

## Unità 4

# Realizzare applicazioni per la comunicazione in rete

### Competenze

- Saper realizzare applicazioni di rete attraverso i socket.

### Conoscenze

- Conoscere le classi Java per la creazione e l'utilizzo dei socket.
- Conoscere le diverse modalità di trasferimento dei pacchetti di dati.
- Conoscere le principali porte e i servizi associati.

### Abilità

- Saper programmare i socket in Java.
- Saper utilizzare le porte e gli indirizzi IP per mettere in comunicazione gli host.
- Saper gestire gli stream in servizi connection-oriented.
- Saper gestire i packet in servizi connectionless.

### Prerequisiti

- Conoscere i protocolli di livello Transport.
- Conoscere i protocolli di livello Application.
- Conoscere il modello Client/Server.
- Conoscere le principali porte e i servizi associati.
- Conoscere la programmazione object-oriented.

### Accertamento dei prerequisiti



Test

- 1 Quali sono le caratteristiche principali del protocollo TCP (Transmission Control Protocol)?
- 2 Quali sono le caratteristiche principali del protocollo UDP (User Datagram Protocol)?
- 3 Su cosa si basa il modello Client/Server?
- 4 Cos'è un URL (Uniform Resource Locator)?
- 5 Come lavora il protocollo HTTP (HyperText Transfer Protocol)?
- 6 Elenca tre porte (well-known ports) riservate per applicazioni e protocolli "noti".

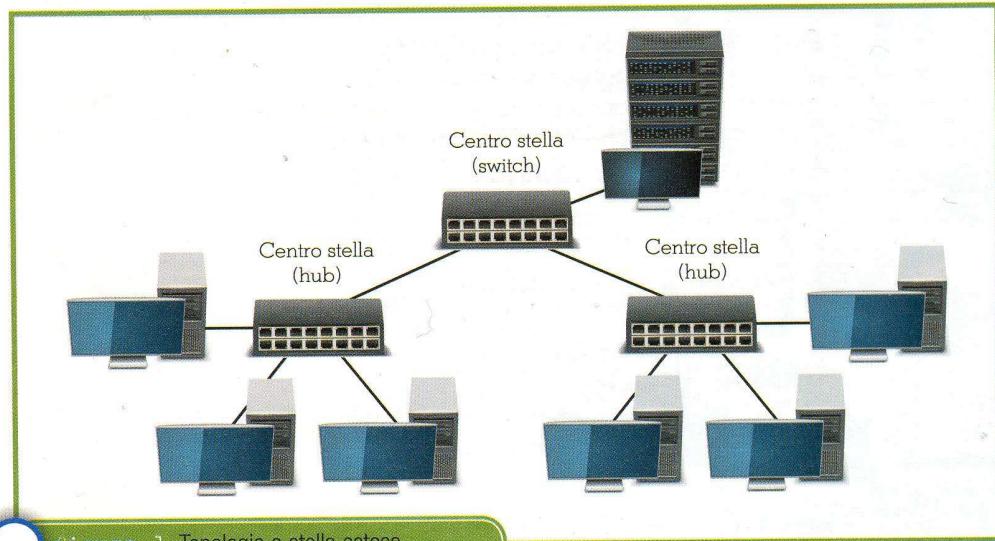
## La struttura della rete

Prima di scrivere applicazioni che consentano agli host di scambiarsi pacchetti di dati, è utile conoscere topologia, mezzi trasmissivi e apparati che rendono fisicamente possibile la trasmissione. Ovunque siano collocate le macchine (anche in remoto), la struttura della rete deve garantirne il collegamento. La struttura è ciò che garantisce a due estremità della rete (siano host, dispositivi o applicazioni) di essere collegate. Il contesto in cui operiamo presuppone che agli estremi del collegamento vi siano una macchina Server e una Client. Il modello cui facciamo riferimento in questa Unità è dunque il Client/Server.

L'architettura di rete entro cui ci muoviamo è invece TCP/IP: i suoi protocolli saranno gli unici a essere utilizzati negli esempi proposti e sviluppati in linguaggio Java. In particolare serve saper utilizzare i protocolli del livello Transport, responsabili della consegna dei pacchetti al destinatario, e i protocolli del livello Application per l'interazione con l'utente. Infine affronteremo la virtualizzazione, ormai diventata elemento fondamentale nell'organizzazione delle reti e in particolare degli applicativi per le reti.

### ● Topologia fisica

La struttura fisica prevalentemente usata nella realizzazione delle reti LAN segue la topologia a **stella estesa** ([figura 1](#)).



**figura 1** Topologia a stella estesa

La topologia a stella estesa (detta anche a **stella gerarchica**) collega tra loro più topologie a stella.

Anche se questa topologia porta a un aumento del numero dei cavi rispetto, per esempio, a quella a bus o ad anello, essa offre notevoli vantaggi in termini di:

- **fault-tolerance** (tolleranza ai guasti): il guasto di un canale o nodo della rete non ne compromette il funzionamento generale;
- **flessibilità ed espandibilità**: lo spostamento di un host da un punto a un altro della rete o l'inserimento di uno nuovo non richiedono il fermo della rete.

Per contro tale topologia è vulnerabile negli apparati che fungono da centro stella: infatti se l'apparato che svolge questo ruolo si guasta, la rete smette di funzionare.

## ● Mezzi trasmissivi

Oltre alla parte cablata, le reti LAN prevedono l'utilizzo della tecnologia wireless su ampie porzioni della rete. Si crea dunque una rete WLAN (Wireless LAN) connessa al resto della rete (**figura 2**) mediante opportuni apparati (access point).

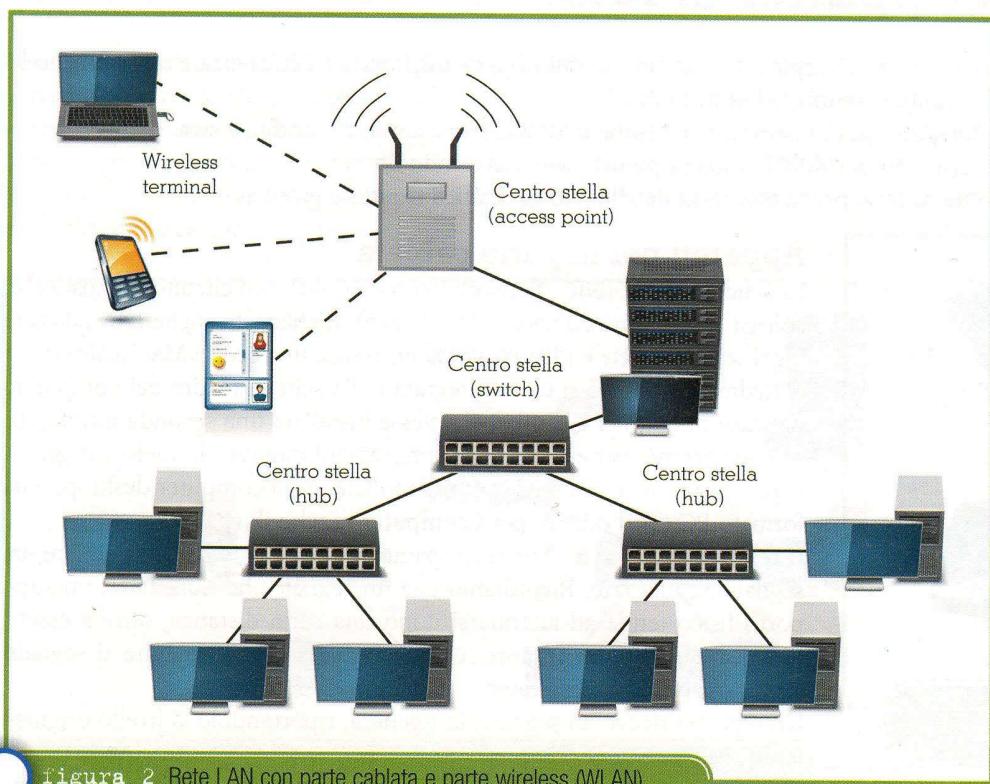


figura 2 Rete LAN con parte cablata e parte wireless (WLAN)

Per quanto riguarda le connessioni in rame, l'utilizzo del cavo **UTP** (Unshielded Twisted-Pair) è da preferirsi per l'ottimo rapporto costi/benefici. Le alternative sono il cavo **STP** (Shielded Twisted Pair) e quello **FTP** (Foil Twisted Pair) più costosi e di più difficile installazione, ma più performanti.

Il cavo UTP ha quattro coppie di fili attorcigliati e non è schermato (unshielded). Ha un'impedenza di  $100 \Omega$ , attualmente è il più usato nel mondo delle telecomunicazioni e ogni tratto raggiunge al massimo 100 m.

I suoi principali vantaggi sono:

- facilità di installazione;
- flessibilità;
- costi contenuti;
- dimensioni piccole (il diametro è di circa 0,43 cm);
- è il tipo di cavo in rame che consente la più elevata velocità di trasmissione.

I connettori usati sono gli **RJ45** (**Figura 3**).

Gli standard di cablaggio hanno introdotto due tipologie di cavi: uno per la voce e uno per i dati; quello più frequentemente raccomandato è l'UTP categoria 5 (Fast Ethernet) implementato con cavi **100Base-T** e **100Base-TX** con velocità fino a 100Mbps su distanze fino a 100 metri.

L'ultima evoluzione ha portato dalla Fast Ethernet alla Gigabit Ethernet mediante la definizione dello standard **IEEE 802.3ab** e le relative implementazioni **1000Base-T** (UTP cat. 5, 5e, 6 o 7) e **1000Base-TX** (UTP cat. 6 o 7) che consentono velocità fino a 1 Gbps e distanze sempre fino a 100 metri.

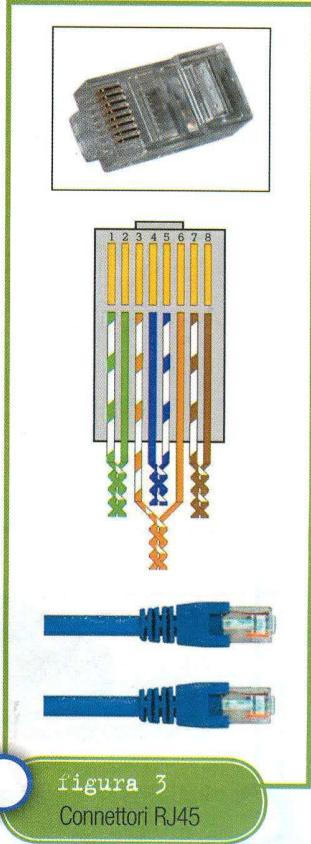


figura 3  
Connettore RJ45

Per quanto riguarda la parte wireless della rete (WLAN), lo standard più diffuso è l'**IEEE 802.11**, con versioni che lavorano nelle bande di frequenza comprese tra 2,4 GHz e 5,2 GHz. Le varie versioni, dalla 802.11b in poi, risultano compatibili tra loro e vanno sotto il nome di **WiFi** (Wireless Fidelity). Se un dispositivo WLAN soddisfa le specifiche WiFi, allora è garantita l'interoperabilità del dispositivo stesso con gli altri dispositivi WiFi.

## ● Apparati di rete

La necessità di segmentare la rete in subnet per migliorarne l'efficienza e aumentarne la flessibilità, comporta l'utilizzo di:

- apparati per la parte cablata (scheda di rete, repeater, hub, bridge e switch);
- apparati per il WiFi (access point e wireless terminal);
- dispositivi per la connessione alla rete geografica (router e gateway).

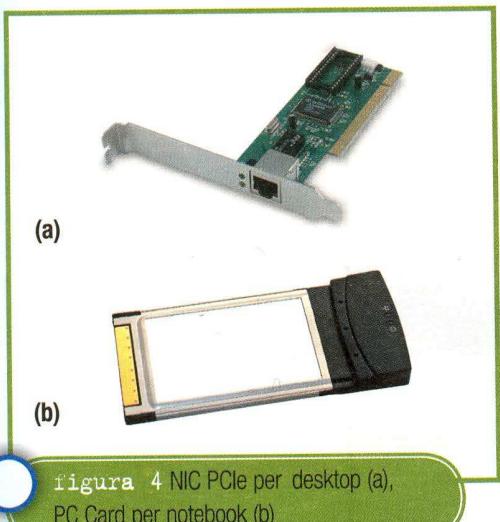


figura 4 NIC PCIe per desktop (a),  
PC Card per notebook (b)

### Apparati per la parte cablata

La **scheda di rete** (NIC, Network Interface Card) è un circuito stampato che collega l'host al mezzo (cioè il PC al cavo). È chiamata anche *LAN adapter*. Ogni scheda di rete è identificata da un codice univoco (il *MAC address*).

Attualmente la NIC si trova integrata nella scheda madre del computer; nel caso in cui, per esempio, si volesse installare una seconda interfaccia di rete (perché, per esempio, con prestazioni migliori di quella integrata) si può ricorrere a una NIC in formato PCIe, per i computer desktop, o in formato PC Card o USB, per i computer notebook (**figura 4**).

Il **repeater** (**figura 5**) è un apparato che permette di ritrasmettere un segnale su una rete. Ricordiamo che un segnale che transita su un supporto fisico tende ad attenuarsi dopo una certa distanza, oltre a essere distorto a causa del rumore, è quindi necessario rigenerare il segnale tramite appunto dei repeater.

Un repeater riceve un segnale, lo rigenera, riportandolo al livello originale, lo risincronizza e lo passa oltre.

Il repeater serve quindi a estendere la lunghezza del canale trasmissivo su LAN omogenee. Per una rete LAN Fast Ethernet vale la regola dei 4 ripetitori: tra 2 host non possono esserci più di 4 ripetitori, questo per evitare la latenza, cioè il ritardo con cui un segnale arriva a destinazione. Una latenza troppo alta rende la rete meno efficiente.

Gli **hub** sono dei repeater multiporta, e in genere possiedono da 4 a 24 porte (**figura 6**). I dati che arrivano su una porta qualsiasi sono ripetuti.

L'hub (o concentratore) ha il compito di ricevere le informazioni dai vari nodi presenti sulla rete e di inoltrarle agli altri nodi collegati alle sue porte. L'hub non è in grado di verificare quale sia il reale destinatario di tali dati, per cui li invia su tutte le porte tranne a quella da cui sono arrivati (modalità broadcast). Saranno gli stessi dispositivi riceventi a valutare se i dati inviati dall'hub siano o meno di loro pertinenza e, in caso contrario, a rifiutarli senza processarli.

Tale operazione, oltre a provocare un traffico inutile sulla rete, crea anche incertezze sulla sicurezza dei dati stessi. Infatti, bisogna considerare che tutte le informazioni potranno essere lette anche dai dispositivi a cui non sono realmente destinate.



figura 5 Repeater per cavi di categoria 5 e 6



figura 6 Hub a 16 porte

Ci sono 3 tipi di hub:

- **passivi**: servono solo come punto di connessione fisica, non vedono i dati che passano. Essendo passivi non necessitano di alimentazione elettrica;
- **attivi**: hanno bisogno di alimentazione elettrica per amplificare e ripulire i segnali che arrivano e trasmetterli sulle altre porte;
- **intelligenti**: chiamati anche *smart hub*, funzionano come gli hub attivi ma al loro interno hanno un microprocessore che fornisce informazioni di diagnostica. Sono più costosi degli hub attivi ma sono utili nelle situazioni di *troubleshooting* (ricerca del guasto).

I bridge sono dispositivi che permettono di collegare tra loro reti differenti (**figura 7**), purché utilizzino lo stesso protocollo. Rendono quindi possibile la suddivisione di grosse reti creando delle sottoreti in modo da facilitare la gestione e il controllo delle stesse. Possono anche permettere la creazione di macroreti partendo da reti locali già esistenti. È possibile in questo modo creare delle reti dipartimentali che verranno poi inglobate nell'unica rete aziendale.

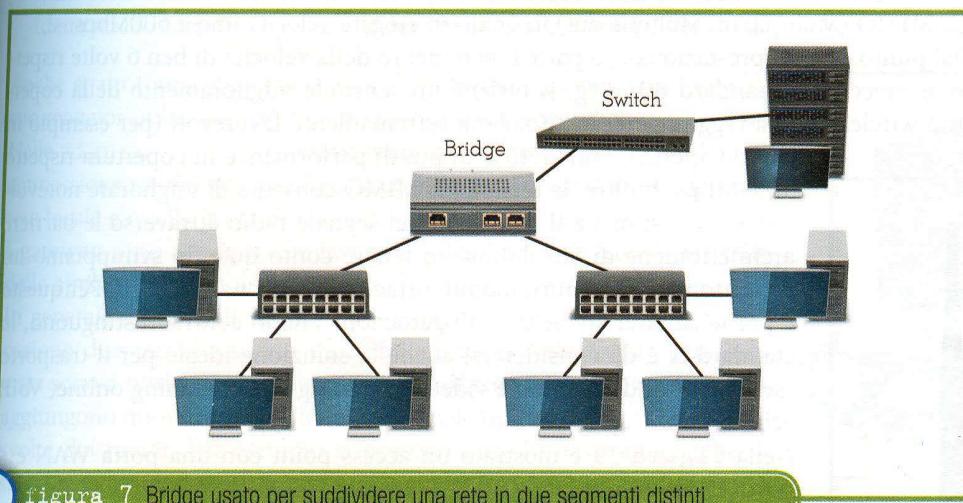


figura 7 Bridge usato per suddividere una rete in due segmenti distinti

Uno **switch** si può definire come un bridge multiporta, infatti mentre un bridge in genere ha 2 porte, lo switch ne ha può avere 24, 32 o più.

Dovendo decidere su più porte, uno switch è un apparato più intelligente di un bridge da cui infatti differisce per le modalità con cui tratta e inoltra i dati. Uno switch è in grado di analizzare il contenuto di un pacchetto di dati ricevuto e di inoltrarlo solo al reale destinatario, riducendo in tal modo il traffico superfluo nella rete. Inoltre gli switch operano a velocità più alte dei bridge.

Uno switch permette che più comunicazioni avvengano in parallelo, infatti durante la comunicazione collega solo le due porte interessate, per cui ci possono essere più colloqui contemporanei con conseguente aumento della *bandwidth* totale.

Gli switch attuali offrono la funzionalità di autoconfigurazione, cioè sono in grado di riconoscere dinamicamente il dispositivo all'altro capo del cavo, adeguando di conseguenza il collegamento interno per garantire una corretta comunicazione. In questo modo non è più necessario usare cavi cross ma si possono usare sempre cavi dritti, in quanto lo switch configura in automatico la porta in modo diverso se dall'altra parte del cavo c'è un altro switch.

Esistono modelli di switch di tipo "ibrido" ossia con porte che usano differenti velocità, per esempio porte 10/100 e porte 1000 Mbps. In **figura 8** è mostrato uno switch con 48 porte 10/100 e 2 porte Gigabit.

### Apparati per la parte wireless

I dispositivi che costituiscono le reti wireless sono due:

- **Wireless Terminal (WT)**: sono dispositivi mobili (notebook, netbook, tablet, palmari, cellulari, smartphone ecc.) dotati di interfaccia



figura 8 Switch multiporta

802.11 integrata o su schede PCMCIA o USB, oppure fissi (personal computer) con schede PCI o adattatori USB.

- **Access Point (AP):** hanno un doppio scopo, da un lato sono bridge che collegano la parte cablata (*wired*) con la parte wireless, dall'altro consentono ai WT di collegarsi alla rete wireless (agiscono quindi da gateway).

Gli access point in commercio utilizzano la tecnologia **wireless-N**, ratificata da IEEE con lo standard **802.11n** (compatibile con le versioni 802.11b/g).

Lo standard ufficialmente ratificato consente il continuo sviluppo armonizzato delle tecnologie dual e multi stream. Per *streams* si intendono i flussi di trasmissione e ricezione che un apparato wireless 802.11n è in grado di emettere. Ogni stream è capace di una velocità di 150Mbps, circa 3 volte superiore a quella raggiungibile con il precedente standard 802.11g. Per esempio, apparati dual stream (2x2) sono in grado di raggiungere velocità di 300Mbps in quanto sono dotati fisicamente di 2 moduli radio di trasmissione e di 2 di ricezione che, uniti, danno la velocità indicata.

Sistemi multi stream hanno più moduli radio (fino a 4) uniti insieme attraverso la tecnologia **MIMO** (Multiple in, Multiple out) in grado di erogare velocità fino a 600Mbps.

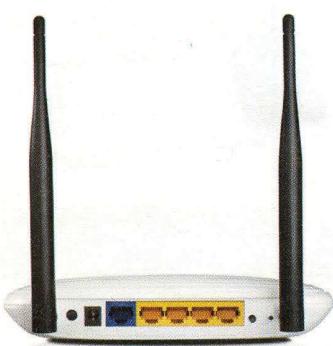
Dal punto di vista prestazionale, a parte l'incremento della velocità di ben 6 volte rispetto al precedente standard 802.11g, si ottiene un sensibile miglioramento della copertura wireless fino a raggiungere, in situazioni estremamente favorevoli (per esempio in campo aperto), fino al 40% in più di performance in copertura rispetto al 54Mbps. Inoltre, la tecnologia MIMO consente di migliorare notevolmente la gestione e il passaggio del segnale radio attraverso le barriere architettoniche di cui dobbiamo tenere conto quando sviluppiamo un progetto wireless (muri, mobili, ostacoli di varia natura ecc.). Per questo e per le caratteristiche e la progettazione che lo contraddistinguono, lo standard N è da considerarsi anche la soluzione ideale per il trasporto "senza fili" di dati, audio e video (streaming video, gaming online, VoIP, ecc.).

Nella **figura 9** è mostrato un access point con una porta WAN e 4 porte LAN.

La porta WAN consente di collegare direttamente l'access point al router per il collegamento alla rete geografica.

Le 4 porte LAN consentono il collegamento verso computer, hub e switch e dunque consentono il collegamento tra i due segmenti, quello cablato e quello wireless (fungono cioè da bridge).

**figura 9** Access point con una porta WAN e 4 porte LAN



### Dispositivi per la connessione alla rete geografica

Il **router** (o "instradatore") è un dispositivo hardware che si occupa di far comunicare tra loro reti differenti ed eterogenee. Il router, in particolare, è il dispositivo utilizzato per permettere l'accesso di tutti i computer di una rete LAN a un'altra rete (per esempio a Internet). Reti diverse parlano "linguaggi" diversi quindi, a livello di trasmissione fisica, di accesso e di controllo, per collegare tra loro due reti non è sufficiente metterle in comunicazione tramite un bridge o uno switch. È necessario, invece, che tra una rete e l'altra venga posto un apposito dispositivo, il router, che parli i protocolli di entrambe le reti e provveda a leggere, tradurre e rispedire (*store and forward*) i dati che lo attraversano.

Il router, quindi, è connesso a due o più reti e si occupa dell'indirizzamento dei messaggi decidendo quale percorso far compiere ai dati sulla base delle informazioni dello stato delle reti alle quali è collegato, cioè determinando il successivo punto della rete a cui inoltrare il pacchetto di dati ricevuto.

Le due attività principali di un router sono dunque:

- scegliere il percorso migliore;
- mettere i pacchetti sull'interfaccia in uscita corretta.

Il router è un **computer dedicato al routing** (instradamento dei pacchetti), necessita di un sistema operativo e dal punto di vista hardware è dotato di almeno due schede di rete e di:

- **CPU:** è un microprocessore che esegue le istruzioni del sistema operativo;

- **RAM**: è una memoria volatile usata per memorizzare la tabella di routing, il file di configurazione, i pacchetti in attesa. La RAM viene condivisa come memoria del processore e memoria di input/output (I/O) per i pacchetti in attesa; la RAM in genere è dinamica (DRAM);
- **Flash**: è una memoria di tipo read-only cancellabile e riprogrammabile, è usata per contenere il sistema operativo e mantiene il suo contenuto anche se non c'è alimentazione (a differenza della RAM);
- **NVRAM**: (Non Volatile RAM) è usata per memorizzare il file di configurazione di startup (quello che viene eseguito all'accensione del router) e come la memoria flash mantiene il suo contenuto anche se non c'è alimentazione;
- **Bus**: la maggior parte dei router contiene un system bus e un CPU bus. Il system bus si usa per le comunicazioni tra la CPU e le interfacce, mentre il CPU bus è usato per accedere alle memorie;
- **ROM**: è usata per contenere i programmi di diagnostica allo startup dell'hardware;
- **Interfacce**: sono le schede di rete del router usate per le connessioni verso l'esterno; generalmente sono di 3 tipi: LAN, WAN, gestionale. Quelle LAN in genere sono di tipo Ethernet o Token Ring, quelle WAN possono essere seriali o ISDN. La porta gestionale (detta anche console o AUX) è usata per la configurazione del router.

Anche se un router può essere usato per segmentare una rete locale, il suo maggior utilizzo è come dispositivo per reti geografiche.

Nelle **figure 10 e 11** sono mostrati due router: il primo per l'interconnessione tra reti geografiche, il secondo per l'interconnessione tra la rete locale e quella geografica.

I **gateway** sono apparati che lavorano sia a livello di rete sia a livello delle applicazioni che usano la rete.

In generale si tratta di sistemi che permettono di mettere in comunicazione due reti che usano differenti protocolli. Da questo punto di vista sono a tutti gli effetti dei router; la differenza è che nei gateway si aggiungono nuove funzionalità oltre a quelle tipiche di *store and forward* svolte dai router. Per esempio si possono introdurre caratteristiche di sicurezza che rendono il router anche firewall.

Un gateway può essere implementato completamente in hardware o completamente in software o un mix di entrambe le soluzioni. Spesso nelle piccole reti locali si prende un "vecchio" computer con un'installazione di Linux (sistema operativo adatto alla gestione della rete e che non richiede configurazioni elevate), lo si equipaggia con due schede di rete (NIC), lo si configura come gateway e, attraverso esso, si rende condiviso l'accesso a Internet per tutti i computer della rete locale.

Spesso i gateway svolgono funzionalità per le applicazioni di rete, per esempio possono agire come Proxy Server per i servizi di connessione a Internet, oppure come traduttore per i servizi di posta elettronica.

Tutti i mezzi trasmissivi, gli apparati e i dispositivi visti in questa lezione, vengono utilizzati nella progettazione delle reti LAN.



**figura 10** Router di fascia "alta", usato su reti geografiche



**figura 11** Router di fascia "bassa", usato in reti locali per l'accesso all'esterno



### verifica le tue conoscenze

- 1 In cosa differiscono la topologia a stella e quella a stella estesa?
- 2 Quali sono i vantaggi nell'uso del cavo UTP?
- 3 Quali sono gli apparati necessari per segmentare una rete?
- 4 Descrivi le caratteristiche del router.

## Realizzare applicazioni C/S attraverso la programmazione socket in Java

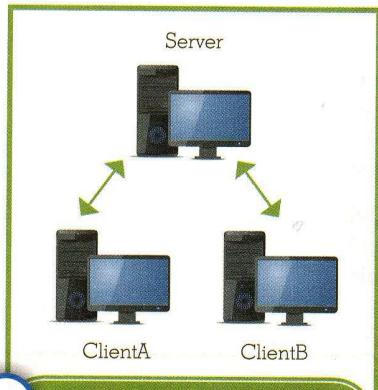


figura 1 Schema di un sistema Client/Server

### ● Introduzione

Il sistema Client/Server (C/S) è il paradigma applicato nella rete **Internet** (TCP/IP): ogni servizio offerto sulla rete ha una componente **Client** e una **Server** (**figura 1**):

- **Server:** è un processo che offre un servizio e può essere raggiunto attraverso la rete, è in grado di accettare le richieste che gli arrivano dai Client, elaborarle, effettuare il servizio richiesto e restituire il risultato al richiedente (o un messaggio di errore se non è riuscito a soddisfare la richiesta); solitamente il processo Server viene avviato all'accensione dell'host e rimane sempre attivo.
- **Client:** è un processo che invia una richiesta a un Server e resta in attesa della risposta; tipicamente diventa attivo quando deve inviare una richiesta e, una volta ricevuta la relativa risposta, diventa inattivo.



Paradigma **Client/Server**: un host, detto *Client*, richiede un servizio a un altro host, detto *Server*, che esegue la richiesta e manda una risposta al Client.

Esempi di applicazioni sono: il Web Server in cui il Client è il browser (HTTP Client) e il Server è un software “HTTP Server” come Apache; altro esempio è il servizio di posta elettronica con SMTP Client e SMTP Server.

In questa UD ci occuperemo di realizzare semplici applicazioni C/S, in Java, attraverso i socket.

### ● La programmazione socket in Java

I socket possono essere pensati come dei canali tramite i quali i processi, che sono messi in comunicazione, possono inviare e ricevere dati, quindi si tratta di un canale **bidirezionale** (**figura 2**). Per analogia ricordano il meccanismo della *pipe* tipica dei sistemi operativi Unix-like che prevede un canale di comunicazione su cui i processi che girano sulla stessa macchina possono leggere/scrivere dati. Qui sta la differenza tra pipe e socket: i processi che comunicano tramite socket non necessariamente devono risiedere sulla stessa macchina.

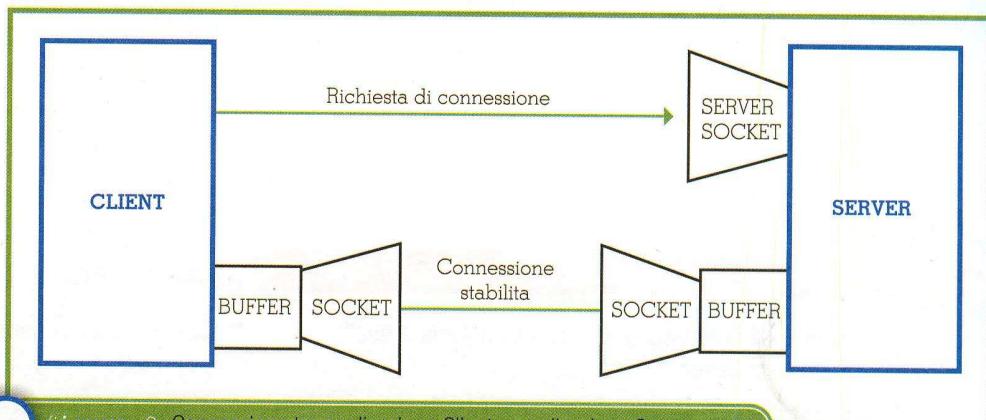


figura 2 Connessione tra applicazione Client e applicazione Server

Un **socket** è un'astrazione per un estremo di una comunicazione TCP/IP.



La sequenza di funzionamento può essere riassunta nei seguenti passi:

1. ...Server in ascolto
2. Client richiede connessione per un servizio
3. Server accetta connessione e svolge il servizio richiesto
4. Client attende risposta
5. Server invia risposta e chiude connessione
6. Client riceve risposta, chiude connessione e termina
7. Server in ascolto...

Da notare che l'interfaccia dei socket è assolutamente generale, quindi possono essere utilizzati anche con altri meccanismi di comunicazione che non siano i protocolli dello stack TCP/IP.

In generale i socket possono:

- essere basati su una connessione o connectionless;
- scambiare dati tramite singoli messaggi o con un flusso continuo di byte;
- fornire un servizio affidabile o inaffidabile.

L'*Application Program Interface* (API) basata sui **Berkeley Sockets** è molto diffusa e usata in molti sistemi, non solo quelli Unix-like. Per esempio, è alla base dei Sockets Windows di Microsoft (WSOCK).

In **tabella 1** sono descritti i *Berkeley Sockets* che rappresentano le primitive utilizzate per TCP dal sistema operativo Unix nella versione di Berkeley.

**tabella 1** Berkeley Sockets

Primitiva	Descrizione	Applicazione che la usa
<b>SOCKET</b>	Crea un nuovo punto di comunicazione ( <i>end point</i> ). I parametri specificano il formato degli indirizzi, il tipo di servizio (per es. <i>reliable byte stream</i> ) e il protocollo (per es. TCP). Restituisce un descrittore del file da usare nelle chiamate successive, in modo analogo a quanto fatto da una normale <i>Open</i> di un file	CLIENT e SERVER
<b>BIND</b>	Associa ( <i>bind</i> ) un indirizzo locale a un socket, così da consentire ai Client remoti di connettersi	SERVER
<b>LISTEN</b>	Alloca spazio per accodare le chiamate entranti nel caso in cui più Client cerchino di connettersi al Server contemporaneamente. LISTEN non è bloccante	SERVER
<b>ACCEPT</b>	Blocca il Server finché non riceve da un Client una <i>Connect Request</i> . Il TCP crea un nuovo socket con le stesse caratteristiche di quello usato con la LISTEN e viene restituito il relativo descrittore di file. A questo punto il Server crea un nuovo processo, (tramite <i>fork</i> ) o un thread, per gestire la connessione sul nuovo socket e ritorna in attesa di una nuova richiesta di connessione sul socket originario inviando una primitiva ACCEPT	SERVER
<b>CONNECT</b>	Questa primitiva è usata dal Client dopo che ha creato un socket. Essa blocca il Client e attiva il processo che gestisce l'instaurazione della connessione con il Server. Quando questa è avvenuta (a seguito della ricezione della primitiva di conferma dal Server), il processo Client è sbloccato e la connessione è attiva	CLIENT
<b>SEND</b>	Invia i dati sulla connessione tra Client e Server	CLIENT e SERVER
<b>RECEIVE</b>	Blocca l'applicazione (Client o Server) in attesa di ricevere i dati dalla connessione	CLIENT e SERVER
<b>CLOSE</b>	Rilascio simmetrico della connessione: sia l'applicazione Client sia quella Server devono chiudere la connessione	CLIENT e SERVER



I dati per la creazione di un socket sono **l'indirizzo IP** dell'host con il quale si vuole comunicare e **una porta** di comunicazione.

Una porta è un intero da 0 a 65.535 ( $2^{16}-1$ ) che verrà utilizzato in fase di comunicazione per distinguere le varie connessioni.

Le porte tra 0 e 1023 (*well-known ports*) sono considerate riservate per applicazioni e protocolli "noti" (HTTP:80, FTP:21, SMTP:25 ecc.).

Ogni linguaggio di programmazione, che sia di alto o basso livello, deve fornire delle API che permettano di lavorare con i socket.

Il linguaggio che utilizzeremo è **Java**, in quanto fornisce un'API *object-oriented* completa e semplice da usare.

La programmazione socket in Java è supportata dal package **java.net** in cui si trovano le classi utili per realizzare applicazioni in rete.

Le principali tra queste classi sono:

1. per utilizzare TCP le classi: **ServerSocket** e **Socket**
2. per utilizzare UDP le classi: **DatagramPacket** e **DatagramSocket**
3. per utilizzare HTTP le classi: **URL** e **URLConnection**

### Le classi **ServerSocket** e **Socket** per TCP

**Transmission Control Protocol** è un protocollo di trasporto molto diffuso, più di UDP, in quanto offre un servizio **connection-oriented** e **affidabile**, garantendo quindi la consegna dei dati in modo ordinato (infatti si parla di *data stream* inteso come *flusso di byte*).

La connessione che TCP stabilisce tra mittente e destinatario offre all'applicazione, che si trova al livello superiore, l'impressione di usufruire di una linea dedicata. La connessione può quindi essere intesa come un canale logico con le seguenti caratteristiche:

- è **full-duplex**: sulla stessa connessione si può trasmettere e ricevere in contemporanea;
- è **point-to-point**: un solo mittente e un solo destinatario;
- necessita l'inizializzazione di *variabili di stato* da parte del mittente e del ricevente (i due processi si devono accordare, prima di iniziare il trasferimento dati).

Il protocollo TCP offre dunque due grossi vantaggi:

- **garanzia di consegna**: se un pacchetto non arriva a destinazione viene rilevato automaticamente tramite un sistema di acknowledgement e timeout, e viene ritrasmesso finché non raggiunge il mittente;
- **ordinamento dei pacchetti**: nel caso in cui i pacchetti arrivino al destinatario in ordine diverso da quello di invio il protocollo esegue un riordinamento; in questo modo la lettura e la scrittura su un canale risultano sequenziali come un qualsiasi altro *stream* di dati.



Grazie a tali garanzie, a livello applicazione possiamo evitare di preoccuparci di molte problematiche sulla consegna di pacchetti.

Vediamo ora come utilizzare le due classi **ServerSocket** e **Socket** e il loro metodi per realizzare un'applicazione Client/Server sotto protocollo TCP.

- Il programma **Server** crea un socket mediante l'apposita classe **ServerSocket** chiamando il costruttore **ServerSocket(port)**, che specifica nel parametro *port* il numero della porta sulla quale il Server si metterà in ascolto delle richieste da parte dei Client (il numero della porta dev'essere maggiore di 1023).
- I programmi **Client** creano la connessione con il socket mediante la classe **Socket** chiamando il costruttore **Socket(serverAddress, port)**, che specifica nel parametro *serverAddress* l'indirizzo IP del Server e nel parametro *port* lo stesso numero di porta usato dal Server.

Il Server, ogni volta che riceve una richiesta di connessione da un Client, utilizza il metodo **accept()** della classe **ServerSocket** per creare un socket che gestisca la connessione con quel Client inviando quanto richiesto.

I principali metodi della classe **ServerSocket** sono:

- **ServerSocket(int port)**: costruttore per la creazione di un ServerSocket associato a una determinata porta;
- **accept()**: metodo per stare in ascolto e ottenere il socket di un Client che richiede la connessione;
- **setSoTimeout(int time)**: metodo per impostare un valore di timeout.

I principali metodi della classe **Socket** sono:

- **Socket()**: costruttore per la creazione di un socket non ancora connesso;
- **Socket(InetAddress address, int port)**: costruttore con parametri per la creazione di un socket connesso all'indirizzo *address* sulla porta *port*;
- **getInputStream(), getOutputStream()**: metodi per ottenere gli *stream* in ingresso e in uscita della connessione;
- **close()**: metodo per chiudere la connessione;
- **setSoTimeout(int time)**: metodo per impostare un valore di timeout;
- **connect(SocketAddress endpoint)** e **connect(SocketAddress endpoint, int timeout)**: metodi per connettersi a un determinato host, specificando eventualmente un valore di timeout;
- **isConnected()**: metodo che restituisce *true* se il socket è ancora connesso;
- **getPort(), getLocalPort(), getInetAddress(), getLocalAddress(), getSoTimeout()...**: metodi per l'interrogazione delle proprietà del socket.



Quasi tutti i metodi della classe **Socket** lanciano eccezioni derivate da **IOException** in caso di varie situazioni di errore (per es. host non trovato o irraggiungibile, timeout scaduto).

## Le classi **DatagramPacket** e **DatagramSocket** per UDP

User **Datagram Protocol** è un protocollo del livello Transport che non prevede l'uso di una connessione tra host mittente e host destinatario (**connectionless**), infatti ciascun datagram UDP è trattato in modo indipendente.

Il servizio offerto da UDP è di tipo Best Effort: i datagram UDP possono essere persi o arrivare fuori sequenza, non si ha quindi alcuna garanzia sulla consegna dei dati trasmessi. UDP fornisce le funzionalità tipiche del livello Transport in termini di multiplexing, grazie all'uso delle porte, e di controllo dell'integrità dei dati.

Alcuni dei vantaggi derivanti dall'uso di **UDP** sono:

- non richiede di stabilire una connessione (**connectionless**); non introduce un ritardo dovuto alla fase di set-up della connessione;
- non mantiene lo stato della connessione: un Server può supportare molti più Client attivi;
- il sovraccarico dovuto all'intestazione del pacchetto è minimo: solo 8 byte contro i 20 di un protocollo connection-oriented;
- il controllo del livello applicativo è più efficace: in mancanza di un controllo della congestione, il mittente non viene mai bloccato.

Non essendo UDP orientato alla connessione, tra due host non si crea uno *stream* stabile.

UDP si usa soprattutto quando non è importante che tutti i **datagram** arrivino a destinazione o che arrivino in ordine sequenziale.



Vediamo ora come utilizzare le due classi **DatagramPacket** e **DatagramSocket** e i loro metodi per realizzare un'applicazione Client/Server sotto protocollo UDP.

Non potendo creare uno stream stabile come avviene per TCP, Client e Server devono preparare dei pacchetti (detti **datagram**) per l'invio e la ricezione dei dati. Inoltre Client e Server dovranno aprire ciascuno un apposito socket (detto **datagram socket**) per scambiarsi i datagram.

Per creare i socket i due programmi useranno la classe **DatagramSocket** chiamando il costruttore **DatagramSocket(port)**, mentre per preparare i datagram useranno la classe **DatagramPacket** chiamando per la ricezione il costruttore **DatagramPacket(byte\_datagram, lunghezza\_datagram)** e per l'invio il costruttore **DatagramPacket(byte\_datagram, lunghezza\_datagram, numero\_porta)**.

Un oggetto della classe **DatagramPacket** rappresenta un pacchetto UDP.

I suoi principali metodi sono:

- **DatagramPacket(byte[] buf, int length)**: costruttore, buf rappresenta l'array dove verranno messi i byte del pacchetto e length il numero di byte da leggere;
- **DatagramPacket(byte[] buf, int length, InetAddress address, int port)**: costruttore dove vengono specificati indirizzo e porta di destinazione, oltre al numero length di byte da inviare;
- **byte[] getData(), int getLength()**: metodi per estrarre i dati ricevuti e la loro lunghezza;
- **InetAddress[] getAddress(), int getPort()**: metodi per ottenere indirizzo e porta dell'host remoto che ha mandato o ricevuto il pacchetto;
- **void setAddress(InetAddress addr), void setPort(int p), void setData(byte[] b), void setLength(int l)**: metodi per impostare i vari campi del pacchetto.

La trasmissione di pacchetti UDP avviene tramite la classe **DatagramSocket**. I suoi principali metodi sono:

- **DatagramSocket()**: costruttore per la creazione di un DatagramSocket associato a una qualsiasi porta libera del sistema;
- **DatagramSocket(int port)**: costruttore per la creazione di un DatagramSocket in ascolto sulla porta *port* specificata;
- **void send(DatagramPacket d)**: metodo per inviare il pacchetto UDP specificato; le informazioni sull'host di destinazione sono contenute nel pacchetto;
- **void receive(DatagramPacket d)**: metodo che consente di restare in ascolto per ricevere il pacchetto UDP; le informazioni sul numero massimo di byte da ricevere sono nuovamente contenute nel pacchetto;
- **void connect(InetAddress address, int port)**: metodo che associa il socket a un indirizzo IP e a una porta di un host, forzando i pacchetti a essere inviati e ricevuti solo da tale host. Non apre però una connessione;
- **void disconnect()**: metodo che annulla l'effetto di una precedente operazione connect();
- **void setSoTimeout(int time)**: metodo che assegna un tempo di timeout per l'operazione di ricezione.

## Le classi URL eURLConnection per HTTP

HyperText Transfer Protocol lavora con il World Wide Web, che è la parte di Internet più usata e cresciuta più velocemente. Un web browser è un'applicazione Client/Server che presenta i dati in formati multimediali sulle pagine web. Il protocollo HTTP regola lo scambio di messaggi tra il Web Server e il Web Client (si parla anche di HTTP Server e HTTP Client o anche di WWW Server e WWW Client). Nell'uso comune il Client corrisponde al browser e il Server al sito web.

Ogni pagina web ha un suo indirizzo simbolico detto **URL (Uniform Resource Locator)**, per esempio <http://www.eclipse.org/downloads/> dove la parte iniziale (qui **http://**) dice al browser il protocollo da usare, la seconda parte dice al browser quale risorsa andare a prendere (un documento, un'immagine, un video) e il nome della macchina specifica, tipicamente un host Server, su cui risiede.

Un **URL** (Uniform Resource Locator) è un identificatore di una risorsa nel Web.

La sua sintassi è: **protocollo://host[:porta][percorso][file][?query][#pos]**.

Gli URL vengono utilizzati per la comunicazione tramite protocolli a livello applicativo come HTTP o FTP.



Un oggetto della classe **URL** rappresenta un Uniform Resource Locator. I suoi principali metodi sono:

- **URL(String url)**: costruttore tramite la rappresentazione dell'URL come stringa;
- **URL(URL context, String url)**: costruttore per specificare degli URL relativi a un URL di partenza;
- **openStream()**: metodo per aprire e scaricare lo stream;
- **getProtocol(), getHost(), getPort(), getFile(), getPath(), getRef(), getQuery()**: metodi per l'interrogazione delle proprietà dell'oggetto URL.

Al momento della sua creazione, un oggetto URL non ha ancora effettuato nessuna operazione di connessione remota.



Un oggetto della classe **URLConnection** rappresenta una connessione a un URL.

- La connessione avviene tramite il metodo **URLConnection openConnection()**: l'oggetto **URLConnection** restituito rappresenta una connessione a un URL;
- **getInputStream()**, **getOutputStream()**: tramite questi metodi, ottenuta la connessione, si può dare inizio a una comunicazione ottenendo gli stream in ingresso e in uscita;
- **getContentLength()**, **getContentType()**, **getExpiration()**, **getHeaderField(String name)**: metodi per l'interrogazione delle proprietà dell'oggetto **URLConnection**;
- **setRequestProperty(String key, String property)**: metodo per impostare i campi della richiesta da inviare;
- **setReadTimeout(int timeout)**, **getReadTimeout()**: metodi per manipolare il timeout in fase di lettura.



Se non ci interessa utilizzare le funzionalità della classe **URLConnection** ma vogliamo solo l'**InputStream** associato (cioè il contenuto dell'URL), allora possiamo usare direttamente il metodo della classe **URL openStream()**.



### verifica le tue conoscenze

- 1 Cos'è un processo Server?
- 2 Cos'è un processo Client?
- 3 Cosa sono i socket?
- 4 Quali classi mette a disposizione il Java per programmare i socket per TCP?
- 5 Quali classi mette a disposizione il Java per programmare i socket per UDP?
- 6 Quali classi mette a disposizione il Java per accedere a risorse web con HTTP?