizionale se (1) usa il metodo GET e (2) include una riga di intestazione If-modified-since.

Vediamone il funzionamento:

1. Un proxy invia un messaggio di richiesta a un web server per conto di un browser richiedente
2. In cache memorizza sia l'oggetto che la data di ultima modifica presente nell'intestazione Last-Modified
3. Se dopo un certo periodo di tempo un client richiede la stessa risorsa, la cache non può rispondere direttamente con quella che ha in memoria perché potrebbe essere stata modificata. Allora la cache effettua un controllo di aggiornamento inviando una GET condizionale al server, specificando nell'header If-modified-since la data che ha in memoria precedentemente prelevata dall'header Last-Modified. Questo comunica al server di inviare l'oggetto solo se è stato modificato rispetto alla data specificata
4. Se è stato modificato, il server risponde con un messaggio HTTP normale compreso di oggetto. Altrimenti manda un messaggio HTTP con corpo vuoto (per non sovraccaricare inutilmente la banda) con messaggio di stato 304 Not Modified che comunica al proxy che può procedere a inoltrare al client la copia dell'oggetto presente in cache
5.