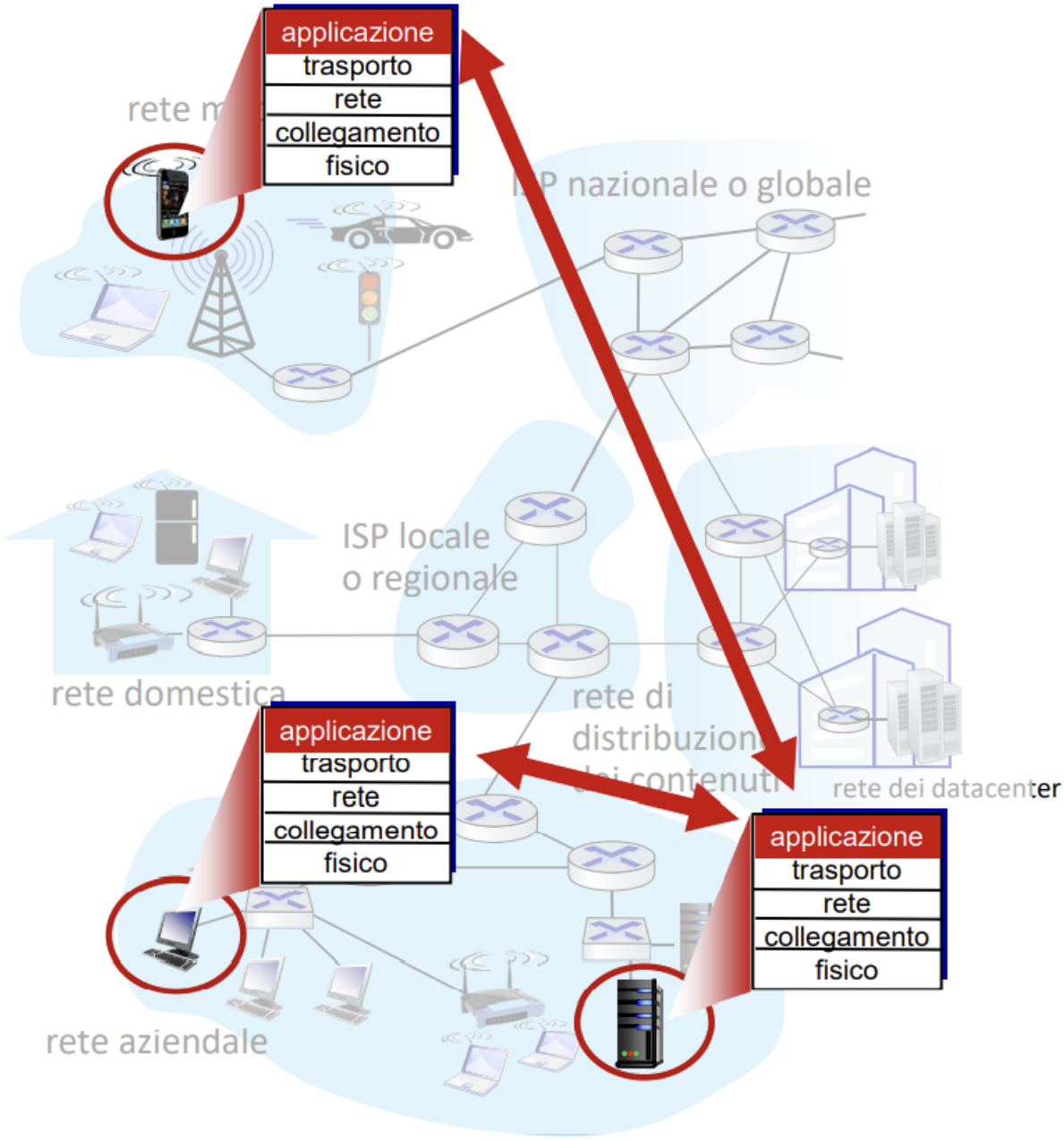
Lezione 4

Creare un'applicazione di rete

- Scrivere programmi che :
  - girano su sistemi terminali differenti
  - comunicano attraverso la rete
  - per esempio, il software di un server web comunica con il software di un browser.
- Non è necessario scrivere programmi per i dispositivi nel nucleo della rete

- i dispositivi del nucleo della rete non eseguono applicazioni utente



Paradigma client-server

- Server :
  - host sempre attivo
  - indirizzo IP fisso
  - spesso in datacenter , per la scalabilità
- Client :
  - contatta , comunica col server

- può contattare il server in qualunque momento
- può avere indirizzi IP dinamici
- *non* comunica direttamente con gli altri client

Spesso , 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 utilizza un **Data Center** che , ospita molti host , creando un potente server virtuale. Può ospitare fino a centinaia di migliaia di server a cui deve essere fornita alimentazione e manutenzione.

Architettura peer-to-peer

- *non* c'è un server sempre attivo
- coppie arbitrarie di host (peer) comunicano direttamente tra loro
- i peer richiedono un servizio ad altri peer e forniscono un servizio in cambio ad altri peer
    - scalabilità intrinseca - nuovi peer aggiungono capacità di servizio al sistema , sebbene generino anche nuovo carico di lavoro.
- i peer non devono necessariamente essere sempre attivi e cambiano indirizzo IP.

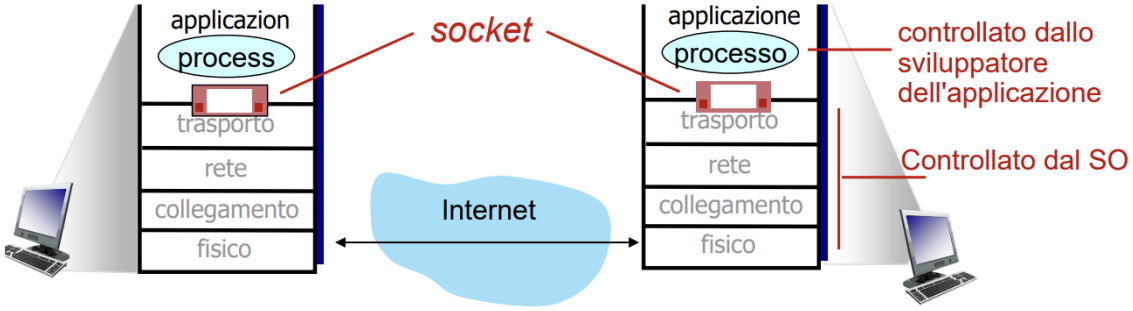
Processi comunicanti

Processo : programma in esecuzione su di un host.

- All'interno dello stesso host , due processi comunicano usando un approccio interprocesso (inter-process-communication).
- Processi su host differenti comunicano attraverso lo scambio di messaggi.

Socket

- Un socket è l'interfaccia tra un processo applicativo e il protocollo a livello applicativo (l'applicazione lato mittente spinge fuori i messaggi tramite la socket , mentre lato ricevente , il protocollo a livello di trasporto ha la responsabilità di consegnare i messaggi alla socket del processo ricevente).
- Un processo invia/riceve messaggi a/da la sua socket
- un socket è analoga a una porta
  - il processo mittente fa uscire il messaggio fuori dalla propria "porta" (socket)
  - il processo mittente presuppone l'esistenza di un'infrastruttura esterna che trasporterà il messaggio attraverso la rete fino alla porta del processo di destinazione.



Indirizzamento

- Per ricevere messaggi , un processo deve avere un identificatore.
  - L'identificatore comprende sia l'indirizzamento IP che i numeri di porta associati al processo in esecuzione su un host
- Un host ha un indirizzo IP univoco a 32 bit.
- *Domanda* : è sufficiente conoscere l'indirizzo IP dell'host su cui è in esecuzione un process per identificare il processo stesso?
  - *Risposta* : no , sullo stesso host possono essere in esecuzione molti processi.
- Numeri di porta assegnati a applicazioni più note :

    - HTTP server : 80
    - Mail server : 25
- Per inviare un messaggio HTTP al server "tizio.edu" :
    - Indirizzo IP : 128.119.245.12
    - numero di porta : 80

Quale servizio di trasporto richiede un'applicazione?

- Perdita di Dati
    - Occorre garantire che i dati inviati siano consegnati corretti e completi.
    - Se un protocollo fornisce questo tipo di servizio, si dice che fornisce un **trasferimento dati affidabile**.
        - alcune applicazioni (come , trasferimento file) richiedono un trasferimento 100% affidabile.
    - Quando un protocollo a livello di trasporto non fornisce il trasferimento dei dati affidabile , i dati potrebbero non arrivare mai a quello ricevente. Questo potrebbe essere accettabile per le **applicazioni che tollerano le perdite**.
        - altre applicazioni (come , audio) possono tollerare qualche perdita.
- Sensibilità al fattore tempo
    - alcune applicazioni (come , telefonia via internet) richiedono che i ritardi siano bassi per essere efficaci.
- Throughput
    - Nel contesto di una sessione di comunicazione tra due processi lungo un percorso in rete , il throughput è la velocità al quale il processo mittente può inviare i bit al processo ricevente.
    - Se il protocollo aa livello di trasporto non può fornire questo throughput , l'applicazione deve codificare i dati a un livello inferiore o rinunciare.

    - Le applicazioni che hanno requisiti di throughput vengono dette **applicazioni sensibili alla banda**.
    - Le applicazioni sensibili alla banda hanno requisiti specifici di throughput , le **applicazioni elastiche**.
    - alcune applicazioni dette "sensibili alla banda" (come , quelle multimediali) per essere efficaci richiedono un'ampiezza di banda minima.
- Sicurezza
    - cifratura , integrità dei dati ecc...

Servizi dei protocolli di trasporto di Internet

Servizio di TCP :

- TCP prevede un servizio orientato alla connessione e il trasporto affidabile dei dati. E offre due servizi :
    1. *Servizio orientato alla connessione :* fa in modo che il client e server si scambino informazioni di controllo a livello di trasporto prima che i messaggi comincino a fluire. Quindi vengono messi in allerta , dopo questa fase esiste una **connessione TCP** tra le socket dei due processi.
    2. *Servizio di trasferimento affidabile :* I processi comunicanti possono contare su TCP per trasportare i dati senza errori e nel giusto ordine.
- Il TCP include anche un meccanismo di controllo della congestione ovvero esegue una "strozzatura" del processo d'invio quando il traffico in rete appare eccessivo e cerca di confinare all'interno della loro porzione di ampiezza di banda le connessioni TCP.
- Trasporto affidabile : fra i processi di invio e di ricezione : dati consegnati senza errori , perdite e nell'ordine di invio.

- Controllo di flusso : il mittente non vuole sovraccaricare il destinatario.
- Controllo della congestione : "strozza" il processo d'invio quando la rete è sovraccarica.
- Orientato alla connessione : è richiesto un setup fra i processi client e server.
- Non offre : temporizzazione , garanzie su un'ampiezza di banda minima, sicurezza.

Servizio di UDP :

- È un protocollo di trasporto leggero , è senza connessione e fornisce un servizio di trasferimento dati non affidabile.
- Quando un processo invia un messaggio tramite la socket UDP , il protocollo non garantisce che questo raggiunga il processo di destinazione e inoltre i messaggi potrebbero giungere a destinazione non in ordine. Non c'è un meccanismo di congestione , pertanto un processo di invio UDP può "spingere" i dati al livello sottostante a qualsiasi velocità.
- Trasferimento dati inaffidabile fra i processi d'invio e di ricezione.
- Non offre : affidabilità , controllo di flusso , controllo di congestione , temporizzazione , ampiezza di banda minima , sicurezza , né setup della connessione.

Rendere sicuro TCP

Socket TCP e UDP :

- nessuna cifratura
- password inviate in chiaro , cioè senza cifratura , attraverso socket attraversano Internet in chiaro.

Transport Layer Security (TLS)

- offre connessioni TCP cifrate
- controllo di integrità dei dati
- autenticazione end-to-end.

TLS implementato a livello applicazione :

- le applicazioni usano librerie TLS , che usano a propria volta TCP.
- testo in chiaro (cleartext) inviato nella "socket" attraversa Internet *cifrato*.

Panoramica su HTTP

HTTP : hypertext transfer protocol

- Protocollo a livello applicazione del Web
- Implementato in due programmi , modello client/server :
  - Client : browser che richiede , riceve e visualizza gli oggetti dal web.
  - Server : il server Web che invia oggetti in risposta alle richieste.
- Definisce sia la struttura dei messaggi sia la modalità con cui client e server si scambiano i messaggi.

HTTP usa TCP :

- il client inizializza la connessione TCP (crea un socket) con il server , sulla porta 80
- il server accetta la connessione TCP dal client
- messaggi HTTP scambiati tra browser e server Web.
- connessione chiusa TCP

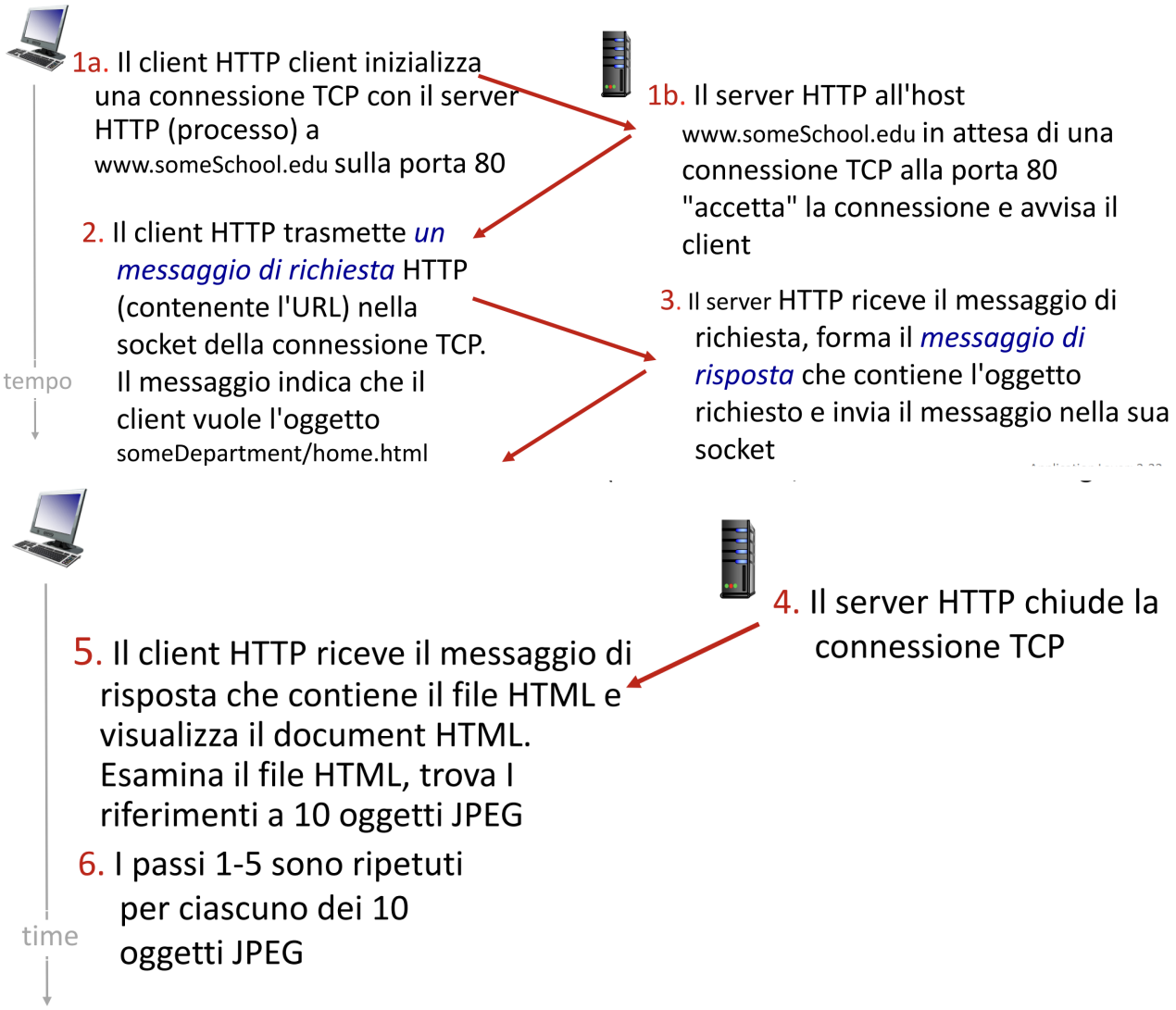
- è un protocollo senza stato
    - il server non mantiene informazioni sulle richieste fatte dal client.

Ci sono due tipi di connessioni :

1. Connessioni non persistenti :
    1. connessione TCP aperta
    2. almeno un oggetto viene trasmesso su una connessione TCP
    3. connessione TCP chiusa
       Lo scaricamento di oggetti multipli richiede connessioni multiple.

Connessioni non persistenti

L'utente immette l'URL: `http://www.someSchool.edu/someDepartment/home.html`
(contiene testo, riferimenti a 10 immagini JPEG)

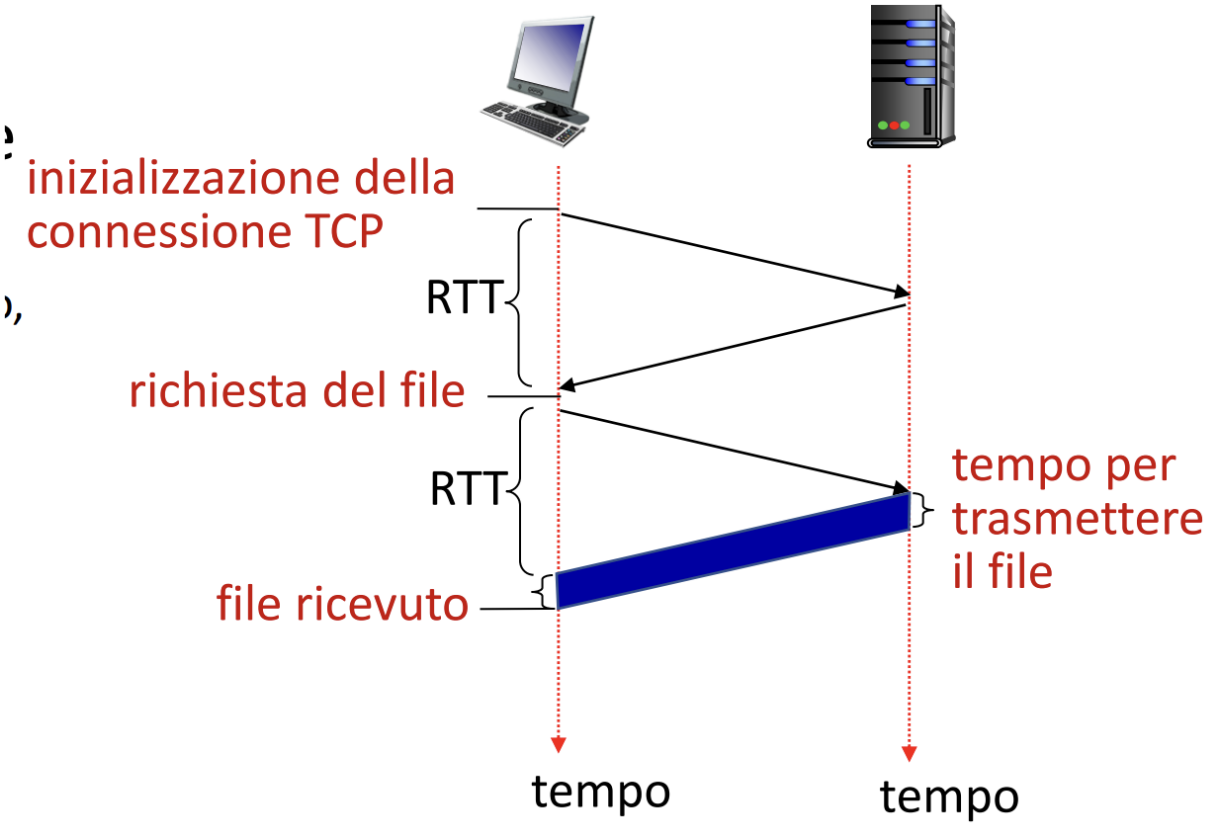


RTT : tempo impiegato da un piccolo pacchetto per andare dal client al server e ritornare al client.

Tempo di risposta :

- un RTT per inizializzare la connessione TCP
- un RTT perché ritornino la richiesta

- tempo di trasmissione del file/oggetto



2. Connessioni persistenti :
    1. connessione TCP connection al server aperta
    2. più oggetti possono essere trasmessi su una *singola* connessione TCP tra client e server
    3. connessione TCP chiusa

Svantaggi delle connessioni non persistenti :

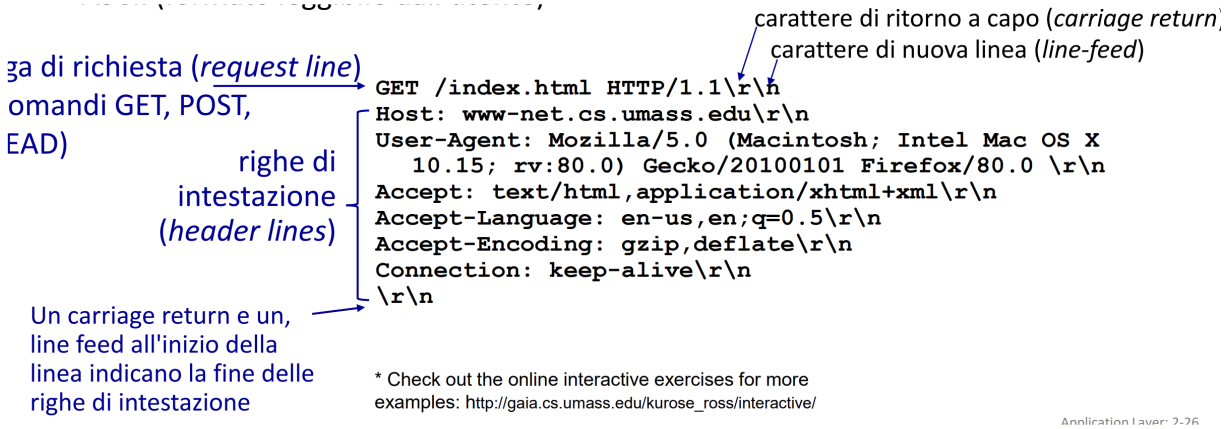
- richiedono 2 RTT per oggetto
- overhead del SO per *ogni* connessione TCP
- i browser spesso aprono connessioni TCP parallele per caricare gli oggetti referenziati

Connessioni persistenti (HTTP 1.1):

- il server lascia la connessione TCP aperta dopo l'invio di una risposta
- i successivi messaggi tra gli stessi client/server vengono trasmessi sulla connessione aperta
- il client invia le richieste non appena incontra un oggetto referenziato
- un solo RTT per tutti gli oggetti referenziati

Messaggio di richiesta HTTP

- Due tipi di messaggi HTTP : *richiesta* , *risposta*.
- Richiesta :



Alcuni esempi di intestazione nei messaggi di richiesta :

- Host :
  - hostname e un numero di porta (se assente si assume 80 per HTTP e 443 per HTTPS) del server al quale sarà inviata la richiesta. Obbligatorio in HTTP/1.1; se assente , il server può rispondere con un 400 Bad Request.
- User-Agent :
  - Identifica l'applicazione , il SO , il *vendor* e/o la versione dello *user agent* che sta effettuando la richiesta.
- Accept :
  - Tipi di contesto , espressi come medita type , compresi dal client.

- Accept-Language :
  - Linguaggi naturali o *locale* preferiti dal client.
- Connection :
  - Controlla se la connessione rimarrà aperta al termine dello scambio richiesta/risposta. Il valore *close* indica che la connessione sarà chiusa (default HTTP/1.0); altrimenti , una lista non vuota di nomi di header , che saranno rimossi dal primo proxy non trasparente o cache , indica che la connessione rimarrà aperta (default HTTP/1.1).

Altri messaggi di richiesta

Metodo POST

- La pagina web spesso include un form per l'input dell'utente
- l'input dell'utente viene inviato dal client al server nel corpo dell'entità di un messaggio di richiesta HTTP POST.

Metodo GET (per inviare dati al server)

- l'input arriva al server nel campo URL della riga di richiesta

Metodo HEAD

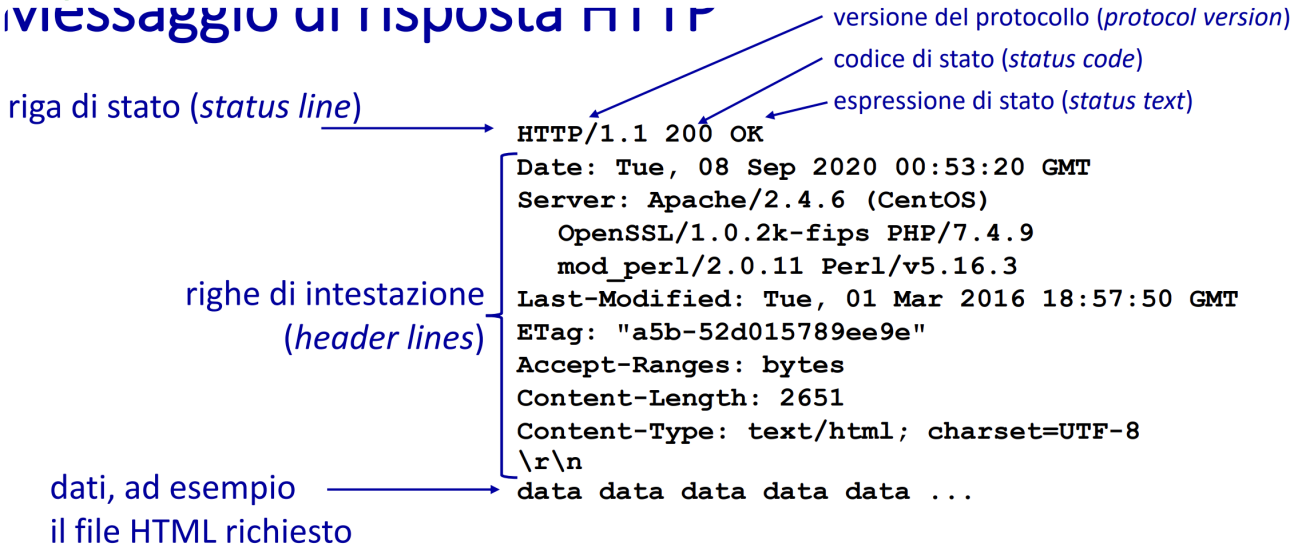
- Richiede le intestazioni (solo) che verrebbero restituite se l'URL specificato fosse richiesto con il metodo HTTP GET.

Metodo PUT

- Carica un nuovo file (oggetto) sul server
- sostituisce completamente il file esistente all'URL specificato con il contenuto del corpo dell'entità del messaggio di richiesta HTTP

PUT.

Risposta :



Alcuni esempi di intestazione nei messaggi di risposta :

- Date :
  - la data e l'ora in cui il messaggio è stato originato.
- Server :
  - descrive il software usato dal server di origine per gestire la richiesta.
- Last-Modified :
  - la data e l'ora in cui il server di origine che l'oggetto sia stato modificato per l'ultima volta.
- Accept-Rangers :
  - indica il supporto del server ai download parziali : il valore , se diverso da *none* , indica l'unità che si può usare per esprimere l'intervallo richiesto.
- Content-Length :
  - lunghezza in byte del corpo dell'entità inviato al ricevente.
- Content-Type :
  - *media type* del corpo dell'entità inviato al ricevente.

Codici di stato della risposta HTTP

- Nella prima riga nel messaggio di risposta dal server al client
- definiti da RFC 9110
- raggruppati in cinque categorie , discriminate dalla prima cifra
  - 1xx Informational : una risposta intermedia per comunicare lo stato di connessione o l'avanzamento della richiesta prima di completare l'azione richiesta e inviare una risposta finale.
  - 2xx Successful : la richiesta è stata ricevuta con successo , compresa e accettata.
  - 3xx Redirect : il client deve eseguire ulteriori azioni per soddisfare la richiesta.
  - 4xx Client Error : la richiesta è sintatticamente scorretta o non può essere soddisfatta.
  - 5xx Server Error : il server ha fallito nel soddisfare una richiesta apparentemente valida.
  - 200 OK : la richiesta ha avuto successo, l'oggetto richiesto viene inviato nella risposta.
  - 301 Moved Permanently : l'oggetto richiesto è stato trasferito , la nuova posizione è specificata nell'intestazione.
  - 400 Bad Request : il messaggio di richiesta non è stato compreso dal server.
  - 404 Not Found : il documento di richiesta non si trova su questo server.
  - 406 Not Acceptable : l'oggetto richiesto non esiste in una forma che soddisfa i cari Accept.
  - 505 HTTP Version Not Supported : il server non ha la versione di protocollo HTTP.