Università degli Studi di Roma "Tor Vergata"
Laurea in Informatica

Sistemi Operativi e Reti
(modulo Reti)
a.a. 2023/2024

Livello di applicazione

dr. Manuel Fiorelli
manuel.fiorelli@uniroma2.it
https://art.uniroma2.it/fiorelli

Basate sulle slide del libro di testo:
https://gaia.cs.umass.edu/kurose_ross/ppt.php

Livello di applicazione: panoramica

- Principi delle applicazioni di rete
- Web e HTTP
- E-mail, SMTP, IMAP
- DNS: il servizio di directory di Internet
- Applicazioni P2P
- Streaming video e reti di distribuzione di contenti
- Programmazione delle socket programming con UDP e TCP

Livello di applicazione: panoramica

Obiettivi:

- Aspetti concettuali e implementativi dei protocolli delle applicazioni di rete
  - modelli di servizio del livello di trasporto
  - paradigma client-server
  - paradigma peer-to-peer
- apprendere informazioni sui protocolli esaminando i protocolli e l'infrastruttura delle più diffuse applicazioni di rete
  - HTTP
  - SMTP, IMAP
  - DNS
  - Sistemi per lo streaming video, CDNs
- programmare applicazioni di rete
  - socket API

Alcune diffuse applicazioni di rete

- posta elettronica (e-mail)
- trasferimento file
- Newsgroup (es. Usenet) e poi forum
- Web
- messaggistica istantanea
- social media
- giochi multiutente via rete
- streaming di video-clip memorizzati (YouTube, Hulu, Netflix)
- condivisione di file P2P
- Applicazioni basate sulla posizione
- Applicazioni di pagamento mobile
- telefonia via Internet, voice-over-IP (VoIP), es. Skype
- videoconferenza in tempo reale
- ricerca sul Web
- shell o desktop remoto
- …

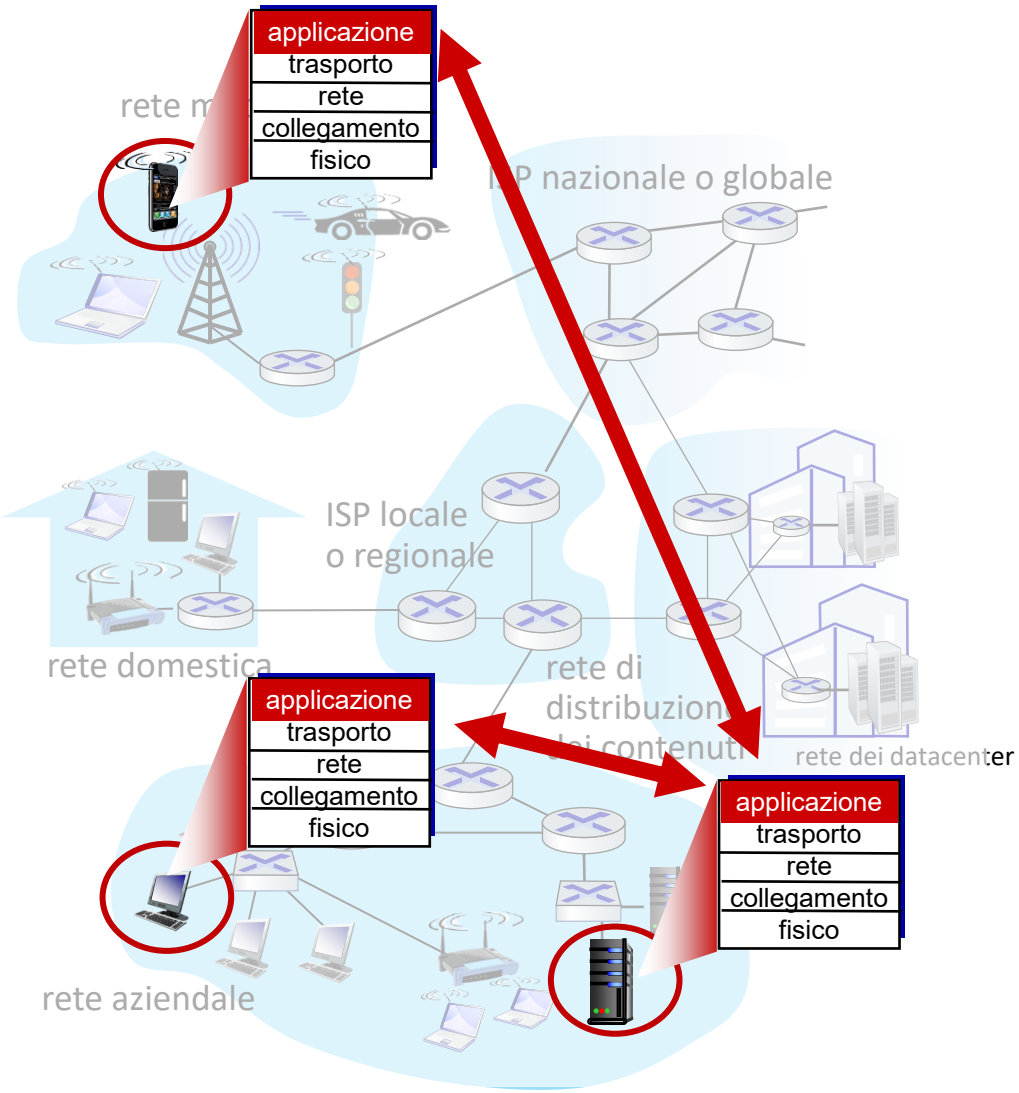
Creare un'applicazione di rete

scrivere programmi che:

- girano su sistemi terminali differenti
- comunicano attraverso la rete
- ad 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
- il confinamento delle applicazioni nei sistemi periferici ha facilitato il rapido sviluppo e la diffusione di una vasta gamma di applicazioni per Internet



Paradigma client-server

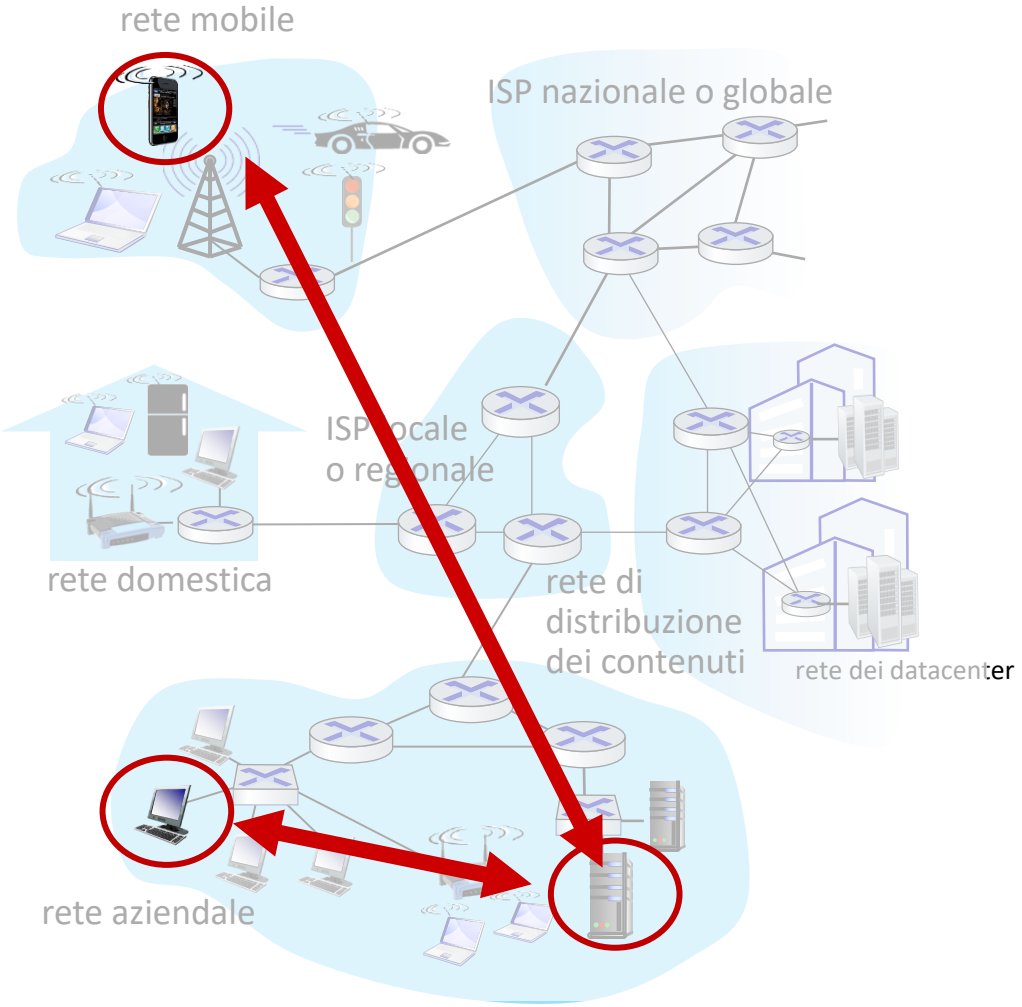
server:

- host sempre attivo
- indirizzo IP fisso
- spesso in datacenter, per la scalabilità

client:

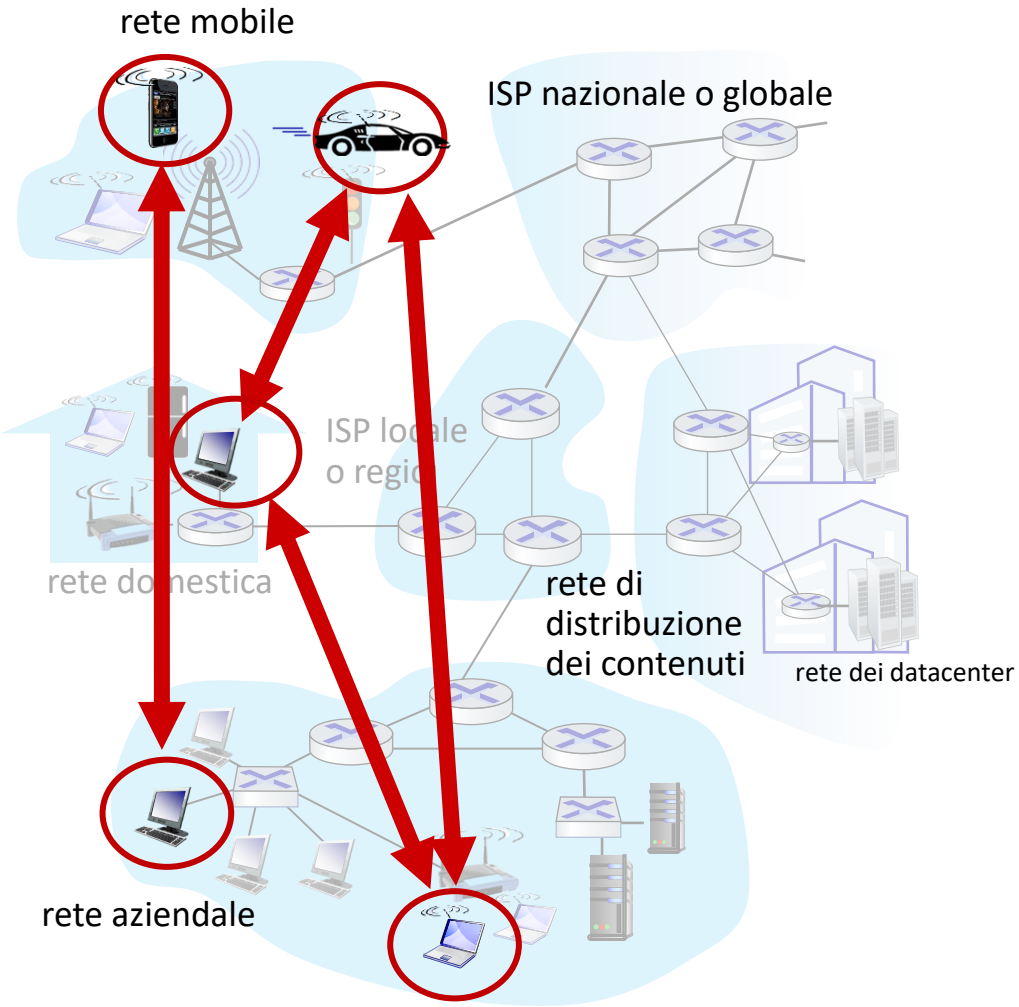
- contatta, comunica col server
- può contattare il server in qualunque momento
- può avere indirizzi IP dinamici
- *non* comunica direttamente con gli altri client

- esempi: Web, posta elettronica, etc.



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
  - Difficile da gestire
- esempio: condivisione di file P2P [BitTorrent]



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) (definito dal sistema operativo)
- processi su host differenti comunicano attraverso lo scambio di messaggi

client, server

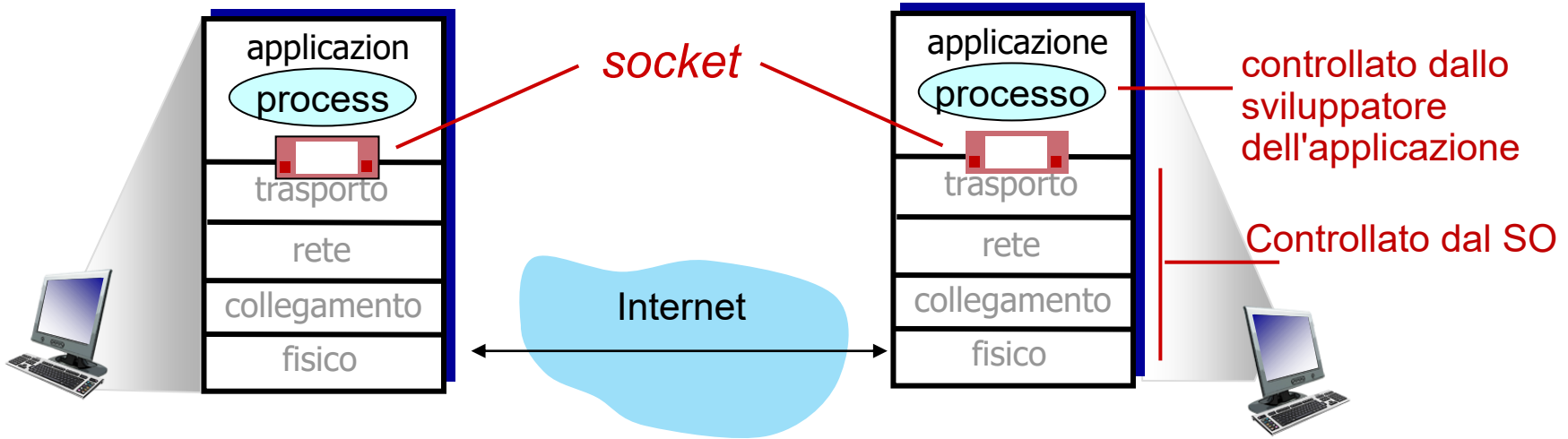
processo client: processo che dà inizio alla comunicazione

processo server: processo che attende di essere contattato

- nota: nelle applicazioni P2P, un processo può essere sia client sia server; cionondimeno, nel contesto di una specifica sessione di comunicazione, ogni processo adotterà l'uno o l'altro ruolo

Socket

- un processo invia/riceve messaggi a/da la sua socket
- una 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 traporterà il messaggio attraverso la rete fino alla "porta" del processo di destinazione



Indirizzamento

- per ricevere messaggi, un processo deve avere un *identificatore*
- Un host ha un indirizzo IP univoco a 32 bit
- *D:* è sufficiente conoscere l'indirizzo IP dell'host su cui è in esecuzione un process per identificare il processo stesso?
- *R:* no, sullo stesso host possono essere in esecuzione *molti* processi.

- *L'identificatore* comprende sia l'indirizzo IP (IP address) che i numeri di porta (port number) associati al processo in esecuzione su un host.
- Numeri di porta assegnati (dallo IANA) a applicazioni più note:
  - HTTP server: 80
  - mail server: 25
- Per inviare un messaggio HTTP al server gaia.cs.umass.edu :
  - indirizzo IP: 128.119.245.12
  - numero di porta: 80

Un protocollo a livello applicazione definisce

- tipi di messaggi scambiati,
  - es, richiesta, risposta
- sintassi dei messaggi:
  - quali sono i campi nel messaggio e come sono descritti
- semantica dei messaggi
  - significato delle informazioni nei campi
- regole per determinare quando e come un processo invia e risponde ai messaggi

protocolli di pubblico dominio:

- definiti nelle RFC, ognuno ha accesso alla definizione
- consentono l'interoperabilità
- ad esempio, HTTP, SMTP

protocolli proprietari:

- ad esempio, Skype, Zoom

Quale servizio di trasposto richiede un'applicazione?

perdita di dati

- alcune applicazioni (ad esempio, trasferimento file, transazioni web) richiedono un trasferimento 100% affidabile
- altre applicazioni(ad esempio, audio) possono tollerare qualche perdita

sensibilità al fattore tempo

- alcune applicazioni (ad esempio, telefonia via Internet, giochi interattivi) richiedono che i ritardi siano bassi per essere “efficaci”

throughput

- alcune applicazioni (dette "sensibili alla banda"), ad esempio, quelle multimediali, per essere efficaci richiedono un'ampiezza di banda minima
- altre applicazioni (le “applicazioni elastiche”) utilizzano l'ampiezza di banda che si rende disponibile

sicurezza

- cifratura, integrità dei dati, …

Requisiti del servizio di trasporto di alcune applicazioni comuni

applicazione	tolleranza alla perdita di dati	throughput	sensibilità al fattore tempo
trasferimento file	no	variabile	no
posta elettronica	no	variabile	no
documenti Web	no	variabile	no
audio/video in tempo reale	sì	audio: 5kbps-1Mbps video:10kbps-5Mbps	sì, centinaia di ms
streaming audio/video memorizzati	Sì	come sopra	sì, pochi secondi
giochi interattivi	sì	fino a pochi kbps	sì, centinaia di ms
messaggistica istantanea	no	variabile	sì e no

Servizi dei protocolli di trasporto di Internet

Servizio di 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:

- *trasferimento dai inaffidabile* fra i processi d'invio e di ricezione
- *non offre:* affidabilità, controllo di flusso, controllo della congestione, temporizzazione, ampiezza di banda minima, sicurezza, né setup della connessione.

D: perché preoccuparsi?
Perché c'è UDP?

Applicazioni Internet: protocollo a livello applicazione e protocollo di trasporto

applicazione	protocollo a livello applicazione	Protocollo di trasporto sottostante
trasferimento file	FTP [RFC 959]	TCP
posta elettronica	SMTP [RFC 5321]	TCP
documenti web	HTTP [RFC 7230, 9110]	TCP
telefonia via Internet	SIP [RFC 3261], RTP [RFC 3550], o proprietario	TCP o UDP
streaming audio/video	HTTP [RFC 7230], DASH	TCP
giochi interattivi	WOW, FPS (proprietario)	UDP o TCP

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 Internet *cifrato*

Livello applicazione: panoramica

- Principi delle applicazioni di rete
- Web e HTTP
- E-mail, SMTP, IMAP
- DNS: il servizio di directory di Internet
- Applicazioni P2P
- Streaming video e reti di distribuzione di contenti
- Programmazione delle socket programming con UDP e TCP

Web e HTTP

Ripassiamo la terminologia…

- una pagina web è costituita da *oggetti,* ciascuno dei quali può essere memorizzato su un server Web differente
- Un oggetto può essere un file HTML, un'immagine JPEG, uno script JavaScript, un foglio di stile CSS, un file audio, …
- una pagina web è formata da *un file HTML di base* che include *diversi oggetti referenziati, ciascuno* referenziato da un *URL*, ad esempio,

`www.someschool.edu/someDept/pic.gif`

nome dell'host (*host name*) | percorso (*path name*)

Panoramica su HTTP

HTTP: hypertext transfer protocol

- protocollo a livello applicazione del Web
- modello client/server:
  - *client:* browser che richiede, riceve (usando il protocollo HTTP) e "visualizza" gli oggetti del Web
  - *server:* il server Web che invia (usando il protocollo HTTP) oggetti in risposta alle richieste

Panoramica su HTTP (continua)

HTTP usa TCP:

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

HTTP è un protocollo "senza stato" (stateless)

- Il server non mantiene informazioni sulle richieste fatte dal client

nota

i protocolli che mantengono lo stato sono complessi!

- la storia passato (stato) deve essere memorizzata
- se il server e/o il client si bloccano, le loro viste dello "stato" potrebbero essere contrastanti e dovrebbero essere riconciliate

Connessioni HTTP: due tipi

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 persistenti

- connessione TCP connection al server aperta
- più oggetti possono essere trasmessi su una *singola* connessione TCP tra client e server
- connessione TCP chiusa

Connessioni non persistenti

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

1a. Il client HTTP client inizializza una connessione TCP con il server HTTP (processo) a www.someSchool.edu sulla porta 80

1b. Il server HTTP all'host www.someSchool.edu in attesa di una connessione TCP alla porta 80 "accetta" la connessione e avvisa il client

2. Il client HTTP trasmette *un messaggio di richiesta* HTTP (contenente l'URL) nella socket della connessione TCP. Il messaggio indica che il client vuole l'oggetto someDepartment/home.html

3. Il server HTTP riceve il messaggio di richiesta, forma il *messaggio di risposta* che contiene l'oggetto richiesto e invia il messaggio nella sua socket

tempo

Connessioni non persistenti (continua)

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

4. Il server HTTP chiude la connessione TCP

5. Il client HTTP riceve il messaggio di risposta che contiene il file HTML e visualizza il document HTML. Esamina il file HTML, trova I riferimenti a 10 oggetti JPEG

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

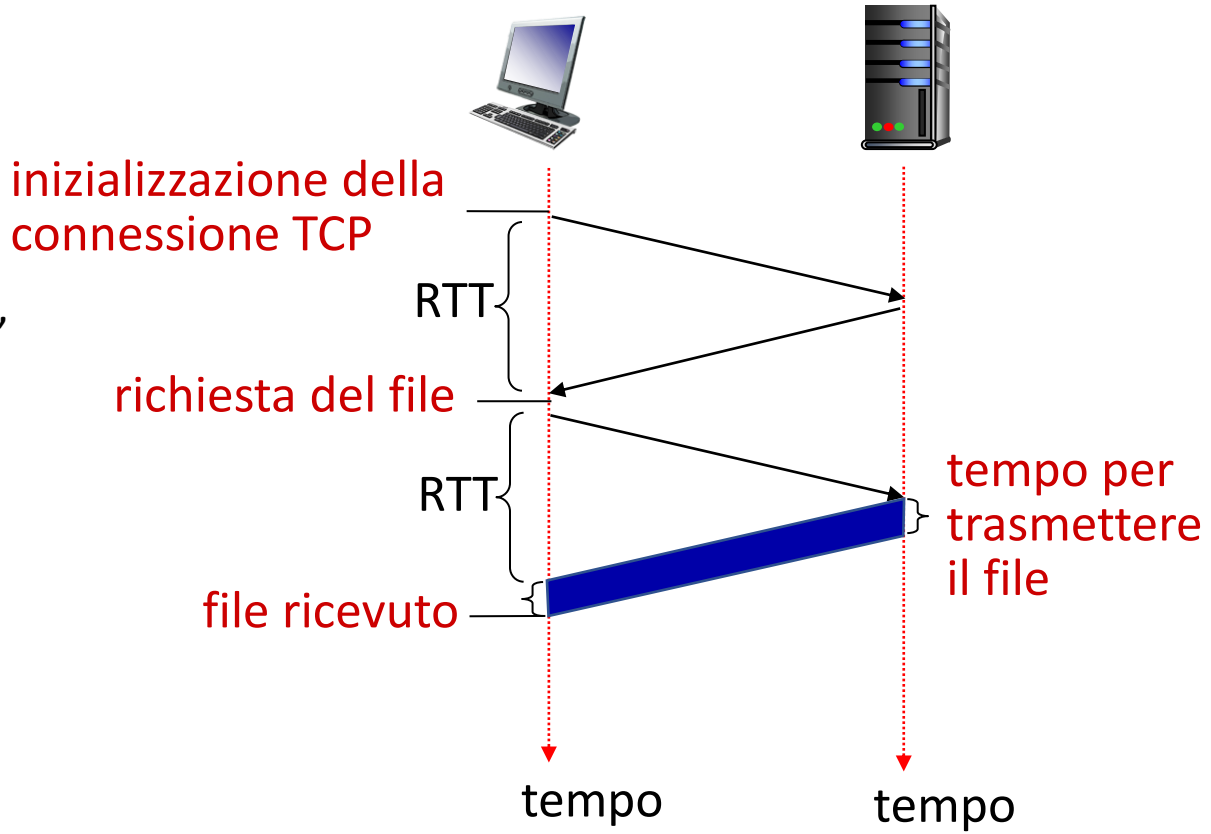
time

Connessioni non persistenti: tempo di risposta

RTT (definizione): tempo impiegato da un piccolo pacchetto per andare dal client al server e ritornare al client (include ritardi di elaborazione, accodamento, propagazione)

Tempo di risposta (per oggetto):

- un RTT per inizializzare la connessione TCP
- un RTT perché ritornino la richiesta HTTP e i primi byte della risposta HTTP
- tempo di trasmissione del file/oggetto



Tempo di risposta con connessioni non persistenti = 2RTT+ tempo di trasmissione del file

Connessioni persistenti (HTTP 1.1)

Svantaggi delle connessioni non persistenti:

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

Connessioni persistenti (HTTP1.1):

- 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*
- Messaggio di richiesta HTTP:
  - ASCII (formato leggibile dall'utente)

carattere di ritorno a capo (*carriage return*)

carattere di nuova linea (*line-feed*)

riga di richiesta (*request line*) (comandi GET, POST, HEAD)

righe di intestazione (*header lines*)

```
GET /index.html HTTP/1.1\r\n
Host: www-net.cs.umass.edu\r\n
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X
   10.15; rv:80.0) Gecko/20100101 Firefox/80.0 \r\n
Accept: text/html,application/xhtml+xml\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Encoding: gzip,deflate\r\n
Connection: keep-alive\r\n
\r\n
```

Un carriage return e un, line feed all'inizio della linea indicano la fine delle righe di intestazione

* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

Messaggio di richiesta HTTP (continua)

Alcuni campi di intestazione nei messaggi di richiesta:

Host
- hostname e 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 sistema operativo, il *vendor* e/o la versione dello *user agent* che sta effettuando la richiesta

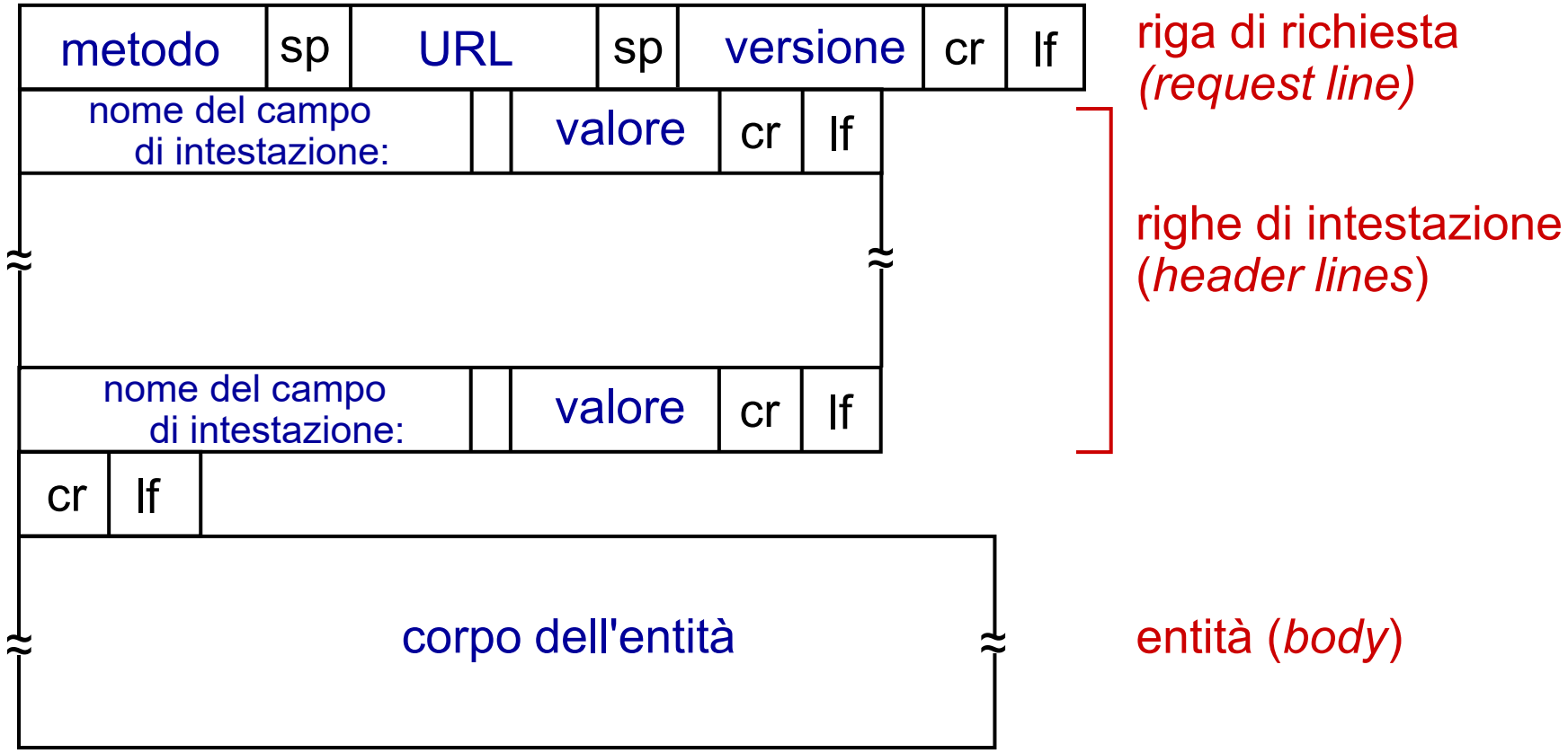
Accept
- Tipi di contento, espressi come medita type, compresi dal client

Accept-Language
- Linguaggi naturali o *locale* preferiti dal client

Accept-Encoding
- Algoritmi di codifica (tipicamente, la compressione) compresi 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 in HTTP/1.0); altrimenti, una lista non vuota di nomi di header (in genere solo keep-alive), che saranno rimossi dal primo proxy non trasparente o cache, indica che la connessione rimarrà aperta (default in HTTP/1.1)

Messaggio di richiesta HTTP: formato generale



Altri messaggi di richiesta HTTP

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 (dopo un '?'):

```
www.somesite.com/animalsearch?monkeys&banana
```

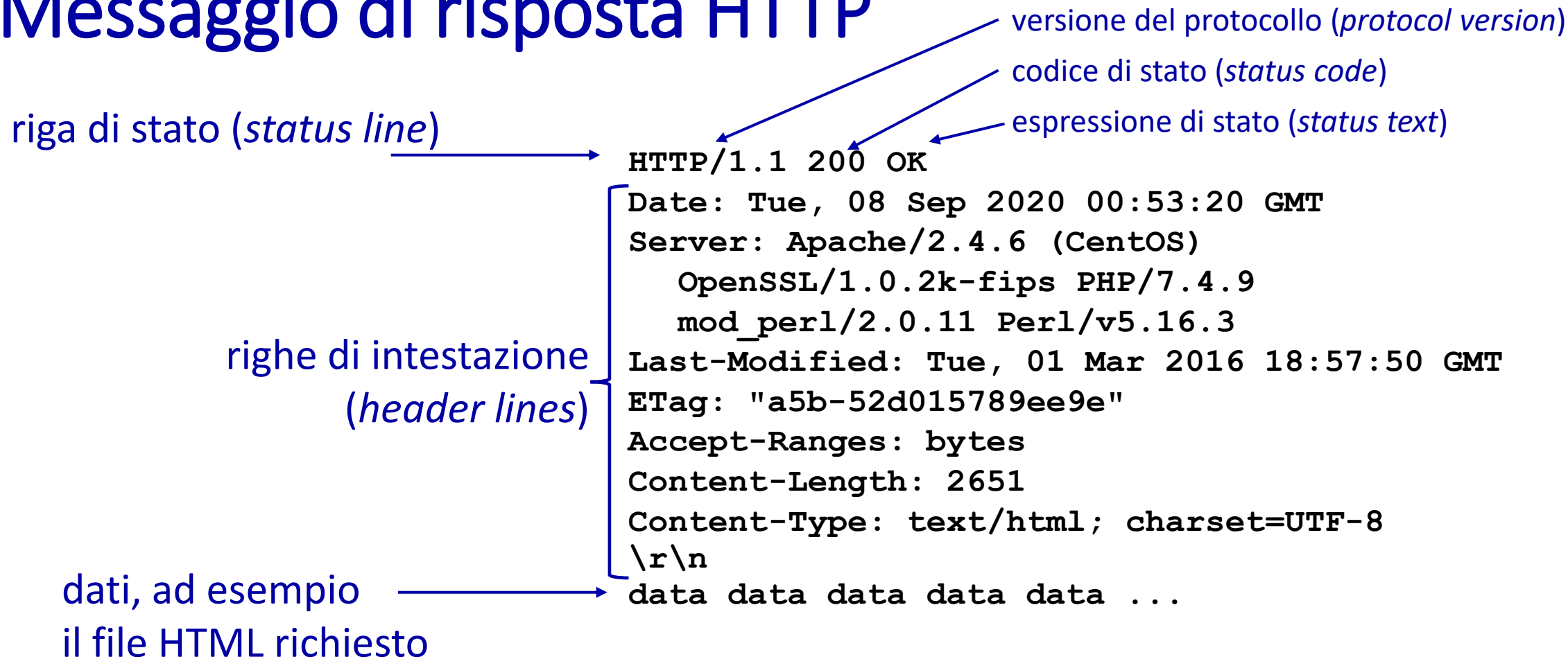
Metodo HEAD:

- 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.

Messaggio di risposta HTTP



* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

Messaggio di risposta HTTP (continua)

Alcuni campi di intestazione nei messaggi di richiesta:

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 (nota: troppi dettagli posso aiutare i malintenzionati a attaccare il server)

Last-Modified
- la data e l'ora in cui il server di origine crede che l'oggetto sia stato modificato per l'ultima volta

Accept-Ranges
- 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 (o che *sarebbe* stato inviato nel caso di un richiesta HEAD)

Content-Type
- *media type* (che indica un formato) del corpo dell'entità inviato al ricevente (o che *sarebbe* stato inviato nel caso di un richiesta HEAD)

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 (assenti in HTTP/1.0)

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

Codici di stato della risposta HTTP

- Nella prima riga nel messaggio di risposta dal server al client
- Alcuni codici di stato e relative espressioni

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 Location: della risposta

400 Bad Request
- Il messaggio di richiesta non è stato compreso dal server

404 Not Found
- Il documento richiesto non si trova su questo server

406 Not Acceptable
- L'oggetto richiesto non esiste in una forma che soddisfa i vari Accept-*

505 HTTP Version Not Supported
- Il server non ha la versione di protocollo HTTP

Provate HTTP (lato client)

1. Collegatevi via netcat al vostro server web preferito:

   % nc -c -v gaia.cs.umass.edu 80 (for Mac)

   >ncat –C gaia.cs.umass.edu 80 (for Windows)

   - apre una connessione TCP alla porta 80 (porta di default per un server HTTP) dell'host gaia.cs.umass.edu.
   - tutto ciò che digitate viene trasmesso alla porta 80 di gaia.cs.umass.edu

2. type in a GET HTTP request:

```
GET /kurose_ross/interactive/index.php HTTP/1.1
Host: gaia.cs.umass.edu
```

   - digitando questo (premete due volte il tasto Invio),trasmettete una richiesta GET minima (ma completa) al server HTTP

3. guardate il messaggio di risposta trasmesso dal server HTTP!

   (oppure usate Wireshark per guardare la richiesta e la risposta HTTP catturate)