Reti di Calcolatori

Seconda lezione

Realizzato da Roberto Savino:

Introduzione

Obiettivi

- Acquisire alcuni concetti di base sul livello applicazione

Sommario:

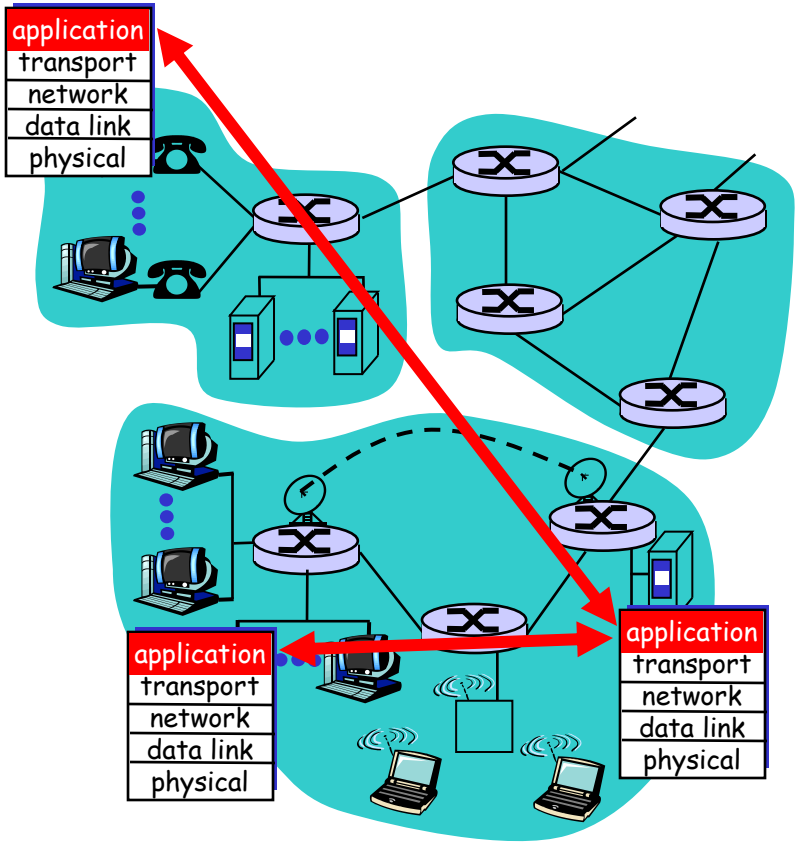
- Servizi forniti dal livello trasporto
- Programmare i Socket in Java
- Studio di un caso: SMTP

Come creare un applicazione di rete

Scrivere programmi che

- Girano su piattaforme differenti
- Comunicano attraverso la rete
- es., Web: Un software server comunica con i browser (clients)

I router intermedi fanno solo da tramite e ignorano il reale contenuto dei pacchetti

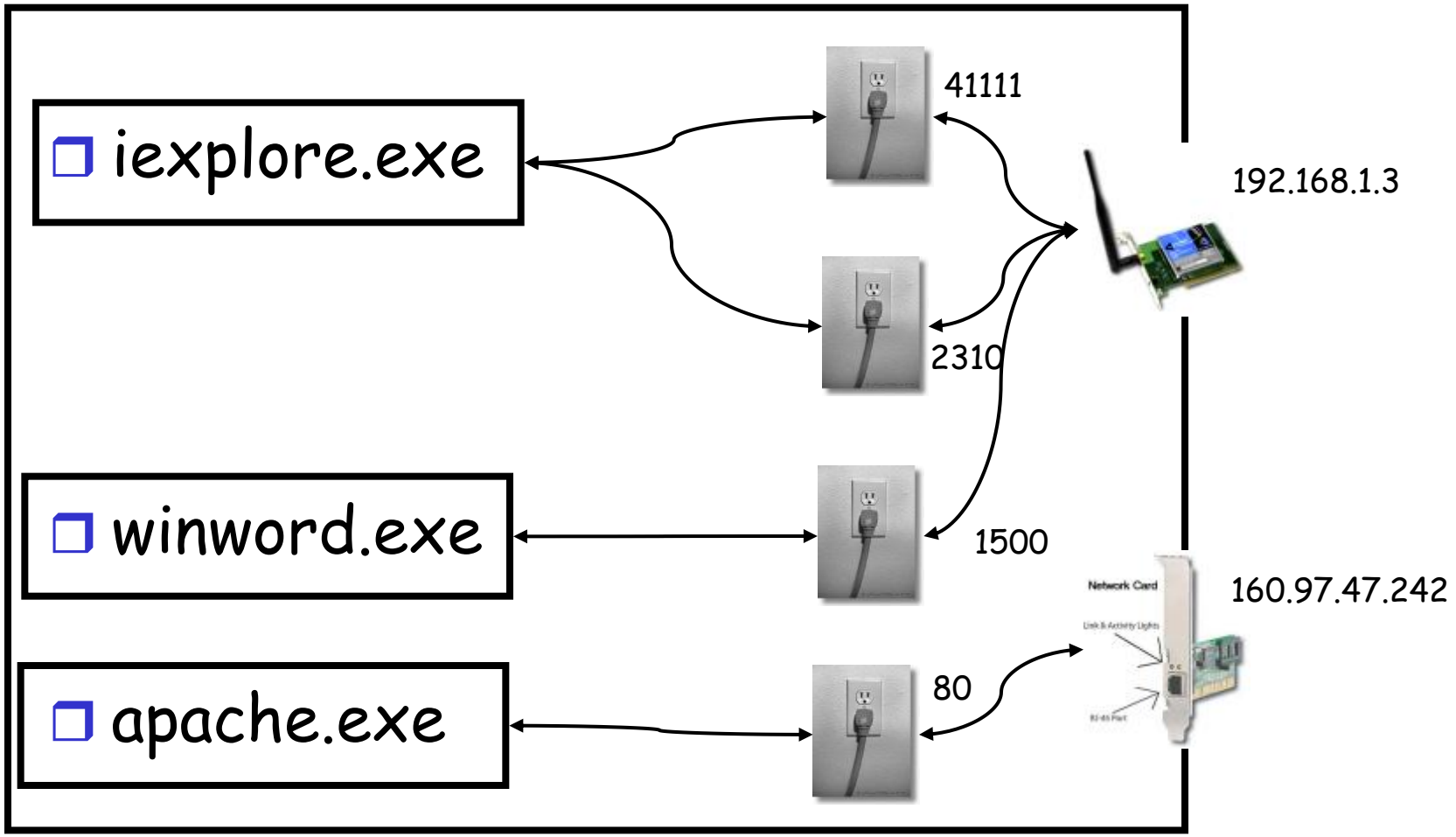


Le regole di internet

- Ogni host possiede più **interfacce** di rete
- Ogni interfaccia è associata a un
  - un indirizzo IP nella forma 160.97.47.24 (quattro byte separati da punti)

- Ogni interfaccia possiede 65535 "porte" di ingresso
- Spesso ad un indirizzo IP è associato un nome (es. www.mat.unical.it <-> 160.97.47.1) ma non necessariamente

Scenario


iexplore.exe
winword.exe
apache.exe
41111
2310
1500
80
192.168.1.3
160.97.47.242
Network Card

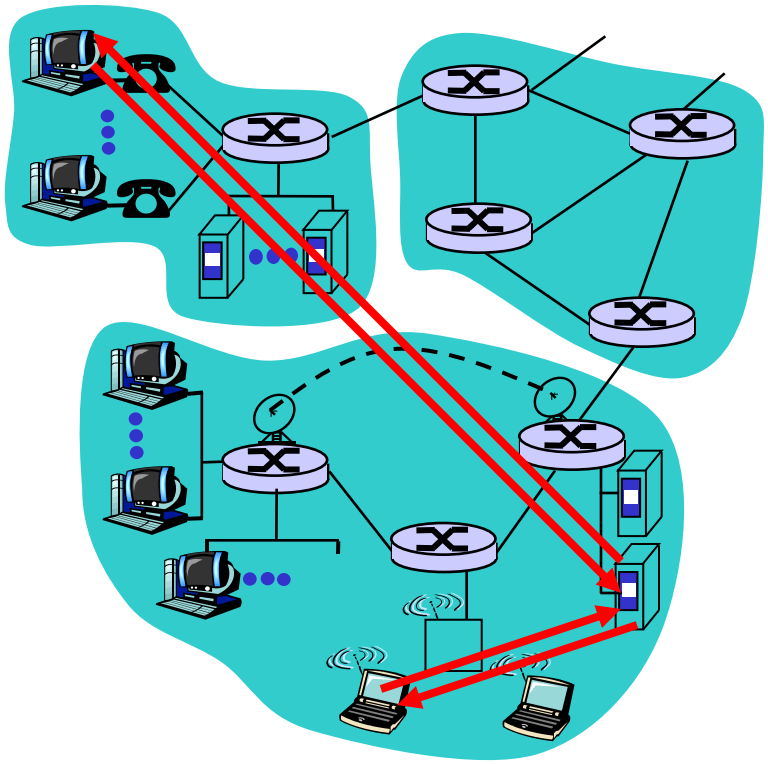
Alcune applicazioni di rete

- E-mail
- Web
- Instant messaging
- Remote login
- P2P file sharing
- Multi-user network games
- Streaming stored video clips
- Internet telephone
- Real-time video conference
- Massive parallel computing

Architetture

- Client-server
- Peer-to-peer (P2P)
- Ibrida (P2P + Client/Server)

Architettura Client/Server



server:

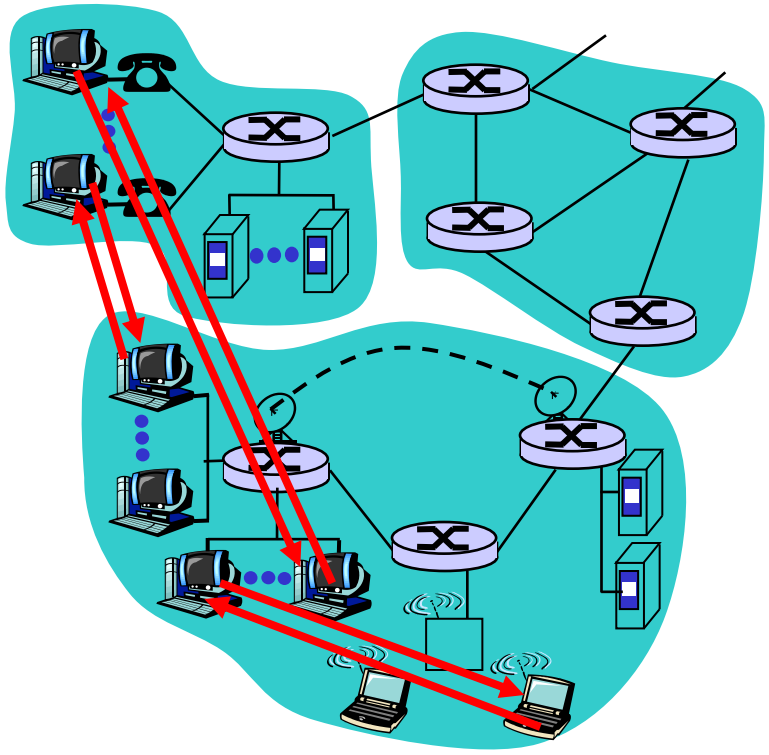
- Sempre acceso
- Indirizzo IP fissato
- server farms for scaling

clients:

- comunicano con il server
- si possono connettere a intermittenza
- Possono avere indirizzi IP dinamici
- Non comunicano tra loro direttamente

Pure P2P architecture

- non serve necessariamente un server
- i sistemi comunicano direttamente tra loro
- i peer possono addirittura cambiare indirizzo IP
- Esempio: la rete Gnutella



Molto scalabile

Tantissimi problemi tecnici da risolvere (non tutti ancora risolti)

Architetture ibride

Napster

- Il trasferimento è P2P
- La ricerca dei file era centralizzata, invece:
  - I peer registravano i nomi dei file su un server centrale
  - I peer interrogavano il server centrale

Messaggeria istantanea

- La chat avviene in modalità P2P
- Il rilevamento della presenza on-line è centralizzato:
  - Quando si va on-line viene registrato l'IP/porta attuale su un server centrale
  - Gli utenti contattano il server centrale per sapere chi è on-line e su che indirizzo/porta è reperibile.

La comunicazione avviene tra processi!!

- All'interno dello stesso host i processi possono comunicare con tecniche varie (IPC)
- Se i processi sono su diversi host, devono usare un sistema a **messaggi**

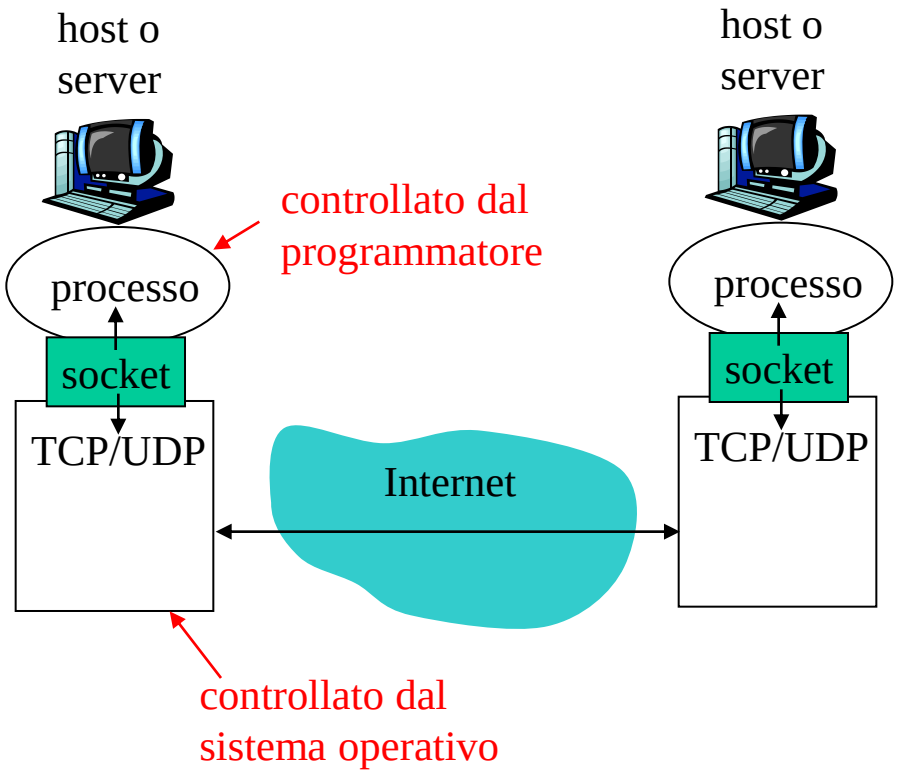
Processo Client: E' il processo che inizia la comunicazione (dice "ou!")

Processo Server: processo che aspetta per essere contattato

- N.B.: Le applicazioni P2P hanno sia thread client che thread server.

I Socket

- Un processo invia e riceve messaggi attraverso i suoi socket
- Il socket è simile a un ingresso/uscita verso il mondo esterno
  - L'invio si affida all'infrastruttura offerta dallo strato di trasporto



- Funzioni di libreria Java: (1) Scelta del protocollo tra TCP o UDP; (2) Possibilità di scegliere solo pochi parametri

Indirizzamento

- Ogni processo deve potere identificare i propri socket
- Non basta l'IP dell'interfaccia (anche se magari è unica)
- Il socket è identificato da una coppia IP:PORTA
- Esempio: 160.97.47.1:80

Cosa definiscono i protocolli Applicazione

- Il tipo di messaggi scambiati
- La sintassi: che informazioni scambiarsi e in che formato
- La semantica dei messaggi: cosa significa ciascun campo?
- Le regole su come e quando certi messaggi devono essere scambiati

Protocolli di pubblico dominio:

- definiti nelle RFC
- es., HTTP, SMTP

Protocolli proprietari:

- es., KaZaA

Esigenze di un protocollo applicazione

Affidabilità

- Fatte salvo le applicazioni multimediali le altre applicazioni non tollerano che si perdano dei dati

Banda

- Ci sono applicazioni che si "arrangiano" con la banda che trovano, altre molto esigenti

Tempo di risposta

- A seconda dell'applicazione, il ritardo può essere un problema

Esigenze delle applicazioni più comuni

Applicazione	Affidabilità	Banda	Latenza
file transfer	no loss	elastic	no
e-mail	no loss	elastic	no
Web documents	no loss	elastic	no
real-time audio/video	loss-tolerant	audio: 5kbps-1Mbps video:10kbps-5Mbps	yes, 100's msec
stored audio/video	loss-tolerant	same as above	yes, few secs
interactive games	loss-tolerant	few kbps up	yes, 100's msec
instant messaging	no loss	elastic	yes and no

C'è un altro parametro che è molto importante: il JITTER

Servizi a disposizione

Servizio TCP:

- *Con connessione:* e' necessaria una fase di set up per aprire una connessione
- *Affidabile:* il programmatore può assumere (+ o -) che il canale sia affidabile
- *Controllo di flusso e congestione:* i pacchetti arrivano in ordine e non bisogna pensare al fatto che si producono troppi dati, basta inviare
- *Cosa non c'è:* nessuna garanzia su latenza e banda

Servizio UDP:

- Trasferimento non affidabile tra mittende e destinatario
- Non fornisce: connessione, affidabilità, controllo di flusso e congestione, garanzie di latenza o banda, sequenza di arrivo

Q: Perchè UDP allora?

Alcuni protocolli standard

Application	Application layer protocol	Underlying transport protocol
e-mail	SMTP [RFC 2821]	TCP
remote terminal access	Telnet [RFC 854], SSH	TCP
Web	HTTP [RFC 2616]	TCP
file transfer	FTP [RFC 959]	TCP
streaming multimedia	RTP [RFC 1889] proprietary	TCP or UDP
Internet telephony	(es., Vonage,Dialpad Skype)	typically UDP

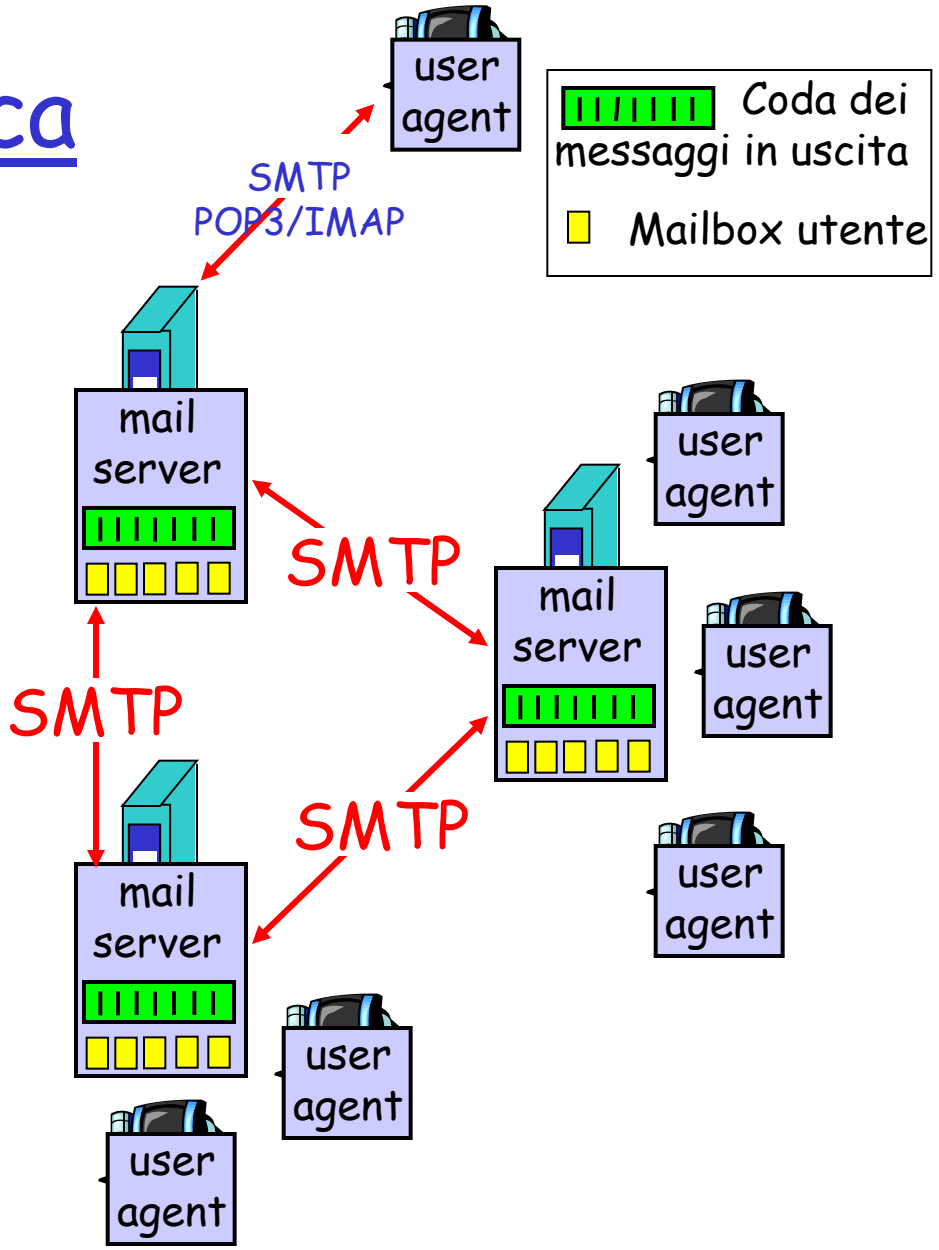
La posta elettronica

Tre componenti:

- Client di posta (Outlook, Eudora, ecc.)
- Server di posta
- Il Simple Mail Transfer Protocol: SMTP

- Protocolli per la lettura: IMAP, POP3

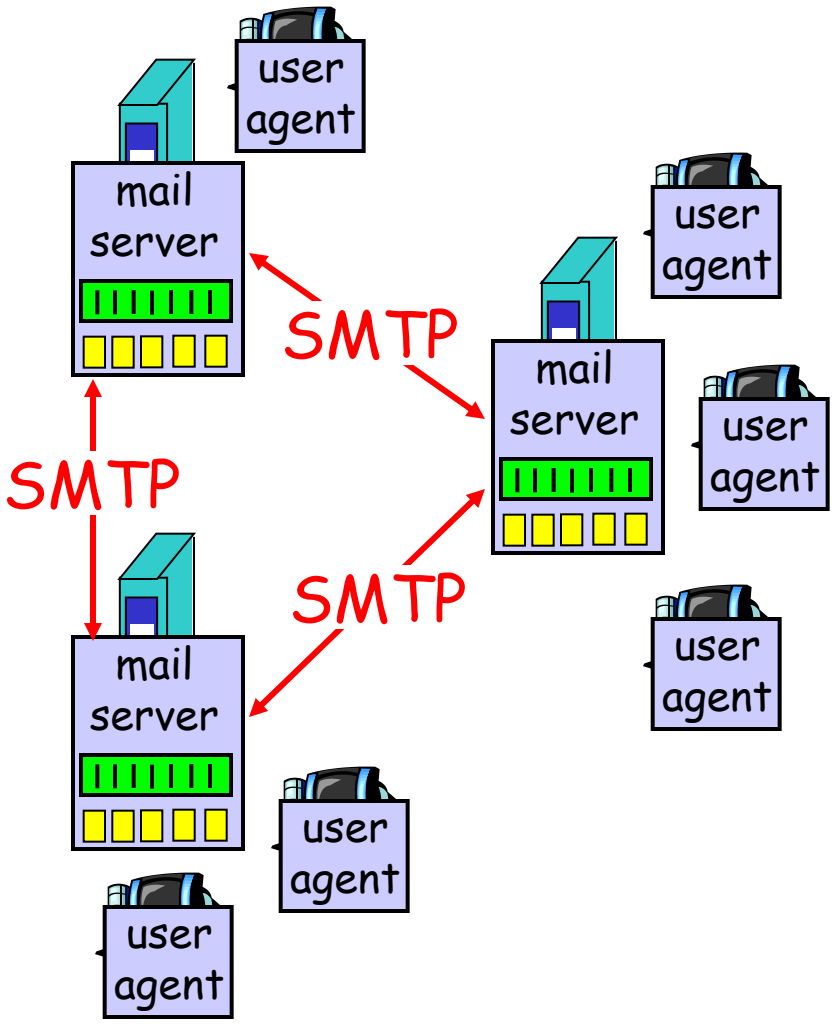
- Meccanismo: i messaggi in arrivi risiedono sul server, dove è presente una mailbox



Cosa fanno i mail server

Mail Servers

- Le mailbox accumulano i messaggi destinati agli utenti
- C'è una coda dei messaggi per i messaggi in partenza
- SMTP regola la comunicazione tra i server
  - client: mail server che invia
  - "server": mail server che riceve

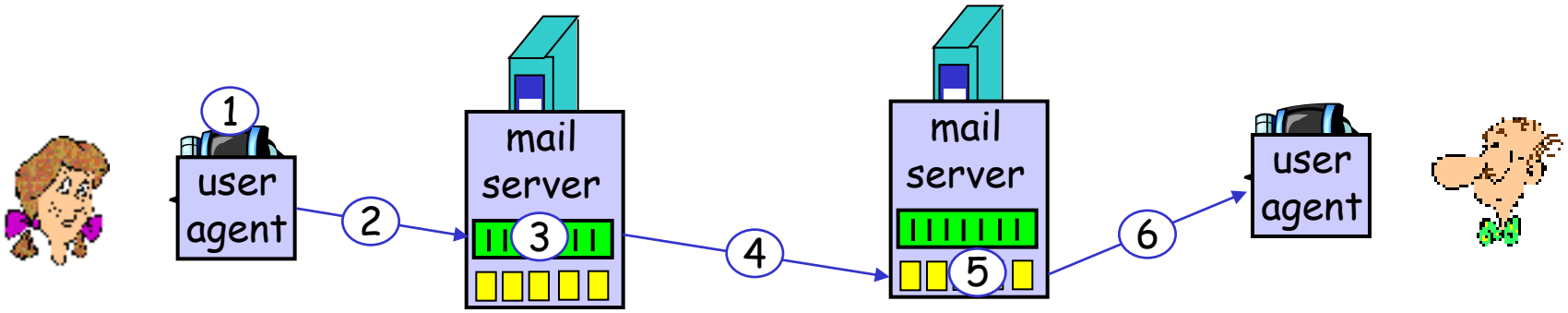


SMTP [RFC 2821]

- usa TCP per trasferire dai client ai server, attivi sulla porta **25**
- Trasferimento diretto da mittente a destinatario
- Tre fasi nel trasferimento
  - handshaking (saluti)
  - trasferimento dei messaggi
  - saluti
- Formato:
  - comandi: testo ASCII leggibile!
  - risposta: un codice di errore e una frase
- I messaggi accettano solo ASCII a 7-bit!

Scenario: Alice manda messaggio a Bob

1) Alice usa Outlook per comporre il messaggio a `bob@someschool.edu`
2) Outlook manda il messaggio al suo mail server; il messaggio è messo in coda
3) Il mail server (nel ruolo di client) apre una connessione con il mail server di Bob
4) Il messaggio è inviato tramite una connessione TCP
5) Il mail server di Bob salva il messaggio nella sua mailbox
6) A un certo punto Bob deciderà di connettersi al mail server usando un protocollo per la LETTURA dei messaggi



Una interazione di esempio

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250  Hello crepes.fr, pleased to meet you
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

Possiamo farlo da soli!

- **`telnet servername 25`**
- see 220 reply from server
- enter HELO, MAIL FROM, RCPT TO, DATA, QUIT commands

above lets you send email without using email client (reader)

SMTP: Alcune considerazioni

- SMTP usa connessioni TCP
- SMTP usa il formato ASCII a 7 bit
- SMTP usa `CRLF.CRLF` per determinare la fine di un messaggio
- SICUREZZA, AUTENTICAZIONE, CREDIBILITA?

Rispetto a HTTP:

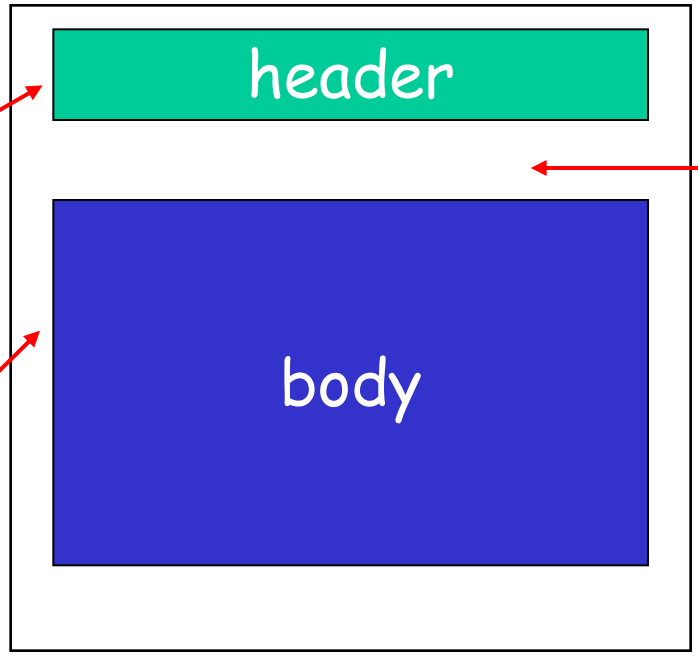
- HTTP: pull
- SMTP: push
- Entrambi funzionano con comandi ASCII facilmente interpretabili
- HTTP: Ogni oggetto viaggia di solito con una connessione separata
- SMTP: tutti gli allegati viaggiano in sequenza sulla stessa connessione

Formato dei messaggi

RFC 822: standard for text message format:

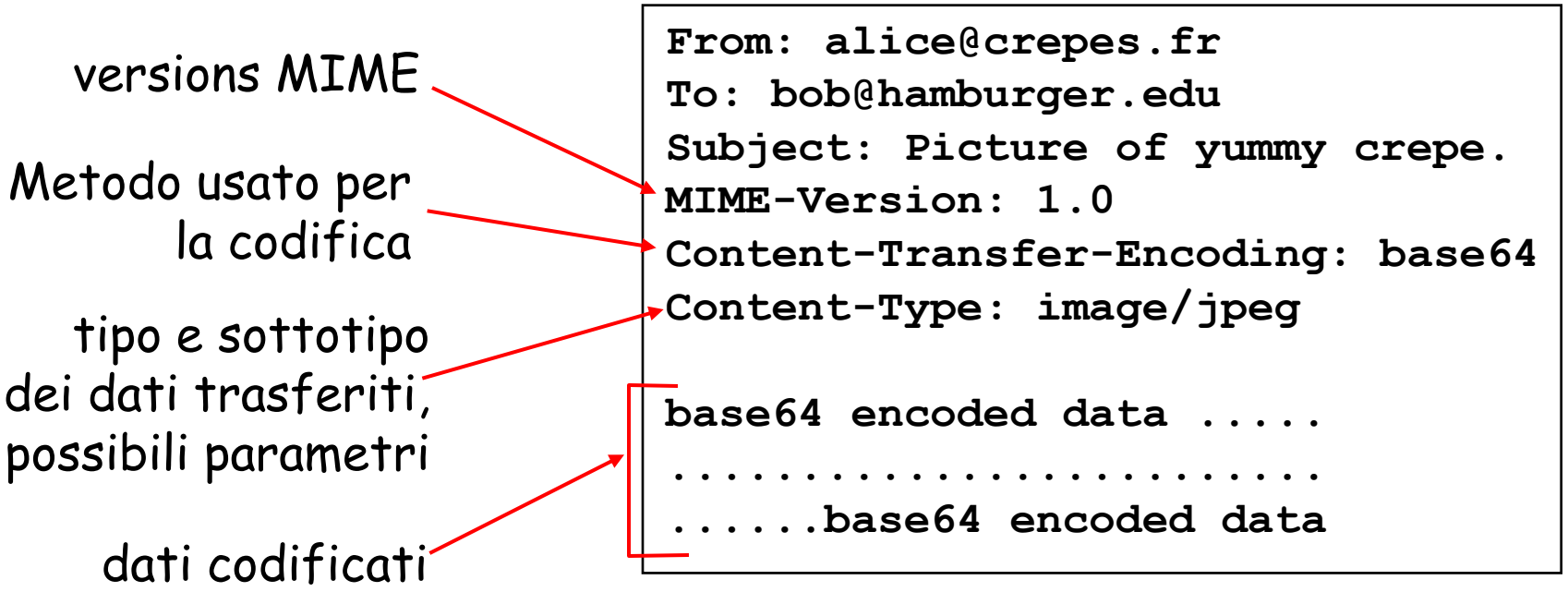
- Intestazioni, e.g.,
  - To:
  - From:
  - Subject:

  differenti dai comandi SMTP!
- body
  - il messaggio, solo caratteri ASCII validi

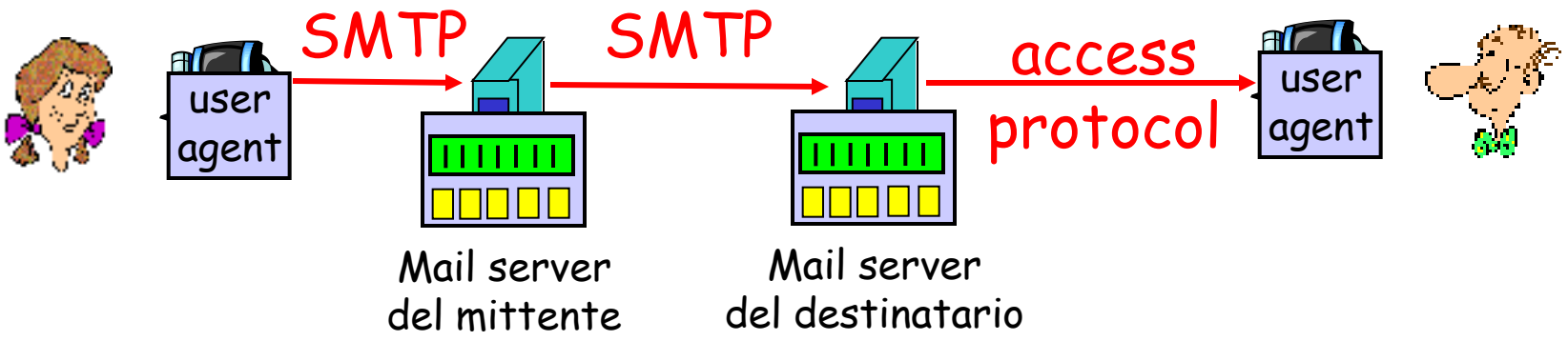


Formato: estensioni

- MIME: multimedia mail extension, RFC 2045, 2056
- delle linee addizionali nel testo dichiarano il tipo di contenuto MIME



Protocolli di lettura della posta



- SMTP: serve solo per la trasmissione non per la consultazione
- Mail access protocol: lettura dal server
  - POP: Post Office Protocol [RFC 1939]
    - autorizzazione (agent <-->server) e download
  - IMAP: Internet Mail Access Protocol [RFC 1730]
    - più sofisticato
    - si possono manipolare i messaggi sul server
  - HTTP: Hotmail , Yahoo! Mail, etc.

Protocollo POP3

Fase di autorizzazione

- comandi:
  - `user`: nome utente
  - `pass`: password
- risposte
  - `+OK`
  - `-ERR`

Fase di lettura, client:

- `list`: elenca i numeri di messaggi
- `retr`: recupera il messaggio in base al numero
- `dele`: cancella
- `quit`

```
S: +OK POP3 server ready
C: user bob
S: +OK
C: pass hungry
S: +OK user successfully logged on
```

```
C: list
S: 1 498
S: 2 912
S: .
C: retr 1
S: <message 1 contents>
S: .
C: dele 1
C: retr 2
S: <message 1 contents>
S: .
C: dele 2
C: quit
S: +OK POP3 server signing off
```

Ancora su POP3 e IMAP

Ancora su POP3

- Il precedente esempio usa una modalità "leggi e cancella" ma non è necessario

IMAP

- Tutti i messaggi restano sul server
- Si possono creare cartelle

Come si programmano i socket TCP

Il client deve contattare il server

- Il programma server deve essere già attivo e avere collegato il socket a una porta, il cui numero deve essere noto al client

Come fa il client a connettersi al server:

- Crea un socket TCP e indica a che IP:porta vuole connetterlo
- All'atto della creazione il livello trasporto si occupa di stabilire una connessione

- Quando viene contattato, il server crea una connessione per rispondere.
  - un server può parlare con **più** client sulla stessa porta
  - I client sono distinti tramite il loro numero di **ip/porta**

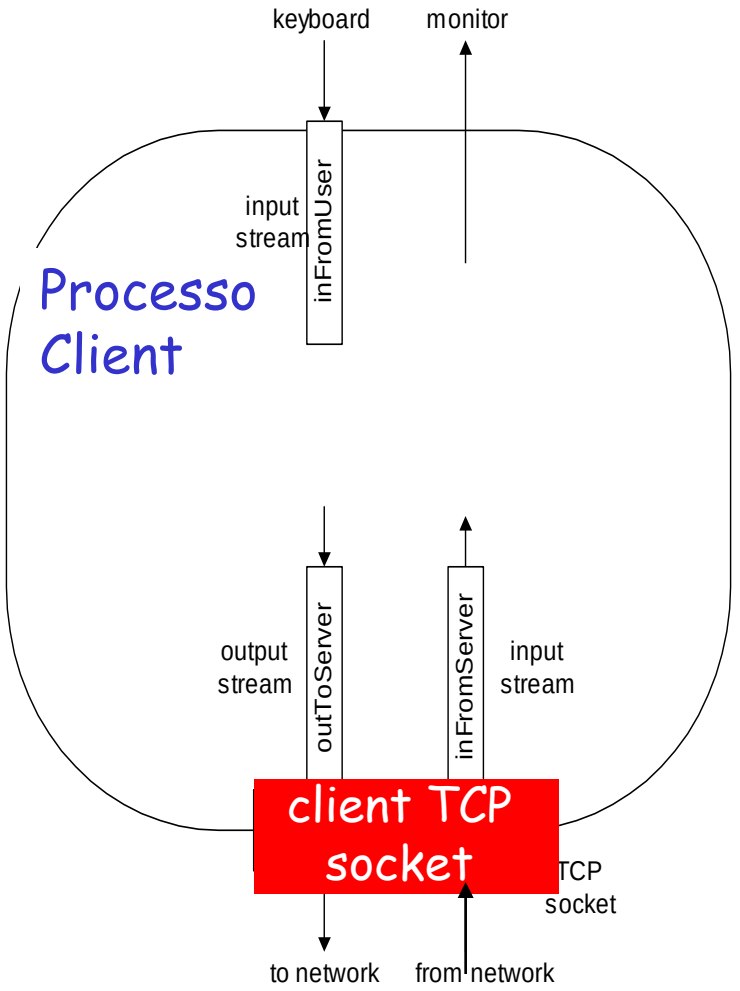
Terminologia

- Uno stream è una sequenza di caratteri che entrano o escono da un processo.
- Uno stream di input è collegato a una qualche sorgente di input, es. tastiera, socket
- Uno stream di output è collegato a una sorgente di output es. schermo, socket.

Esempio di programmazione

Esempio di applicazione client-server:

1) il client legge una linea da stdin (lo stream **`inFromUser`**), e la manda al server (tramite lo stream **`outToServer`**)
2) Il server legge una linea dal socket
3) Il server converte la linea in maiuscole e poi la manda al client così modificata
4) Il client legge la linea elaborata e la manda in output (tramite lo stream **`inFromServer`**)



Cosa succede