

Livello applicazione: Il Web ed il protocollo HTTP

Ing. Nadia Ranaldo

1

Il livello applicazione

Obiettivi:

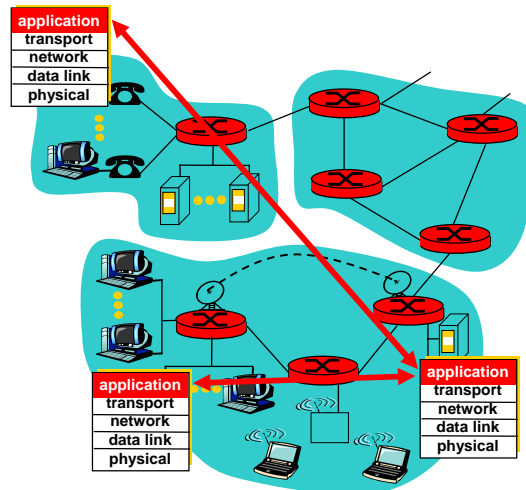
- Aspetti concettuali/ implementativi dei protocolli applicativi
- Protocolli specifici:
 - HTTP, FTP, SMTP, POP, DNS
- Programmazione di applicazioni di rete
 - Uso delle socket

2

Il livello applicazione

■ Diffuse applicazioni di rete:

- Web
- posta elettronica
- messaggistica istantanea
- autenticazione in un calcolatore remoto (Telnet e SSH)
- condivisione di file P2P
- trasferimento di file su due calcolatori (FTP)
- giochi multiutente via rete
- streaming di videoclip memorizzati
- telefonia via Internet
- videoconferenza in tempo reale



3

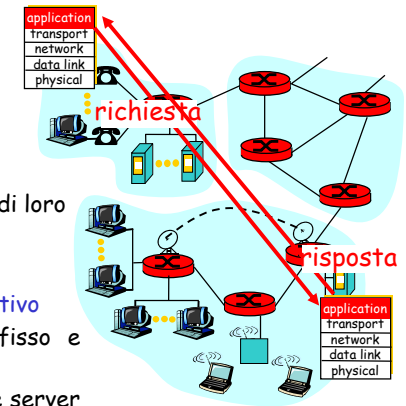
Architettura client-server

Client:

- Inizia il dialogo col server
- Può essere attivo saltuariamente
- Di solito richiede un servizio
- Nel caso del Web, il client è integrato nel browser
- I client non comunicano direttamente tra di loro

Server:

- Fornisce il servizio al client, su richiesta
- Tipicamente si trova su un host **sempre attivo**
- Il server dispone di un indirizzo IP fisso e conosciuto
- Uso di server farm per creare un potente server virtuale
- Es., un Web server invia una pagina Web richiesta, un mail server accede alla casella di posta elettronica

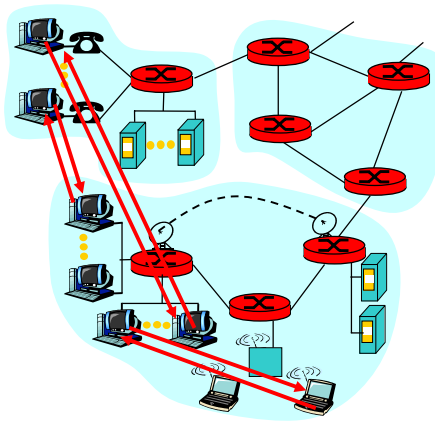


4

Architettura peer-to-peer pura

- No server sempre attivi
- Coppie arbitrarie di host (peer) che comunicano direttamente
- I peer possono cambiare indirizzo IP
- Esempio: Gnutella

Altamente scalabile (milioni di peer) ma difficile da gestire (natura distribuita e decentralizzata)



5

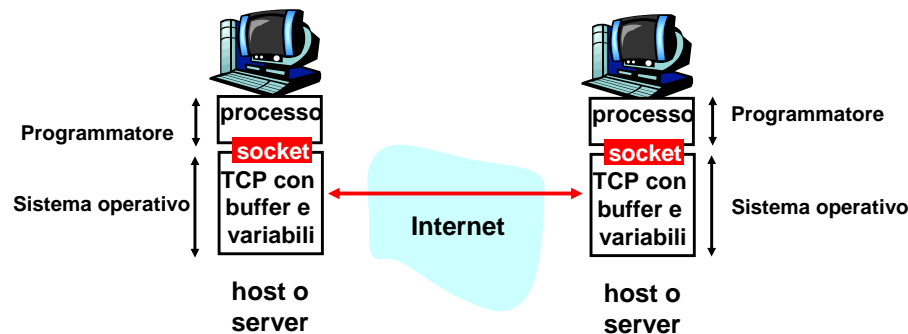
Applicazioni e protocolli applicativi

- **Applicazione: processi distribuiti in comunicazione tra loro**
 - In esecuzione su host remoti
 - Si scambiano messaggi per eseguire l'applicazione
 - Es., posta, FTP, WWW
- **Protocolli applicativi**
 - Costituiscono una parte di ogni applicazione di rete
 - Definiscono il formato dei messaggi scambiati e il loro significato
 - Usano i servizi degli strati inferiori
- Un **processo** è un programma in esecuzione su un host
- Sullo stesso host i processi comunicano mediante meccanismi di comunicazione interprocesso definiti dal SO
- Processi in esecuzione su host diversi comunicano mediante meccanismi definiti dal **protocollo dello strato di applicazione (application layer protocol)**
- **Processo client** (richiede un servizio) **processo server** (risponde ad una richiesta (coppia client-server anche in una sessione P2P))

6

Accesso ai servizi dei livelli inferiori: socket

- **Un processo invia e riceve messaggi mediante le socket**
 - interfaccia tra le applicazioni di rete e lo strato di trasporto all'interno di un host
 - API (Application Programming Interface) tra l'applicazione e la rete



7

Accesso ai servizi dei livelli inferiori: indirizzamento

- **Per identificare un processo destinatario:**
 - Identificare l'host: **Indirizzo IP** dell'host su cui il processo destinatario è in esecuzione (stringa univoca di 32 bit) oppure il suo indirizzo simbolico
 - Identificare il processo destinatario in esecuzione sull'host: **Numero di porta (port number)** - permette all'host mittente di identificare il processo locale destinatario del messaggio
 - Un numero tra 0 e 65535
 - Alle applicazioni più note sono assegnati numeri di porta specifici
 - I server Web sono identificati dal numero di porta 80
 - Il processo di un server di posta elettronica che usa il protocollo SMTP è identificato dal numero di porta 25

8

Requisiti delle applicazioni di rete

- Un'applicazione sceglie quale protocollo di trasporto utilizzare sulla base dei servizi di cui necessita

Trasferimento dati affidabile

- Alcune applicazioni (es., audio) sono tolleranti alla perdita di dati (fino a un certo punto)
- Altre invece richiedono una comunicazione assolutamente affidabile (posta elettronica, trasmissione di file, etc.)

Ampiezza di banda

- Alcune applicazioni (es. multimediali) richiedono una banda minima
 - In futuro si adotteranno tecniche di codifica adattativa per codificare a frequenze che rispettano l'ampiezza di banda disponibile
- Altre (dette "elastiche") usano la banda a disposizione (posta elettronica, Web, etc.)

Ritardo

- Alcune applicazioni (es., telefonia Internet, giochi interattivi in rete) richiedono stretti vincoli di temporizzazione sulla consegna dei dati
 - Lunghi ritardi nella telefonia Internet causano pause innaturali durante la conversazione

Requisiti di alcune applicazioni di rete

Application	Data loss	Bandwidth	Time Sensitive
file transfer	No	elastica	no
e-mail	No	elastica	no
Web documents	No	elastica	no
real-time audio/video	SI	audio: 5Kb-1Mb video: 10Kb-5Mb	SI, 100 ms
stored audio/video	SI	Come sopra	SI, pochi s
interactive games	SI	Fino a 10 Kbps	SI, 100 ms
chat	NO	elastica	SI e NO

I servizi di trasporto di Internet

Servizio TCP :

- Orientato alla connessione:* richiesto "setup" tra client e server
- Trasporto affidabile (reliable transfer)* tra processi mittente e ricevente
- Controllo di flusso (flow control):* il mittente non sovraccarica il ricevente
- Controllo della congestione (congestion control):* si limita il mittente quando la rete è sovraccarica
- Non offre:* garanzie di banda e ritardo minimi

Servizio UDP :

- Trasporto non affidabile* tra processi mittente e ricevente
- Non offre:* connessione, affidabilità, controllo di flusso, controllo di congestione, garanzie di ritardo e banda minima

Relazione tra applicazione e trasporto

Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	proprietario (es. RealNetworks)	TCP o UDP
Internet telephony	proprietario (es. Vonage, Dialpad)	tipicamente UDP

- Il servizio del World Wide Web (WWW) è realizzato mediante un programma client ed un programma server
- Il client ed il server eseguono su due host diversi e si scambiano messaggi HTTP (*HyperText Transfer Protocol*)
- Il protocollo HTTP stabilisce il formato e le modalità di scambio dei messaggi
- Il **programma client**, (o **user agent**) per il Web è detto **browser** e permette di visualizzare pagine Web:
 - MS Internet Explorer, Netscape Communicator
- Una **pagina Web** (documento) consiste di **oggetti**, ovvero file: tipicamente è costituita da un file base HTML e da diversi oggetti referenziati
 - immagini JPEG, immagini GIF, applet Java, una clip audio, etc.
- Il **programma server** per il Web è detto **server Web** memorizza documenti in formato HTML, immagini e altri oggetti accessibili da remoto mediante un **URI**
 - Apache (pubblico dominio), MS Internet Information Server

13

- **HyperText Markup Language**
- E' il linguaggio impiegato per scrivere documenti sul Web mediante la descrizione del contenuto semantico
 - **Markup** codice che mantiene informazioni sulla formattazione del testo
 - **Hypertext** il testo e' disseminato di **hyperlink**, ossia di punti speciali che ci permettono di collegarci ad un'altra pagina semplicemente cliccando su una sezione di testo, un'immagine, un bottone, ...
- Il linguaggio HTML si basa su coppie di **tag** di apertura e di chiusura che delimitano attributi e valori.
 - I tag definiscono ogni elemento di una pagina Web, quale un paragrafo di testo, una tabella o un'immagine
 - Indicano al browser il tipo delle componenti contenute all'interno di un file html
 - È compito poi del browser determinare come visualizzare al meglio i titoli, etc.

14

```
<html>
<head>
<title>Insegnamento di Elementi di Informatica a.a.
  2003/2004</title>
</head>
<body bgcolor="#CDE9F1">
<h2 align="center"><font color="#000000" face="Comic Sans
  MS">Universit&agrave; degli Studi del Sannio </font></h2>
<h2 align="center"><font color="#000000" face="Comic Sans
  MS">Facolt&agrave; di Ingegneria </font></h2>
...
</body>
</html>
```

15

- Un URI rappresenta un modo per localizzare una risorsa su una rete:
- Esistono due tipi di URI:
 - **URL: Uniform Resource Locator**
 - **URN: Uniform Resource Name**
- Un URL consente di individuare una risorsa presente in una specifica posizione usando la seguente sintassi
`<protocollo>://<nome del server>
 <[:porta]>/<percorso/><nome del file>#sezione`
- Protocollo:
 - file: un file sul proprio disco locale
 - ftp: un server FTP
 - http: un server WWW
 - etc.
- Il percorso si riferisce alla directory principale utilizzata per la gestione dei documenti
 - non è necessariamente la directory principale del file system
- La sezione rappresenta la parte del file da recuperare

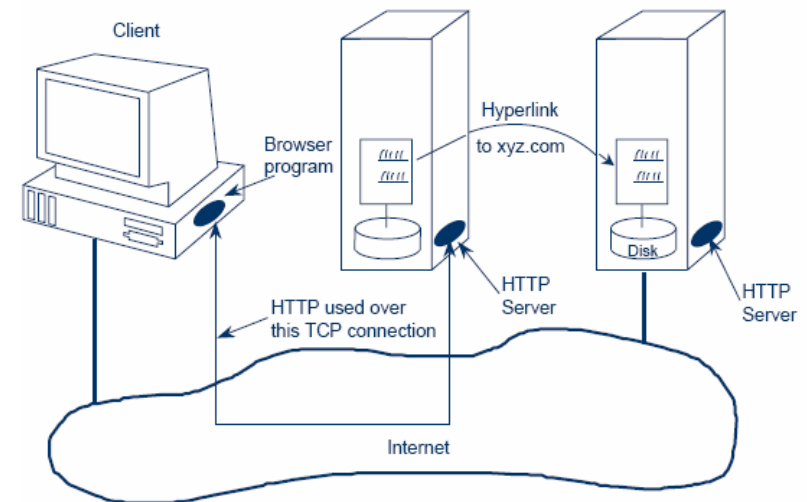
16

URI: Uniform Resource Identifier (2)

- Un **URL** può essere:
 - **URL assoluto**: completo
 - **URL relativo**: eredita il protocollo, il nome del server ed il percorso del suo documento padre
 - Permette di spostare o di copiare intere strutture di documenti HTML da un sito ad un altro senza disallineare tutti i link interni
- Un **URN** individua una risorsa con un nome globale, non fa alcun riferimento alla posizione
 - Usando un URN è possibile individuare una risorsa presente in diverse posizioni e che può spostarsi da un sito ad un altro
 - Per esempio, il recupero potrebbe avvenire dal server più vicino
- Le specifiche degli URL sono descritte dagli RFC 1630, 1738 e 1808
- Gli URN non hanno ancora specifiche e non sono implementati in nessun software di rete

17

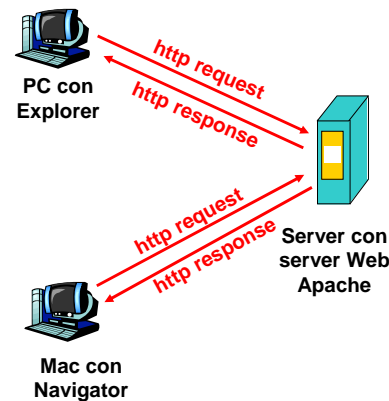
Web browser e server



18

Il protocollo HTTP

- Quando un client intende visualizzare una pagina Web, invia un messaggio HTTP di richiesta ad un server
- Il server risponde inviando un messaggio HTTP di risposta contenente gli oggetti richiesti
- Il protocollo HTTP utilizza TCP per il trasporto e l'interfaccia delle socket per la comunicazione basata su TCP
- Esistono due versioni di HTTP
 - HTTP /1.0 (RFC 1945)
 - HTTP /1.1 (RFC 2616)
- Un server HTTP non memorizza informazioni di stato associate ai client
 - I server HTTP sono *stateless*
- Le connessioni HTTP possono essere non persistenti (HTTP/1.0) o persistenti (HTTP/1.1) - inizio del 1998



19

Connessioni non persistenti

- Si consideri l'ipotesi di trasferire da uno stesso server una pagina Web contenente un numero n di altri oggetti
- Il protocollo HTTP/1.0 definisce una sequenza di passaggi per richiedere la pagina Web:
 1. Il client stabilisce una connessione TCP con il server
 2. Il client costruisce un messaggio HTTP, contenente l'URL (o la parte dell'URL) relativo alla posizione del documento richiesto, e lo invio al server
 3. Il server usa le informazioni contenute nel messaggio per cercare nella memoria locale la presenza del documento richiesto ed invia una risposta al client contenente il file
 4. Il server chiude la connessione TCP
 5. Il client riceve il messaggio di risposta, chiude la connessione ed esamina il documento Web
 - Dall'analisi del documento, il client riscontra la presenza di altri n oggetti da prelevare dallo stesso server
 - Per ogni altro degli n oggetti, vengono ripetuti i passi da 1 a 5

20

Connessioni non persistenti - esempio (1)

L'utente accede alla URL www.someSchool.edu/someDepartment/home.index

(contiene testo e
i riferimenti a 10
immagini jpeg)

1a. Il client HTTP inizia una connessione TCP verso il server (processo) http sull'host www.someSchool.edu. La porta 80 è quella standard (default) per i server HTTP

1b. Il server HTTP presso l'host www.someSchool.edu è "in ascolto" sulla porta 80. "Accetta" la richiesta di connessione e ne dà conferma al client

2. Il client http invia un messaggio di richiesta HTTP contenente l'URL

3. Il server HTTP riceve il messaggio di richiesta, costruisce un messaggio di risposta contenente l'oggetto richiesto (someDepartment/home.index), inoltra il messaggio attraverso la socket

Tempo

21

Connessioni non persistenti - esempio (2)

5. Il client HTTP riceve il messaggio di risposta contenente il file html, visualizza la pagina html. Analizzando il file html, il browser trova i riferimenti a 10 oggetti jpeg

4. Il server http chiude la connessione TCP (dopo che il client ha ricevuto interamente il messaggio di risposta).

6. I passi 1-5 sono ripetuti per ciascuno dei 10 oggetti jpeg

• Sono richieste 11 connessioni TCP per ottenere la pagina Web!

22

Connessioni non persistenti (1)

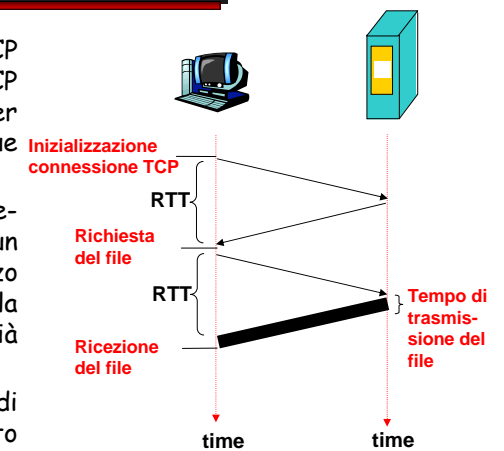
- Mediante l'uso delle connessioni non persistenti per ogni oggetto è necessario ristabilire la connessione (n+1 connessioni) con lo stesso server
 - Ogni connessione trasporta soltanto un messaggio di richiesta e uno di risposta
- Le n connessioni possono essere realizzate sia in modo sequenziale che parallelo
- Con l'esecuzione parallela si ottiene una maggiore velocità di risposta e si riesce a sfruttare meglio la banda del canale di comunicazione
 - Molti browser aprono da 5 a 10 connessioni TCP parallele e ciascuna di questa gestisce una transazione di richiesta-risposta
- Qual è la quantità di tempo che intercorre tra una richiesta di un oggetto e il momento in cui l'intero oggetto viene ricevuto dal client (transazione HTTP)?
 - Utilizziamo il concetto di **Round Trip Time (RTT)**: tempo necessario ad inviare un messaggio TCP ed ottenere il messaggio di risposta

23

Connessioni non persistenti (2)

- Per stabilire una connessione TCP sono necessari tre messaggi TCP (**three-way handshake**) per riscontrare la connessione nei due sensi
- I primi due messaggi del three-way handshake richiedono un tempo pari a 1 RTT, il terzo messaggio contiene l'ack per la connessione e può contenere già la richiesta HTTP
- Pertanto il tempo complessivo di una transazione HTTP è dato approssimativamente da:

$2 * RTT + \text{il tempo di trasmissione del file da parte del server}$



24

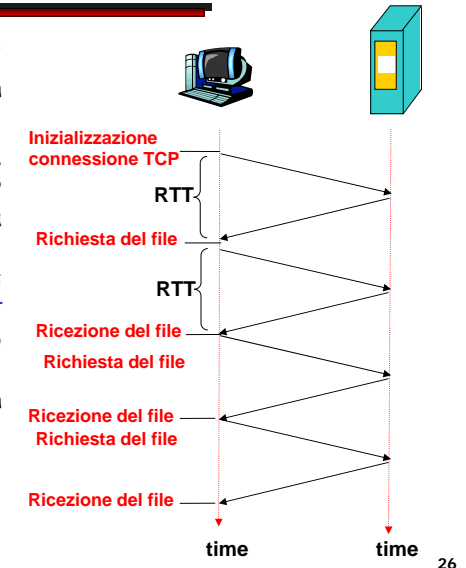
Connessioni persistenti (1)

- Le connessioni non persistenti presentano diversi inconvenienti:
 - Per ogni oggetto deve essere stabilita una nuova connessione
 - Ogni connessione consuma risorse sul server (si pensi che un server può ricevere centinaia di connessioni contemporaneamente)
 - Per ogni oggetto è richiesto un tempo di trasferimento pari a $2 \cdot RTT$
 - Trascurando il ritardo di trasmissione del file
 - La comunicazione su di una connessione TCP è avviata in modo lento usando l'algoritmo di "slow-start" per evitare la congestione: avere diverse connessioni significa incorrere ripetutamente nel ritardo dello "slow-start"
- Le connessioni persistenti consentono di superare alcuni di questi problemi:
 - Usando connessioni persistenti il server chiude la connessione solo dopo aver trasferito al client tutti gli oggetti
 - In realtà è difficile individuare la trasmissione dell'ultimo oggetto di un documento Web:
 - Il server chiude la connessione quando non riceve più richieste per un intervallo di tempo fissato (tempo di timeout, configurabile)
 - In questo modo è possibile trasferire anche più pagine Web sulla stessa connessione

25

Connessioni persistenti (2)

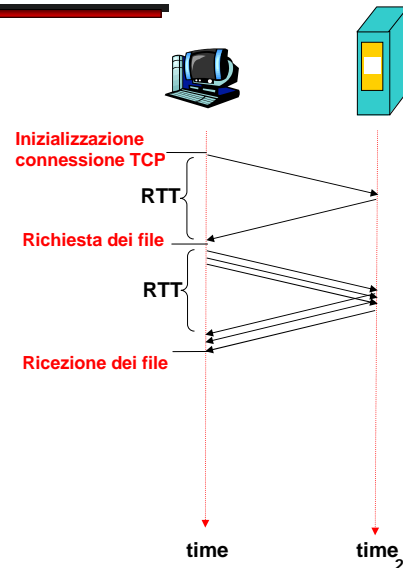
- Le connessioni persistenti si distinguono in connessioni con pipelining e senza pipelining
- In una connessione **senza pipelining**, una richiesta è inoltrata al server dopo aver ricevuto la risposta della precedente richiesta
 - Il tempo necessario per ogni oggetto diventa pari a $1 RTT$ (tranne che per il primo oggetto per il quale il tempo sarà $2 RTT$)
 - Il server è in attesa passiva (non fa nulla) per un tempo pari a RTT



26

Connessioni persistenti (3)

- Le cose migliorano con l'impiego del **pipelining**: il client inoltra una richiesta appena possibile (appena trova un riferimento)
 - Il tempo complessivo necessario a richiedere ed ottenere gli oggetti dal server è pari a $1 RTT$ (nel caso in cui RTT è sufficientemente ampio)
 - L'algoritmo slow-start non introduce molto ritardo: solo le prime richieste vengono rallentate
 - Il server è in attesa passiva per un tempo molto basso
 - Impiego migliore della banda
- HTTP/1.1, default: connessioni persistenti e con pipelining

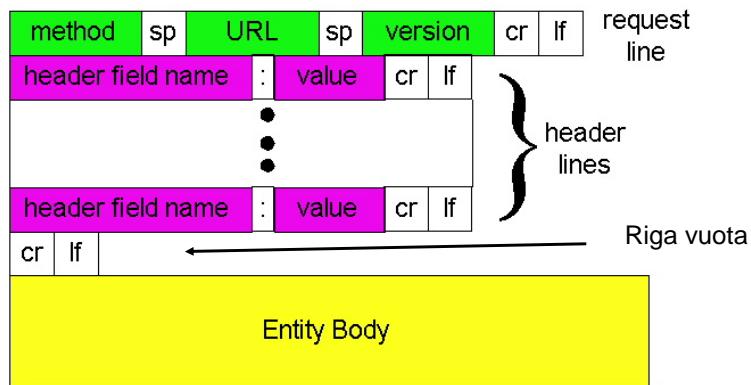


Formato dei messaggi HTTP: msg. di richiesta (1)

- Due tipi di messaggi http: *richiesta e risposta*
 - Messaggio HTTP di richiesta:
 - ASCII (formato testo leggibile)
- Chiudi la connessione al termine della richiesta
- Linea di richiesta (campo metodo: GET, POST, HEAD)
- ```
GET /somedir/page.html HTTP/1.1
```
- Linee di intestazione
- ```
Host: www.someschool.edu
Connection: close
User-agent: Mozilla/4.0
Accept: text/html, image/gif, image/jpeg
Accept-language: fr
```
- Carriage return, line feed ulteriore indica la fine delle linee di intestazione

28

Formato dei messaggi HTTP: msg. di richiesta (2)



29

Formato dei messaggi HTTP: msg. di richiesta (3)

- Il metodo **HEAD** è simile al metodo **GET**
 - Il messaggio di risposta non conterrà il documento richiesto ma un insieme di informazioni relative al documento (l'HEAD della pagina Web)
 - Ad esempio può essere utilizzato per verificare se una copia del file nella cache locale è ancora valida (verificando se la data dell'ultima modifica è cambiata)
- Il metodo **POST** è utilizzato per richiedere al server un oggetto che dipende dall'elaborazione di informazioni contenute nell'entity body
 - L'entity body non è usato con il metodo **GET**, ma solo con il metodo **POST**
 - È utilizzato per recuperare le informazioni inserite dall'utente in un form HTML
 - Ad esempio per una query inviata ad un motore di ricerca
 - Le richieste generate con un form possono utilizzare anche il metodo **GET**
 - I dati immessi nei campi del form sono inseriti nell'URL
 - Es. www.somesite.com/animalsearch?scimmie&banane
- Il metodo **PUT** permette di caricare un file su un server Web in un percorso specifico (directory) (HTTP/1.1)
- Il metodo **DELETE** permette di cancellare un oggetto su un server Web (HTTP/1.1)

30

Formato dei messaggi HTTP: msg. di risposta (1)

Riga di stato (versione del protocollo, codice di stato, messaggio di stato)

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 05 May 2005 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 2004 ...
Content-Length: 6821
Content-Type: text/html
```

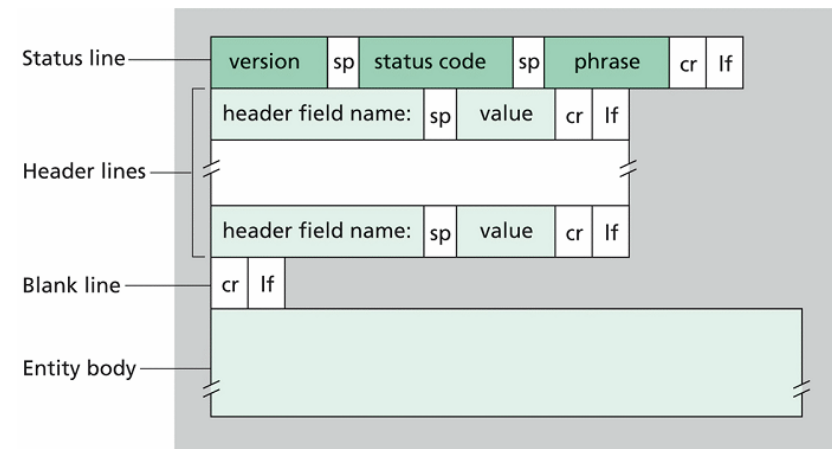
Linee di Intestazione
data di costruzione del messaggio

Corpo dell'entità,
ad es.
file html richiesto

- ```
data data data data data ...
```
- Client HTTP 1.0: Server chiude la connessione al termine della richiesta
  - Client HTTP 1.1: mantiene aperta la connessione oppure chiude se Connection: close

31

## Formato dei messaggi HTTP: msg. di risposta (2)



32

## Formato dei messaggi HTTP: msg. di risposta (3)

- **Codici di stato**
  - Prima riga del messaggio di risposta server->client
- **Alcuni esempi:**
  - 200 OK**
    - La richiesta ha avuto successo, l'oggetto richiesto è inviato nel messaggio di risposta
  - 301 Moved Permanently**
    - L'oggetto richiesto è stato spostato in modo permanente. Il nuovo indirizzo è specificato nell'intestazione (Location:). Il browser impiegherà il messaggio di risposta per inoltrare la richiesta alla nuova destinazione
  - 400 Bad Request**
    - Richiesta incomprensibile al server (codice di errore generico)
  - 404 Not Found**
    - Il documento non è stato trovato sul server
  - 505 HTTP Version Not Supported**
    - Il server non dispone della versione del protocollo HTTP richiesta

33

## Formato dei messaggi HTTP: msg. di risposta (4)

- I codici da 100 a 199
  - una risposta temporanea alla richiesta, durante il suo svolgimento
- I codici da 200 a 299
  - indicano che la richiesta è stata ricevuta ed accettata
- I codici da 300 a 399
  - il server ha ricevuto e capito la richiesta, ma sono necessarie altre azioni da parte del client per portare a termine la richiesta
- I codici da 400 a 499
  - indicano che il client ha commesso un errore nella preparazione del messaggio di richiesta (errore sintattico o richiesta non autorizzata)
- I codici da 500 a 599
  - La risposta può anche essere corretta, ma il server non è in grado di soddisfare la richiesta per un problema interno

34

## Formato dei messaggi HTTP: msg. di risposta (5)

- **100 Continue**
  - se il client non ha ancora mandato il body
- **201 Created**
  - PUT con successo
- **401 Unauthorized**
  - manca l'autorizzazione
- **403 Forbidden**
  - richiesta non autorizzabile
- **500 Internal server error**
  - tipicamente un programma in esecuzione sul server ha generato errore
- **501 Not implemented**
  - metodo non conosciuto dal server

35

## Prova (1)

- Eseguire una richiesta di connessione mediante Telnet verso il server Web [www.ing.unisannio.it](http://www.ing.unisannio.it) sulla porta 80
- `telnet www.ing.unisannio.it 80`
- Inviare una richiesta per richiedere la pagina Web `/ranaldo/reti2007/home.htm`
- `GET /ranaldo/reti2007/home.htm HTTP/1.1`  
`host:www.ing.unisannio.it (doppio invio)`
- Osservare il messaggio di risposta

36

## Prova (2)

```

Nadia@Nadia ~
$ telnet www.ing.unisannio.it 80
Trying 193.206.108.66...
Connected to web.ing.unisannio.it.
Escape character is '^]'.
GET /ranaaldo/reti2006/home.htm HTTP/1.1
host:www.ing.unisannio.it

HTTP/1.1 200 OK
Date: Mon, 03 Apr 2006 17:21:31 GMT
Server: Apache/2.0.52 (Red Hat)
Last-Modified: Tue, 28 Mar 2006 10:49:34 GMT
ETag: "1db149-442-d8c02b80"
Accept-Ranges: bytes
Content-Length: 1090
Content-Type: text/html; charset=ISO-8859-1
Connection: Keep-Alive

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<title>Insegnamento di Elementi di Informatica a.a. 2003/2004</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
</head>

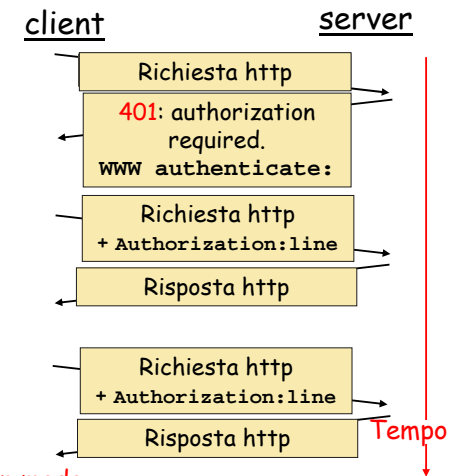
```

37

## Autenticazione delle richieste

**Obiettivo:** controllare l' accesso ai documenti sul server

- **stateless:** il client deve autenticare ogni richiesta
- Fase di autenticazione: tipicamente **log e password**
  - Senza autenticazione il server rifiuta la connessione  
WWW authenticate:
  - Inserire log e password nell'header
  - authorization: riga nell'header del messaggio di richiesta



**Il browser memorizza log e password in modo che l'utente non debba digitarli ogni volta (Attenzione: trasmessi in chiaro!)**

38

## Cookie (1)

- HTTP è stateless: il server non è tenuto a mantenere informazioni su connessioni precedenti
- Può essere utile che i server Web possano identificare gli utenti o per limitare l'accesso da parte di questi ultimi o per fornire contenuti in funzione della loro identità
- **Un cookie è una breve informazione scambiata tra il server ed il client**
  - Siti di e-commerce (ad esempio Amazon), siti pubblicitari, portali (yahoo, etc.)
- Tramite un cookie il client mantiene lo stato di precedenti connessioni, e lo manda al server di pertinenza ogni volta che richiede un documento
- Esempio: tramite un cookie si viene rediretti sulla pagina in Italiano tutte le volte che ci si ricollega allo stesso server
- I cookie sono definiti dell'RFC 2109 (su proposta di Netscape)

39

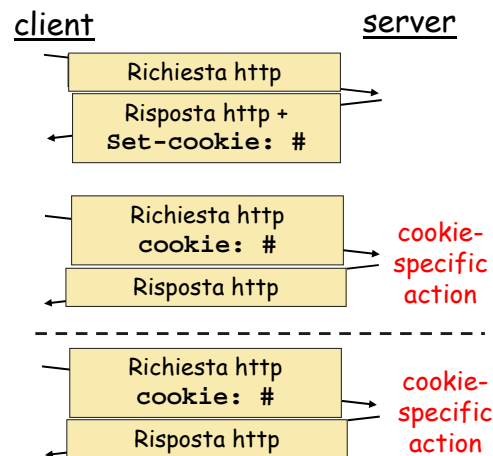
## Cookie (2)

- I cookie usano due header: uno per la risposta ed uno per le richieste successive:
- **Set-Cookie:** header della risposta, contiene un numero identificativo memorizzato sul server e che identifica una sessione di un utente
  - il client può memorizzarlo e rispedito alla prossima richiesta
- **Cookie:** header della richiesta
  - Il client decide se spedito sulla base del nome del documento, dell'indirizzo IP del server e dell'età del cookie
- Il browser memorizza un file di cookie
  - riga=host del server e numero identificativo sul server
- Sul server Web sono gestiti (mediante un database) i cookie associati a ciascun utente
- Un browser può essere configurato per accettare o rifiutare i cookie
  - Alcuni siti Web richiedono necessariamente la capacità del browser di accettare i cookie

40

## Cookie (3)

- Il server invia un "cookie" al client con la risposta  
Set-cookie: 1678453
- Il client presenta il cookie in accessi successivi (anche giorni dopo)  
cookie: 1678453
- I cookie possono essere usati per creare un livello di sessione utente al di sopra di HTTP privo di stato



41

## Cache - lato client (1)

- Un aspetto importante nelle interazioni distribuite è quello di minimizzare il traffico tra client e server e ridurre il tempo di risposta
- A tale scopo si utilizza il concetto di cache
- La cache può essere utilizzata sia dal lato client che dal lato rete
- Cache dal lato client** (dette cache private): gli oggetti ricevuti dai server Web sono memorizzati localmente in modo da evitare successive richieste ai server per gli stessi oggetti
  - Prima di inviare una richiesta, il client interroga la cache, se l'oggetto non è presente nella cache allora viene inviata la richiesta al server
- La cache nei client introduce, però, un **problema di consistenza**: se un oggetto venisse letto sempre dalla cache il client potrebbe visualizzare anche oggetti non aggiornati

42

## Cache - lato client (2)

- Il protocollo HTTP ha un meccanismo per evitare il problema dell'inconsistenza: **conditional GET**

- La richiesta contiene il metodo **GET** ed il campo **If-modified-since**:

### Esempio

- Il client richiede un oggetto non in cache

```
GET /ranaldo/index.htm HTTP/1.0
```

- Il server Web invia un messaggio di risposta con l'oggetto

```
HTTP/1.0 200 OK
```

```
Date: Mon, 10 Apr 2000 10:22:29
```

```
Last-Modified: Thu, 7 Mar 2000 09:23:24
```

```
data data data
```

- Il client salva l'oggetto nella cache locale e la data dell'ultima modifica

43

## Cache - lato client (3)

- Il client richiede lo stesso oggetto che è ancora nella cache; quindi deve verificare che non sia stato modificato sul server

```
GET /ranaldo/index.htm HTTP/1.0
```

```
If-modified-since: Thu, 7 Mar 2000 09:23:24
```

- Se l'oggetto non è stato modificato da quella data, il server invia un messaggio di risposta in cui indica qual è la data in cui l'oggetto è stato modificato l'ultima volta

```
HTTP/1.0 304 Not Modified
```

```
Date: Mon, 10 Apr 2000 10:22:29
```

```
Last-Modified: Thu, 7 Mar 2000 09:23:24
```

```
(entity body vuoto)
```

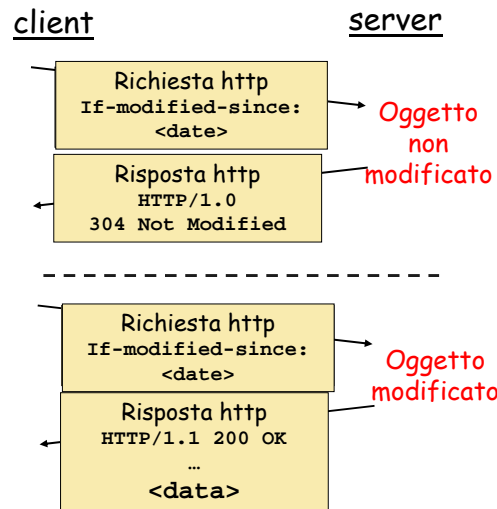
- Il client può utilizzare l'oggetto in cache locale

- Se l'oggetto è stato modificato, allora il server invia l'oggetto modificato

44

## Cache - lato client (4)

- Occorre comunque una interazione, però il secondo messaggio di risposta del server non contiene l'entity body se l'oggetto nella cache locale è valido



45

## Cache - lato rete (1)

- Le cache dal lato rete sono chiamati **cache Web** o **server proxy** o **proxy**
- I server proxy sono computer dotati di una buona capacità di memorizzazione di massa interposti tra il client ed il server
  - Nella memoria di massa contengono gli oggetti richiesti più di recente
- Un client usa un server proxy per evitare di inviare messaggi ai server Web (**server di origine**)
- Per indirizzare un messaggio ad un proxy è necessario configurare opportunamente il client in modo che la richiesta venga inviata prima al proxy

46

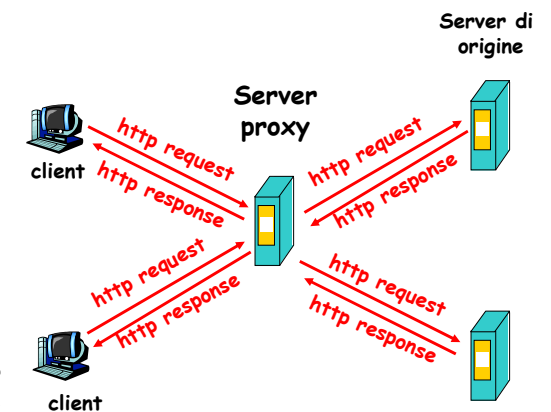
## Cache - lato rete (2)

- Il client stabilisce una connessione con il server proxy ed invia un messaggio di richiesta per un oggetto presente su un server Web
  - La connessione TCP avviene con il server proxy, ma nel messaggio HTTP è contenuto il server di origine
- Se il server proxy ha una copia dell'oggetto, la spedisce al client direttamente, altrimenti invia una richiesta al server Web
- Una volta ricevuta la risposta dal server Web, il proxy la memorizza localmente e ne invia una copia al client sulla connessione che questo aveva inizialmente aperto
- Richieste successive per lo stesso oggetto sono servite più velocemente

47

## Cache - lato rete (3)

- Si noti che il server proxy si comporta sia come client che come server nelle interazioni HTTP



48

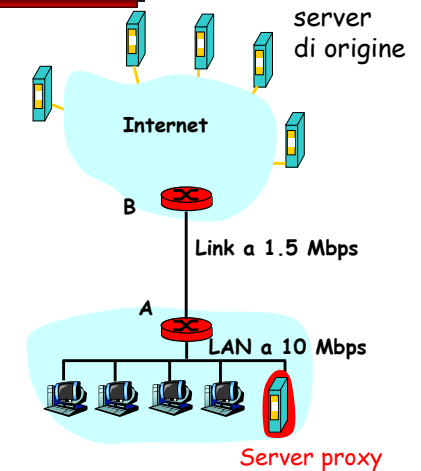
## Cache - lato rete (4)

- La cache lato rete permette di migliorare di molto il tempo di risposta di un server Web
- Il vantaggio è tanto più evidente quanto più è bassa l'ampiezza di banda della rete fra il proxy ed il server Web rispetto a quella tra il client ed il proxy
- Si riduce il traffico sulla rete soprattutto in prossimità del server Web
  - Il traffico si concentra essenzialmente tra client e server proxy
  - Il traffico tra server proxy e server di origine è limitato al mantenimento della consistenza della cache
- Tipicamente un ISP ha un (o più) proprio server proxy installato nella propria rete e può configurare i browser della rete in modo che puntino tutti a quella sede

49

## Esempio (1)

- Una rete locale a 10 Mbps è collegata ad Internet tramite una coppia di router A e B separati da un link a 1.5 Mbps
- Sulla rete locale sono in esecuzione browser Web
- Ad Internet sono connessi dei server Web
- Assunzione (cache rete vicina al client - ad es. stessa rete locale)
- Tempo di risposta minore se si utilizza un server proxy: è "più vicino" al client
  - Diminuisce il traffico verso server lontani
  - Il link di uscita della rete di un ISP istituzionale/locale è spesso un collo di bottiglia



50

## Esempio (2)

- Si supponga che:
  - La dimensione media di un oggetto Web sia di 100 Kb
  - Il tempo necessario ( $d_{Int}$ ) affinché una richiesta Web passi dal router B al server Web e l'oggetto di risposta arrivi allo stesso router sia di 2 s
  - Il numero medio di richieste prodotte dai browser verso il server Web sia di 15 al s
- Si vuole calcolare il **tempo di risposta  $t_r$** 
  - Tempo necessario affinché venga ricevuto dal client l'oggetto richiesto al server Web
  - Esso è legato al ritardo presente sulla LAN ( $d_{LAN}$ ), al ritardo tra i due router A e B ( $d_{AB}$ ) ed al ritardo su Internet  $d_{Int}$
  - Supporremo che  $d_{LAN}$  e  $d_{AB}$  siano i ritardi dovuti alla trasmissione degli oggetti restituiti dai server Web, mentre saranno trascurati i ritardi di trasmissione delle richieste
    - I messaggi di richiesta sono di piccole dimensioni

$$t_r = d_{LAN} + d_{AB} + d_{Int}$$

51

## Esempio (3)

- Di conseguenza  $d_{LAN}$  è il tempo dovuto alla ricezione di un oggetto dal router A fino al client che desidera l'oggetto
- Mentre  $d_{AB}$  è legato al ritardo di trasmissione dell'oggetto Web tra il router B ed il router A e al ritardo di accodamento (trascurando ritardo di propagazione e di elaborazione)
- $d_{AB} = d_{tras} + d_{queue}$
- Per valutare l'entità di  $d_{LAN}$  e  $d_{AB}$  è necessario stimare l'intensità del traffico
  - Per la rete locale;
 
$$it = L/R = (100 \text{ Kb}) \times (15 \text{ richieste /s}) / (10 \text{ Mbps}) = 0,15$$
  - Per il link geografico
 
$$it = L/R = (100 \text{ Kb}) \times (15 \text{ richieste /s}) / (1,5 \text{ Mbps}) = 1$$
- Per la rete locale il ritardo di trasmissione è pari a:
 
$$d_{tras} = L/R = 100 \text{ Kb} / 10 \text{ Mbps} = 100 \times 10^3 / (10 \times 10^6) = 0,01 \text{ s} = 10 \text{ ms}$$

52

## Esempio (4)

- Quindi per la rete locale si può stimare un ritardo di circa 10 ms e un'intensità di traffico di 0,15
  - Se l'intensità del traffico sulla LAN è bassa si può supporre che ci siano poche collisioni e quindi l'ampiezza di banda media è circa uguale a quella a burst, pertanto  $d_{LAN} = 10$  ms
- Per il link geografico il ritardo è molto alto (velocità di trasmissione più bassa ed elevato ritardo di accodamento con  $it=1$ )
- Pertanto il tempo di risposta sarà dell'ordine dei minuti
- Una prima soluzione è quella di impiegare un link a capacità maggiore tra i due router, per esempio a 10 Mbps
  - In tal caso  $it = 0,15$  ed il ritardo  $d_{AB}$  è circa 10 ms (trascurando il ritardo di accodamento)
- In questo caso il ritardo complessivo è dato da
$$t_r = d_{LAN} + d_{AB} + d_{Int} = 10 \text{ ms} + 10 \text{ ms} + 2 \text{ s} = 2,02 \text{ s}$$

53

## Esempio (5)

- Ma una soluzione di questo tipo è molto costosa!
- Una soluzione basata su cache Web è molto più vantaggiosa sia in termini di costo che di prestazioni
- Viene installato un server proxy sulla rete locale
- Le richieste vengono inoltrate al server proxy
- Nel caso di **miss** (richiesta di oggetti non presenti in cache) la richiesta deve essere inviata al server
- Tipicamente la percentuale di successo (**hit rate**) oscilla nel range di 0,2 - 0,7; supponiamo valga 0,4
  - Quindi il 40 % delle richieste sarà soddisfatto dal server proxy con un ritardo pari a  $d_{LAN} = 10$  ms

54

## Esempio (6)

- Il 60 % sarà inoltrato verso Internet
  - $it = La/R = (100 \text{ Kb}) \times (15 \text{ richieste} \times 0,6 / \text{s}) / (1,5 \text{ Mbps}) = 0,6$
  - Con  $it=0,6$  la rete non è molto trafficata e quindi possiamo supporre che il ritardo di accodamento sia di pochi ms quindi trascurabile
  - Il ritardo di trasmissione sul link dal router A al router B è pari a:  $100\text{Kb}/1,5\text{Mbps} = 66,7$  ms
  - Quindi il ritardo complessivo nel caso di miss è dato da:
$$t_{r(\text{miss})} = d_{LAN} + d_{AB} + d_{Int} = 10 \text{ ms} + 66,7 \text{ ms} + 2 \text{ s} = 2,0767 \text{ s}$$
- Il tempo di risposta medio è dato da:
$$t_r = 0,4 \times t_{r(\text{hit})} + 0,6 \times t_{r(\text{miss})} =$$
$$= 0,4 \times (0,010 \text{ s}) + 0,6 \times (2,0767) = 1,25 \text{ s}$$
- Si ottiene quindi un tempo medio di risposta inferiore del tempo di risposta che si otterrebbe se si utilizzasse un link a 10 Mbps!

55