LPR Corso A
a.a. 2005/2006

Lezione n. 2

- Gestione indirizzi IP
- La classe JAVA InetAddress
- Thread Pooling

INFORMAZIONI UTILI

- La frequenza non è obbligatoria

Materiale Didattico:

- Lucidi delle lezioni

- Elliotte Rusty Harold *Java Network Programming* (in inglese)
  Terza edizione aggiornata, disponibile in libreria. Fa riferimento a JAVA 5.0
  Prima edizione tradotta in italiano, non disponibile + obsoleta

- Semeria *Understanding IP Addressing: Everything You Ever Wanted to Know*.
  Download dalla mia pagina web.

- M.L.Liu *Distributed Computing, Principles and Applications* Non disponibile in biblioteca. In libreria solo su ordinazione e si prevedono tempi lunghi

PROGRAMMAZIONE DI RETE: INTRODUZIONE

Programmazione di rete: per poter collaborare alla realizzazione di un task, due o più *processi* in esecuzione su *hosts diversi*, distribuiti sulla rete devono *comunicare,* ovvero scambiarsi informazioni

Comunicazione = Utilizza protocolli (= insieme di regole che i partners della comunicazione devono seguire per poter comunicare) ben definiti

Alcuni protocolli utilizzati in INTERNET:

- TCP (Trasmission Control Protocol) un protocollo connection-oriented
- UDP (User Datagram Protocol) protocollo connectionless

PROGRAMMAZIONE DI RETE: INTRODUZIONE

Identificare un processo con cui si vuole comunicare

$\Rightarrow$

Occorre identificare

- la rete all'interno della quale si trova l'host su cui e' in esecuzione il processo
- l'host all'interno della rete
- il processo in esecuzione sull'host

Identificazione dell'host = defnita dal protocollo IP (Internet Protocol)
Facciamo riferimento ad IPv4 (IP versione 4). IPv6 (versione 6) si basa sugli stessi principi

Identificazione del processo = utilizza il concetto di *porta*

Porta = Intero da 0 a 65535

INDIRIZZAMENTO DEGLI HOSTS

Materiale da studiare: Harold, capitolo 2 + rapporto Semeria

Ipotesi semplificativa: Un host ha una unica interfaccia di rete.

- Ogni host su una rete IPv4 è identificato da un numero rappresentato su 4 bytes = *indirizzo IP dell'host* (assegnato da IANA).

  (se un host, ad esempio un router, presenta più di una interfaccia sulla rete ⇒ più indirizzi IP per lo stesso host, uno per ogni interfaccia)

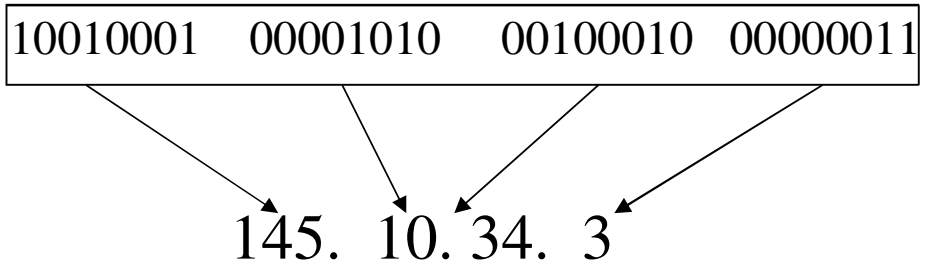
- indirizzi disponibili in IPv4, $2^{32} = 4.294.967.296$ (specifica IPV4 Settembre 1981)

- alcuni indirizzi sono *riservati:* ad esempio indirizzi di *loopback,* individuano il computer su cui l'applicazione è in esecuzione

- indirizzi IP del mittente e del destinatario inseriti nel pacchetto IP

INDIRIZZAMENTO DEGLI HOSTS

- *Esempio:*



Ognuno dei 4 *bytes,* viene interpretato come un numero decimale *senza segno*

$\Rightarrow$

Ogni valore varia da 0 a 255 (massimo numero rappresentabile con 8 bits)

- Evoluzione: in IPv6 indirizzo IP comprende 16 bytes (supportato a partire da JAVA 1.4)

INDIRIZZAMENTO DEGLI HOSTS: ALLOCAZIONE DINAMICA INDIRIZZI IP

Assegnazione degli indirizzi IP:

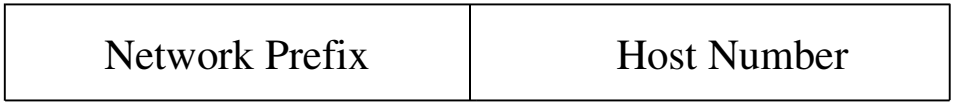
- *Statica:* ad un host viene assegnato un indirizzo IP fisso (utilizzata ad esempio per i servers)
- *Dinamica:* l'indirizzo IP di un host può variare *dinamicamente*.

Assegnazione dinamica:

- utilizzata per hosts che si collegano in modo temporaneo alla rete (es collegamento telefonico). Viene assegnato un nuovo indirizzo IP al momento del collegamento
- utilizza *servers DHCP* (Dynamic Host Configuaration Protocol)
- dal punto di vista del programmatore di rete: tenere presente che l'indirizzo IP di un host può cambiare in relazione ad esecuzioni diverse della stessa applicazione, ed anche durante una stessa esecuzione della applicazione

STRUTTURA INDIRIZZI IP: CLASSFULL ADDRESSING

- *Classfull Addressing:* gerarchia a due livelli



- *Network Prefix* : identifica la rete in cui si trova l'host
- *Host Number* : identifica il perticolare host su quella rete

tutti gli host di una rete presentano lo stesso prefisso

due hosts in reti diversi possono avere lo stesso host number

è possibile specificare network prefixes di lunghezza diversa, in questo modo si possono individuare reti di dimensioni diverse (maggiore la dimensione del network prefix, minore la il numero di hosts sulla rete)

STRUTTURA INDIRIZZI IP: CLASSFULL ADDRESSING

- IPv4 definisce 5 classi di indirizzi IP (A,B,C,D,E)

  - Classi A,B,C caratterizzate da diverse lunghezze dei network prefixes (quindi da diverso numero di hosts per rete)..
  - Classe D utilizzata per *indirizzi di multicast* (la vedremo meglio in seguito).
  - Classe E riservata

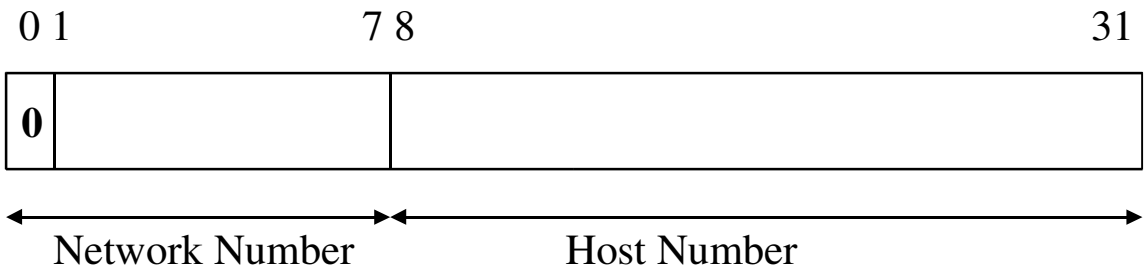
- La classe determina il confine tra il network prefix e l'host number

- Ogni classe è individuata dalla *configurazione dei primi bits* dell'indirizzo IP
  (esempio indirizzi in classe A hanno il primo bit dell'indirizzo =0)

  $\Rightarrow$

  questo semplifica le operazioni effettuate dal router

STRUTTURA INDIRIZZI IP: CLASSFULL ADDRESSING



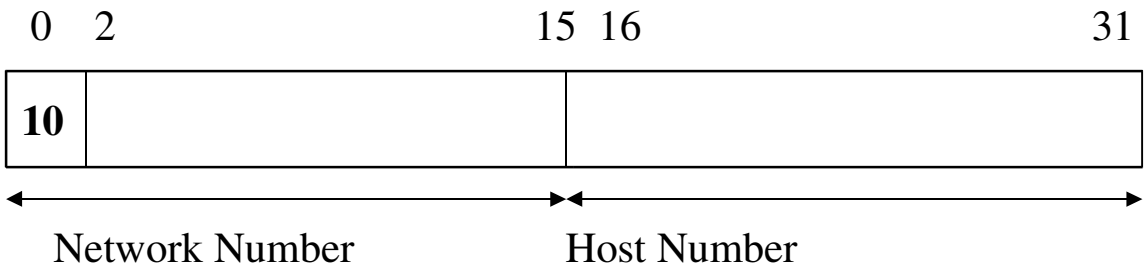
CLASSE A

• Numero di reti = $2^7 - 2$
(0.0.0.0 riservato, 127.xxx.xxx.xxx *indirizzi di loopback)* = indirizzo che individua il *computer locale* su cui è in esecuzione l'applicazione acui corrisponde il nome *localhost*)

•Numero di hosts = $2^{24} - 2$ (circa 16 milioni di hosts, configurazioni di tutti 0 e tutti 1 sono riservate, tutti 1 = broadcast sulla rete)

•Indirizzi utilizzabili: da 1.xxx.xxx.xxx a 126.xxx.xxx.xxx

STRUTTURA INDIRIZZI IP: CLASSFULL ADDRESSING



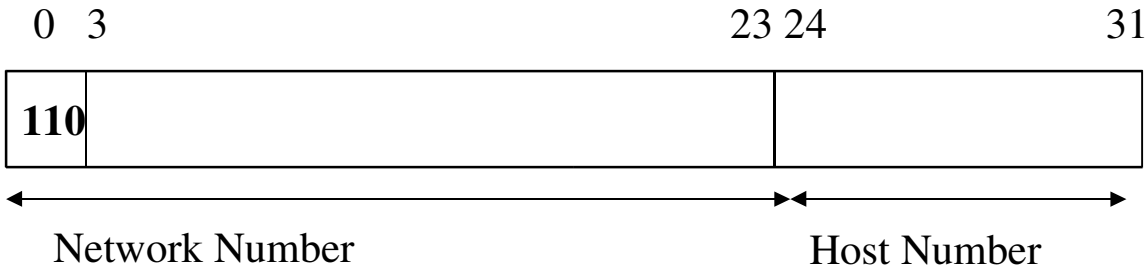
CLASSE B

• Numero di reti = 2^{14} (circa 16.000)

•Numero di hosts = $2^{16} - 2$ (circa 65000 hosts per rete, configurazioni di tutti 0 e tutti 1 sono riservate)

•Indirizzi utilizzabili = da 128.0.xxx.xxx a 191.255.xxx.xxx

STRUTTURA INDIRIZZI IP: CLASSFULL ADDRESSING



CLASSE C

• Numero di reti = 2^{21}

•Numero di hosts = $2^{8} - 2$ (254 hosts, configurazioni di tutti 0 e tutti 1 sono riservate.)

•Indirizzi utilizzabili: 192.0.0.xxx a 233.255.255.xxx

CLASSFULL ADDRESSING: VANTAGGI E SVANTAGGI

- *Vantaggi:*
  semplicità di implementazione dei routers

- *Svantaggi*
  flessibilità limitata. Poche classi predeterminate

  $\Rightarrow$

  allocazione poco efficiente dello spazio degli indirizzi.
  Esempio: una rete include 500 hosts. Non può utilizzare un indirizzo in classe C, perché questo al massimo consente 254 hosts diversi nella rete.
  Utilizzazione di un indirizzo in classe B $\Rightarrow$ spreco di molti indirizzi IP. Soluzione possibile: definire due reti utilizzando due indirizzi in classe C.

- Alternativa: *Classless Addressing* o *Subnetting* (discusso nelle lezioni successive)

INDIRIZZI IP E NOMI DI DOMINIO

- *Problema:* gli indirizzi IP semplificano l'eleborazione effettuata dai routers, ma poco leggibili per gli utenti della rete

- *Soluzione:* assegnare un *nome simbolico unico* ad ogni host della rete
  - si utilizza uno spazio *di nomi gerarchico*

    esempio: fujih0.cli.di.unipi.it (host fuji presente nell'aula H alla postazione 0, nel dominio cli.di.unipi.it)
  - livelli della gerarchia separati dal punto.
  - nomi interpretati da destra a sinistra (diverso dalle gerarchia di LINUX)
  - alla radice della gerarchia:

    un dominio per ogni paesi + *big six* (edu,com,gov,mil,org,net)

- Corrispondenza tra nomi ed indirizzi IP gestita da un servizio di nomi

  DNS = Domain Name System

INDIRIZZAMENTO A LIVELLO DI PROCESSI

- Su ogni host possono essere attivi contemporaneamente più *servizi* (es: e-mail, ftp, http,…)

- Ogni servizio viene incapsulato in un diverso processo

- L'indirizzamento di un processo avviene mediante il concetto di *porta*

- Porta = intero compreso tra 1 e 65535. Non *è un dispositivo fisico*, ma una *astrazione* per individuare i singoli servizi (processi).

- Porte comprese tra 1 e 1023 riservati per particolari servizi. Linux :solo i programmi in esecuzione su root possono ricevere dati da queste porte. Chiunque può inviare dati a queste porte.

  Esempio: porta 7 *echo*
  porta 22 *ssh*
  porta 80 *HTTP*

- In LINUX: controllare il file /etc/hosts

JAVA: LA CLASSE INETADDRESS

Materiale da studiare: Harold, capitolo 6

Classe JAVA. NET.InetAddress (importare JAVA.NET).

Gli oggetti di questa classe sono strutture con due campi

- hostname = una stringa che rappresenta il nome simbolico di un host
- indirizzo IP = un intero che rappresenta l' indirizzo IP dell'host

La classe InetAddress:

- non definisce costruttori
- Fornisce tre *metodi statici* per costruire oggetti di tipo InetAddress
  - *public static* InetAddress.*getByName* (*String* hostnme) *throws UnKnownHostException*
  - *public static* InetAddress.*getAllByName*(*String* hostname) *throws UnKnownHostException*
  - *public static* InetAddresss *getLocalHost*(*) throws UnKnownHostException*

JAVA: LA CLASSE INETADDRESS

public static InetAddress *getByName* (*String* hostname)
throws UnKnownHostException

- Cerca l'indirizzo IP corrispondente all'host di nome hostname e restituisce un oggetto di tipo InetAddress = nome simbolico dell'host + l'indirizzo IP corrispondente

- In generale richiede una interrogazione del DNS per risolvere il nome dell'host
  ⇒ il computer su cui è in esecuzione l'applicazione deve essere connesso in rete

- Può sollevare una eccezione se non riesce a risolvere il nome dell'host (ricordarsi di gestire la eccezione!)

Esempio:

```
try  { InetAddress address = InetAddress.getByName ("fujim0.cli.di.unipi.it");
       System.out.println (address);
       }
catch (UnKnownHostException e) {System.out.println ("Non riesco a risolvere il nome"); }
```

JAVA: LA CLASSE INETADDRESS

public static InetAddress [ ] *getAllByName* (*String* hostname)
throws UnKnownHostException

utilizzata nel caso di hosts che posseggano piu indirizzi (esempio web servers)

public static InetAddress *getLocalHost* ()
throws UnKnownHostException

per reperire nome simbolico ed indirizzo IP del computer su cui è in esecuzione l'applicazione

Getter Methods = consentono di reperire i campi di un oggetto di tipo InetAddress (non effettuano collegamenti con il DNS $\Rightarrow$ non sollevano eccezioni)

public String getHostName ( )
public byte [] getAddress ( )
public String getHostAddress ( )

JAVA: LA CLASSE INETADDRESS

public static InetAddress *getByName* (*String* hostname)
throws UnKnownHostException

- Il valore di hostname può essere l'indirizzo IP (una stringa corrispondente alla dotted form)

- in questo caso:

  - la getByName *non contatta* il DNS
  - il nome dell'host prende come valore l'indirizzo IP passato come parametro
  - il DNS viene contattato solo quando viene *richiesto esplicitamente* il nome dell'host tramite il metodo getHostName( )
  - la getHostName non solleva eccezione, se non riesce a risolvere l'indirizzo IP.

JAVA: LA CLASSE INETADDRESS

- accesso al DNS = operazione potenzialmente molto costosa
- la classe InetAddress effettua il caching dei nomi risolti. Se creo un InetAddress per lo stesso host, il nome viene risolto con i dati nella cache.
- se il risultato della ricerca è negativo (esempio timeout sulla ricezione della risposta dal DNS) la ricerca non viene ripetuta
- Problemi: indirizzi dinamici, tentativi ripetuti di accesso al DNS.
- Soluzione: specificare il numero di secondi in cui una entrata nella cache rimane valida

  networkaddress.cache.ttl

  networkaddress.cache.negative.ttl

  -1 = never expires

LA CLASSE INETADDRESS: ESEMPIO DI UTILIZZO

- implementazione in JAVA della utility UNIX *nslookup*
- *nslookup*
  - consente di tradurre nomi di hosts in indirizzi IP e viceversa
  - i valori da tradurre possono essere forniti in modo interattivo oppure da linea di comando
  - si entra in modalità interattiva se non si forniscono parametri da linea di comando
  - consente anche funzioni più complesse (vedere LINUX)

LA CLASSE INETADDRESS: ESEMPIO DI UTILIZZO

```
import java.net.*;
import java.io.*;

public class HostLookUp {
      public static void main (String [] args} {
            if (args.length > 0) {
               for (int i=0; i<args.length; i++) {
                        System.out.println (lookup(args[i])) ;
                                                }
               }
            else {/* modalita' interattiva*/….. }
```

LA CLASSE INETADDRESS: ESEMPIO DI UTILIZZO

```
private static String lookup(String host) {

    InetAddress node;
    try { node =InetAddress.getByName(host)
            }
    catch (UnknownHostException e) {return "non ho trovato l'host" }
    if  (ishostname(host)) {return node.getHostAddress( );}

            else{ return node.getHostName ( );
            }
    }
```

LA CLASSE INETADDRESS: ESEMPIO DI UTILIZZO

```
private static boolean isHostName (String host)

        {char[ ] ca = host.toCharArray();
                for (int i = 0; i<ca.length; i++)
                { if(!character.isDigit(ca[i])) {
                        if (ca[i] !='.') return true;
                        }
                }
        return false;
        }
        }
```

THREAD POOLING

- i threads sono stati introdotti per migliorare le prestazioni, soprattutto in applicazioni I/O bound, ma…

- l'attivazione di un *thread*, la sua t*erminazione, il cambio di contesto tra un threads* introduce un overhead

  ⇒

  l'attivazione di un numero elevato di threads può annullare i benefici dell'esecuzione concorrente, specie in applicazioni *CPU bound*

- *Thread Pooling (Thread Farm)* : per limitare il numero di threads attivati

  ⇒

  – dividere la computazione in tasks
  – attivare un numero limitato di threads assegnare ad ogni thread l'esecuzione di un insieme di tasks.
  – *riusare* lo stesso thread per l'esecuzione di più tasks

THREAD POOLING: IMPLEMENTAZIONE

- Definizione di un task

  esempio: effettuare la compressione di un file, gestire la connessione di rete con un client,….

- Creazione di un *Task Pool* = struttura dati condivisa tra i threads in cui vengono inseriti i tasks da eseguire

  esempio: riferimento al file da comprimere, riferimento al socket su cui avviene la connessione con il client

- Attivazione di k thread. Un thread (Produttore) inserisce i tasks nel Task Pool, i restanti k-1 threads (consumatori) prelevano i tasks dal pool e li eseguono

- Sincronizzazione di tipo Produttore/Consumetore sul Pool Condiviso

- Problema: eventuale condizione di terminazione dei threads

ESERCIZIO
WebLookUp: ELABORAZIONE DI INDIRIZZI IP

Un web server registra l'indirizzo IP di ogni host che richiede un collegamento in un file *LogFile.*

Scrivere un programma JAVA, *WebLookUp*, che elabora *LogFile* e visualizza, per ogni indirizzo IP contenuto in tale file, il nome dell'host corrispondente.

WebLookUp deve effettuare una interrogazione al DNS per ogni indirizzo IP letto da LogFile. L'esecuzione sequenziale di tali interrogazioni al DNS può provocare una notevole degradazione delle prestazioni del programma.

Per ridurre tale degradazione si utilizzi la tecnica del *thread pooling*. Attivare un thread (produttore) che legga gli indirizzi IP dal *LogFile* e li inserisca nel pool *IPPool.*

Attivare quindi un insieme di threads che prelevino gli indirizzi da *IPPool* e ed effettuino le interrogazioni al DNS in modo concorrente.

Il programma *deve terminare* quando tutti gli indirizzi IP contenuti nel file sono stati elaborati.