

# Intermediario

martedì 6 settembre 2022 18:29

Si consideri un sistema di rete composto da:

- 1 applicazione **Intermediario** K applicazioni **Venditore**, N applicazioni **Cliente**.

- 1) L'Intermediario accetta connessioni tramite socket **TCP** sulla porta **2345**.
- 2) Un **Cliente** si connette all'Intermediario per sottomettere una **Richiesta** e per ottenere una **Risposta** (sullo stesso socket).
- 3) Una **Richiesta** è un oggetto **<idProdotto, quantità>** e viene inoltrato a tutti i venditori.
- 4) I **Venditori** ricevono le richieste sulla porta **UDP 6789** e possono rispondere con un messaggio **UDP**. Ogni venditore può rispondere alla Richiesta con una **Risposta** (una stringa composta da "**idProdotto,quantità,prezzoTotale,idVenditore**") entro un tempo massimo di 1 minuto. **socket.setSoTimeout(6000 ms)**

Trascorso **1 minuto**, l'Intermediario valuta tutte le risposte pervenute e invia indietro al Cliente la Risposta che presenta il prezzoTotale **minimo**. Se non è presente nessuna risposta da parte dei venditori, l'Intermediario invia indietro una Risposta predefinita <"**idProdotto**", "quantità", -1, -1>.

Si realizzino una classe Intermediario ed una classe Cliente che implementano le funzionalità sopra descritte. All'interno del costruttore dell'Intermediario viene passata una lista di oggetti Venditore.

Inoltre, si implementino due main:

- 1) il primo main crea e avvia un Intermediario (nome dell'host: **intermediario.eu**) che riceve le richieste da parte dei Clienti;
- 2) il secondo main crea e avvia un Cliente che inoltra una richiesta di acquisto prodotto all'Intermediario e attende la sua risposta.

Non è richiesta la scrittura della classe Venditore. Si noti che il sistema deve essere in grado di gestire diversi Clienti.

GAIA BERTOLINO

```
public class Intermediario {  
    private static final int serverPort = 2345;  
    Venditore[] vendori;  
    ServerSocket socket;  
    LinkedList<Richiesta> richieste;  
    LinkedList<String> risposte;  
  
    public Intermediario (Venditore[] vendori) {  
        this.vendori = vendori;  
        socket = new ServerSocket(serverPort);  
        richieste = new LinkedList<Richiesta>();  
        risposte = new LinkedList<String>();  
    }
```

```

public static void main(String[] args) {
    try {
        while (true) {
            Socket s = socket.accept();
            socket.setSoTimeout(6000);
            new GestoreClienti(s).start();
            new GestoreVenditori().start();
        }
    } catch (InterruptedException e) {
        if (risposte.size() == 0) {
            Socket s = new Socket(serverPort);
            PrintWriter w = new PrintWriter(s.getOutputStream());
            w.println("idProdotto,quantità,-1,-1");
        }
        else {
            int min = 1000000;
            String winner = "";
            for(String st: risposte) {
                String[] divisa = st.split(",");
                if (divisa[2] < min) {
                    min = divisa[2];
                    winner = st;
                }
            }
            Socket s = new Socket(serverPort);
            PrintWriter w = new PrintWriter(s.getOutputStream());
            w.println(winner);
        }
    }
}

class GestoreVenditori extends Thread {
    private static final int venditoriPort = 6789;
    public void run() {
        DatagramSocket datasocket = new DatagramSocket(venditoriPort);
        byte[] buffer = new byte[512];
        DatagramPacket datapacket = new DatagramPacket(buffer,buffer.length);
        String ris = new String(datapacket.getBytes());
        risposte.add(ris);
    }
}

class GestoreClienti extends Thread {

    Socket s;

```

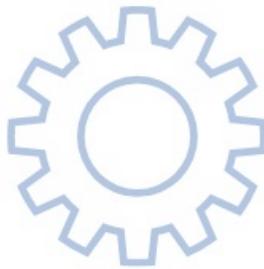
```

private static final int venditoriPort = 6789;

public GestoreClienti(Socket s) {
    this.s = s;
}

public void run() {
    ObjectInputStream obj = new ObjectInputStream(s.getInputStream());
    Richiesta ric = (Richiesta) obj.readObject();
    richieste.add(ric);
    MulticastSocket ds = new MulticastSocket(venditoriPort);
    byte[] buffer = new byte[512];
    buffer = obj.readObject().getBytes();
    DatagramPacket dp = new DatagramPacket(buffer, buffer.length, new
    InetSocketAddress("270.0.0.1"), venditoriPort);
    ds.send(dp);
}
}

```



```
class Cliente {
```

```
    private static final int serverPort = 2345;
```

**APPUNTI DI INGEGNERIA**

**INFORMATICA**

GAIA BERTOLINO

```
    Richiesta r;
```

```
    public Cliente(Richiesta r) {
        this.r = r;
    }
}
```

```
    public static void main(String[] args) {
```

```
        try {
```

```
            Socket s = new Socket(serverPort);
```

```
            ObjectOutputStream obj = new ObjectOutputStream(s.getOutputStream());
```

```
            obj.writeObject(r);
```

```
            ServerSocket socket = new ServerSocket(serverPort);
```

```
            Socket s1 = socket.accept();
```

```
        }
```

```
}
```

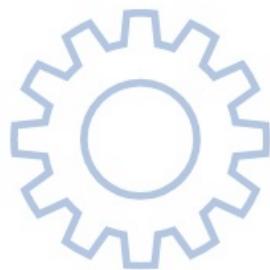
```
}
```

```
class Richiesta {
```

```
    int idProdotto;
```

```
    int quantita;
```

```
public Richiesta (int idProdotto, int quantita) {  
    this.idProdotto = idProdotto;  
    this.quantita = quantita;  
}  
}
```



# APPUNTI DI INGEGNERIA INFORMATICA

GAIA BERTOLINO

# Gare d'appalto

lunedì 5 settembre 2022 17:32

Si vuole realizzare un sistema distribuito su rete, per lo svolgimento di gare d'appalto per l'esecuzione di opere pubbliche. Il sistema è costituito da:

1 applicazione **Giudice** di gara; → statico  
n applicazioni **Partecipanti** (identificate da un intero *id* univoco).

Un ente che intende realizzare un'opera pubblica invia al Giudice di gara una **Richiesta**, contenente la descrizione dell'opera da realizzare e l'importo massimo disponibile per la sua realizzazione.

**Il Giudice invia** gli estremi della Richiesta a tutti i Partecipanti. **MULTICAST**

**Ciascun Partecipante risponde** inviando al Giudice una **Offerta**, contenente l'*id* del Partecipante ed l'*importo* da esso richiesto per la realizzazione dell'opera.

Il Giudice di gara **seleziona** quindi l'Offerta con l'importo richiesto inferiore (a parità di prezzo richiesto, seleziona l'Offerta con *id* del Partecipante inferiore). Questa Offerta, ritenuta vincitrice della gara, viene inviata dal Giudice di gara all'ente richiedente, e, per notifica, a tutti i Partecipanti.

Si richiede di realizzare in Java l'applicazione *Giudice*.

CLASSI :

- Giudice
- Partecipanti
- Offerta

L'applicazione **Giudice** è basata sulle seguenti operazioni:

- Usa la porta **TCP 2000** per la connessione da una applicazione remota (che rappresenta l'ente che intende realizzare l'opera).
- L'applicazione connessa invia un oggetto istanza della classe **Richiesta**. La classe **Richiesta**, che implementa l'interfaccia Serializable, memorizza la descrizione dell'opera e l'importo massimo disponibile per la sua realizzazione.
- Comunica a **tutti** i Partecipanti una stringa con la descrizione dell'opera e l'importo massimo. Per esempio, la stringa "Stadio olimpico di Arcavacata - 1000". La stringa viene inviata ai Partecipanti in broadcast (si usi la porta **3000** e l'indirizzo **230.0.0.1**). I metto Address .getByName()
- Usa la porta **TCP 4000** per ricevere una sequenza di **n** connessioni da parte delle applicazioni Partecipanti (non è richiesto di gestire connessioni multiple).
- **Riceve un oggetto** istanza della classe **Offerta** da ciascuna applicazione Partecipante connessa. La classe **Offerta**, che implementa Serializable, memorizza un intero che rappresenta l'*id* del Partecipante ed un intero che rappresenta l'*importo* richiesto per la realizzazione dell'opera.
- **Seleziona l'Offerta vincitrice**. L'Offerta selezionata viene inviata all'applicazione remota (che rappresenta l'ente richiedente). A tutti i Partecipanti, inoltre, invia in broadcast una stringa contenente l'*id* del vincitore e l'importo da esso richiesto (ad esempio, la stringa "12 - 950" indica che il Partecipante con id 12 ha vinto la gara con un importo richiesto di 950).

La costante **n** è nota all'applicazione Giudice. Si richiede di implementare anche le classi Richiesta e Offerta.

```
package garaDAppalto;
```

```
import java.io.ObjectInputStream;
import java.io.Serializable;
import java.net.DatagramPacket;
import java.net.InetAddress;
import java.net.MulticastSocket;
import java.net.ServerSocket;
```

```

import java.net.Socket;
import java.util.LinkedList;

public class Giudice {

    private final static int entePort = 2000;
    private final static int multicastPort = 3000;
    private final static String multicastAddress = "230.0.0.1";
    private final static int ricezionePort = 4000;
    private static int n;
    private static LinkedList<Offerta> offerte;

    public Giudice(int n) {
        this.offerte = new LinkedList<Offerta>();
        n = n;
    }

    public static void main(String[] args) {
        try {
            // Ricezione opera da un ente
            ServerSocket socket = new ServerSocket(entePort);
            Socket s = socket.accept();
            // Ricezione di un messaggio dall'ente
            // In questo caso ricevo un oggetto quindi ricorro a ObjectInputStream
            ObjectInputStream richiestaArrivata = new ObjectInputStream(s.getInputStream());
            Richiesta richiesta = (Richiesta) richiestaArrivata.readObject();

            String dalnoltrare = richiesta.descrizione + " " + richiesta.importoMassimo;
            // Inoltro multicast
            MulticastSocket server = new MulticastSocket(multicastPort);
            byte[] buffer = new byte[256];
            InetAddress group = InetAddress.getByName(multicastAddress);
            buffer = dalnoltrare.getBytes();
            DatagramPacket packet = new DatagramPacket(buffer, buffer.length, group, multicastPort);

            server.close();

            int i = 0;
            ServerSocket socketRic = new ServerSocket(ricezionePort);
            while (i < n) {
                Socket sRic = socketRic.accept();
                ObjectInputStream obj = new ObjectInputStream(sRic.getInputStream());
                Offerta offerta = (Offerta) obj.readObject();
                offerte.add(offerta);
                i++;
                sRic.close();
            }
        }
    }
}

```

```

        Offerta winner = offertaVincente(offerte);

    } catch(Exception e) {}

}

public static Offerta offertaVincente(LinkedList<Offerta> offerte) {
    int min = 0;
    Offerta win = new Offerta(-1);
    for(Offerta o: offerte) {
        if (o.importo < min) {
            min = o.importo;
            win = o;
        }
    }
    return win;
}

}

class Partecipanti {

}

class Offerta implements Serializable {
    static int ID;
    int importo;

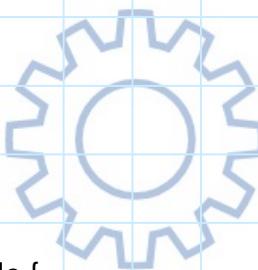
    public Offerta (int importo) {
        this.ID += 1;
        this.importo = importo;
    }
}

class Richiesta implements Serializable {
    String descrizione;
    int importoMassimo;

    public Richiesta(String descrizione, int importoMassimo) {
        this.descrizione = descrizione;
        this.importoMassimo = importoMassimo;
    }
}

```

7D



## APPUNTI DI INGEGNERIA INFORMATICA

GAIA BERTOLINO

## Domanda 1

Si descrive come funziona il protocollo go-back-N.

## Domanda 2

Si descrivono le principali differenze fra trasferimento intra-AS e inter-AS.

## Esercizio 1

Si realizzano un Web Service che permette di ottenere informazioni su un insieme di negozi. Ogni negozio è identificato dalla parola chiave (*String*) ed è caratterizzato dalla provincia in cui si trova. In particolare il servizio espone:

- un metodo che, dato il nome di una provincia e una data "dataFine", restituisce la parola chiave di tutti gli incassi fino a quella provincia che ha incassato di più fino a dataFine (sommatoria di tutti gli incassi fino a dataFine);
- un metodo che, data una parola chiave e una data, restituisce un array contenente le date in cui il negozio con quella parola chiave ha incassato almeno quella cifra (array vuoto se non viene generato alcun risultato);

Non è richiesto di gestire l'inservimento degli incassi dei negozi (si può assumere che gli incassi siano già memorizzati).

Si implementa in Java una classe che implementa il servizio, si chiede di utilizzare le opportunità strutturali che il permettono di memorizzare gli incassi giornalieri di ogni negozio. Si stimano le esecuzioni delle query.

## Allegato A) wsdl

```
<wsdl:definitions targetNamespace="http://www.example.com/wsd/CatenaService">
<schema targetNamespace="http://www.example.com/wsd/CatenaService">
<complexType name="Data">
<sequence>
<element name="Giorno" type="xsd:string"/>
<element name="Mese" type="xsd:int"/>
<element name="Anno" type="xsd:int"/>
</sequence>
<complexType name="ArrayListData">
<sequence>
<element name="item" type="Data" minOccurs="0" maxOccurs="unbounded"/>
</sequence>
</complexType>
<message name="GetGuadagnoDaA">
<part name="in1" type="xsd:string"/>
<part name="in2" type="xsd:int"/>
<part name="out1" type="xsd:string"/>
</message>
<message name="DateGuadagnoRequest">
<part name="in1" type="xsd:string"/>
<part name="in2" type="xsd:int"/>
</message>
<message name="DateGuadagnoResponse">
<part name="out1" type="ArrayListData"/>
</message>
<portType name="CatenaServicePort">
<operation name="GetGuadagnoDaA">
<input message="tns:GetGuadagnoDaARequest" name="GetGuadagnoDaARequest"/>
<output message="tns:GetGuadagnoDaAResponse" name="GetGuadagnoDaAResponse"/>
</operation>
<operation name="DateGuadagno">
<input message="tns:DateGuadagnoRequest" name="DateGuadagnoRequest"/>
<output message="tns:DateGuadagnoResponse" name="DateGuadagnoResponse"/>
</operation>
</portType>
<service name="CatenaService">
<port name="CatenaServicePort" binding="tns:CatenaServicePort"/>
<binding name="CatenaServicePort" type="wsdl:rpc">
<parameter name="parameterOrder" value="1,2" />
<input message="tns:GetGuadagnoDaARequest" name="GetGuadagnoDaARequest"/>
<output message="tns:GetGuadagnoDaAResponse" name="GetGuadagnoDaAResponse"/>
</binding>
</service>
</wsdl:definitions>
```

**DIMES** Esame di Reti di Calcolatori del Corso di Laurea in Ingegneria Informatica  
Prova scritta del 20 gennaio 2021 - Primo Parte - Durata: 50 minuti

```
2 INPUT < DATA
1 OUTPUT - STRING
2 INPUT < STRING
1 OUTPUT - DOUBLE
1 OUTPUT - ARRAYOFDATA
```

## public class CatenaService {

```
HashMap< String , LinkedList< Negozio >>=db1;
HashMap< String , Negozio > = db2;
```

## public String GuadagnoDaA ( Data dataFine, String provincia ) {

```
LinkedList< Negozio > negozi = db1.getValue(provincia);
int max = 0;
```

```
Negozio wiunex = null;
```

```
for ( Negozio u : negozi ) {
    if ( u.getIncassi().getMaxValue() > max ) {
        max = u.getIncassi().getMaxValue();
        wiunex = u;
    }
}
```

```
return wiunex.piva;
```

## public ArrayList&lt; Data &gt; DateGuadagno ( String piva, double cifra ) {

```
ArrayList< Data > ris = new ArrayList< Data >();
```

```
Negozio u = db2.getValue(piva);
```

```
for ( Entry< Data, Integer > e : u.getIncassi().entrySet() ) {
    if ( e.getKey() >= cifra )
        ris.add( e.getKey() );
}
```

```
return ris;
```

## class Data implements Serializable {

```
int Giorno;
```

```
int Mese;
```

```
int Anno;
```

## public Data ( int Giorno, int Mese, int Anno ) {

```
this.Giorno = Giorno;
```

```
this.Mese = Mese;
```

```
this.Anno = Anno;
```

```
}
```

## class ArrayOfData implements Serializable {

```
Data[] item = {};
```

## public ArrayList&lt; Data &gt; ( Data[] item ) {

```
this.item = item;
```

```
}
```

## public void add ( Data data ) {

```
Data[] temp = item.copyOf(item.length+1);
for ( int i = 0; i+1 < item.length;
      temp[i] = item[i];
      item = temp;
```

## public ArrayList&lt; Data &gt; () {

```
item = {};
```

```
}
```

## class Negozio implements Serializable {

```
String piva;
```

```
String provincia;
```

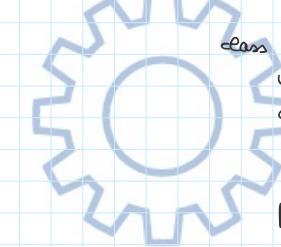
```
HashMap< Data, Integer > incassi;
```

## public Negozio ( String piva, String provincia ) {

```
this.piva = piva;
```

```
this.provincia = provincia;
```

```
this.incassi = incassi;
```



**APPUNTI DI INFORMATICA**  
**GAIA BERTOLINO**

03 Mar  
2022**DIMES**Esame di Reti di Calcolatori del Corso di Laurea in Ingegneria Informatica  
Prova scritta del 29 marzo 2022 – Prima Parte – Durata: 50 minuti**Domanda 1**

Si descriva a cosa serve e come funziona il controllo di flusso nel protocollo TCP.

**Domanda 2**

Si descriva a cosa serve e come funziona il protocollo BitTorrent.

**Esercizio**

Si realizzi un *Web Service* che permette di ottenere informazioni sui prezzi dei distributori di benzina delle diverse regioni. Ogni *distributore* è identificato da una *partita iva* ed è caratterizzato dalla *regione di appartenenza*, dalla *regione sociale*, dal *prezzo del diesel* e della *benzina*.

In particolare, il servizio espone:

1. un metodo che, dato il *nome di una regione*, restituisce il *distributore* con il miglior prezzo della benzina.
2. un metodo che restituisce il *nome della regione che*, considerando tutti i distributori presenti sul suo territorio, ha il *minore prezzo medio* del diesel.

In accordo a quanto specificato nel WSDL allegato, si implementi in Java una classe che implementa il servizio.

**Allegato all'esercizio**

```
<wsdl:definitions targetNamespace="http://www.examples.com/wsdl/DistributoriBenzinaService">
    <wsdl:types>
        <xsd:schema targetNamespace="http://www.examples.com/wsdl/DistributoriBenzinaService">
            <xsd:import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
            <complexType name="Distributore">
                <sequence>
                    <element name="PartitaIva" type="xsd:string"/>
                    <element name="RagioneSociale" type="xsd:string"/>
                    <element name="Regione" type="xsd:string"/>
                    <element name="PrezzoDiesel" type="xsd:double"/>
                    <element name="PrezzoBenzina" type="xsd:double"/>
                </sequence>
            </complexType>
        </xsd:schema>
    </wsdl:types>
    <wsdl:message name="MinPrezzoBenzinaRequest">
        <wsdl:part name="In" type="xsd:string"/>
    </wsdl:message>
    <wsdl:message name="MinPrezzoBenzinaResponse">
        <wsdl:part name="In" type="Distributore"/>
    </wsdl:message>
    <wsdl:message name="RegioneMinMediaDieselRequest">
        <wsdl:part name="In" type="xsd:string"/>
    </wsdl:message>
    <wsdl:message name="RegioneMinMediaDieselResponse">
        <wsdl:part name="In" type="xsd:string"/>
    </wsdl:message>
    <wsdl:portType name="DistributoriBenzinaService">
        <wsdl:operation name="MinPrezzoDiesel" parameterOrder="in0">
            <wsdl:input message="MinPrezzoBenzinaRequest" name="MinPrezzoBenzinaRequest" />
            <wsdl:output message="MinPrezzoBenzinaResponse" name="MinPrezzoBenzinaResponse" />
        </wsdl:operation>
        <wsdl:operation name="RegioneMinMediaDiesel" parameterOrder="in0">
            <wsdl:input message="RegioneMinMediaDieselRequest" name="RegioneMinMediaDieselRequest" />
            <wsdl:output message="RegioneMinMediaDieselResponse" name="RegioneMinMediaDieselResponse" />
        </wsdl:operation>
    </wsdl:portType>...</wsdl:definitions>
```

LINKED LIST → .size()

Federazione: *Eufzy < type >* e *i bushuep · eufzySet()*

package esami;

```
import java.io.Serializable;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.Map.Entry;

public class DistributoriBenzinaService {
    HashMap<String, LinkedList<Distributore>> db;

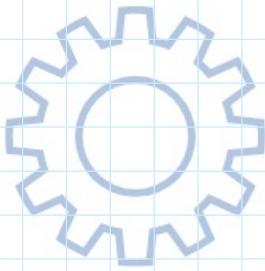
    public DistributoriBenzinaService() {
        this.db = new HashMap<String, LinkedList<Distributore>>();
    }

    // Data la regione da il distributore col prezzo minore
    public Distributore MinPrezzoDiesel(String regione) {
        LinkedList<Distributore> selected = db.get(regione);
        double min = 10000;
        Distributore winner = null;
        for (Distributore d: selected) {
            if (d.PrezzoDiesel < min) {
                min = d.PrezzoDiesel;
                winner = d;
            }
        }
        return winner;
    }

    // Da il nome della regione col prezzo medio del diesel minore
    public String RegioneMinMediaDiesel() {
        int min = 100000;
        int somma = 0;
        String region = "";
        for (Entry<String, LinkedList<Distributore>> e : db.entrySet()) {
            somma = 0;
            for (Distributore d: e.getValue()) {
                somma += d.PrezzoDiesel;
            }
            somma = somma/(e.getValue().size());
            if (somma < min) {
                min = somma;
                region = e.getKey();
            }
        }
        return region;
    }

    class Distributore implements Serializable {
        String PartitaIva;
        String RagioneSociale;
        String Regione;
        double PrezzoDiesel;
        double PrezzoBenzina;

        public Distributore(String PartitaIva, String RagioneSociale, String Regione,
                           double PrezzoDiesel, double PrezzoBenzina) {
            this.PartitaIva=PartitaIva;
            this.Regione = Regione;
            this.RagioneSociale = RagioneSociale;
            this.PrezzoBenzina=PrezzoBenzina;
            this.PrezzoDiesel=PrezzoDiesel;
        }
    }
}
```



# APPUNTI DI INGEGNERIA INFORMATICA

GAIA BERTOLINO

**DIMES** Esame di Reti di Calcolatori del Corso di Laurea in Ingegneria Informatica  
Prova scritta del 7 novembre 2019 – Durata: 2 ore e 30 minuti!

Cognome e Nome:	Matricola:
-----------------	------------

**Esercizio 4 (fino a 5 punti)**

Si realizzzi un Web Service che permette di ottenere alcune informazioni sugli esami sostenuti dagli studenti di un corso di laurea. In particolare, il servizio esponde:

- un metodo che, data una matricola, restituisce l'elenco degli esami sostenuti da quello studente (per ogni esame viene restituito l'ID dell'esame).
- un metodo che, dati l'ID di un esame (String) e un voto, restituisce la lista degli studenti che hanno conseguito quel voto in quell'esame.

come specificato nel file WSDL allegato. Si implementi in Java una classe che implementa il servizio. Scagliare le opportune strutture che permettono di salvare i prodotti venduti. Si minimizzi l'esecuzione delle query.

**Allegato all'esercizio 4**

```
<wsdl:definitions targetNamespace="http://www.example.com/wstl/EsamiService">
<wsdl:types>
<wsdl:import namespace="http://schemas.xmlsoap.org/soap/encoding/" />
<wsdl:complexType name="Studente">
<element name="Matricola" type="xsd:string"/>
<element name="Nome" type="xsd:string"/>
<element name="Cognome" type="xsd:string"/>
</wsdl:complexType>
<wsdl:complexType name="EsamiListType">
<element name="esameId" type="xsd:int"/>
<element name="voto" type="xsd:int"/>
</wsdl:complexType>
<wsdl:complexType name="StudentiListType">
<element name="studenti" type="tns:Studente"/>
</wsdl:complexType>
<wsdl:portType name="EsamiService">
<wsdl:operation name="EsamiStudente" parameterOrder="1,2">
<wsdl:input message="tns:EsamiStudenteRequest"/>
<wsdl:output message="tns:EsamiStudenteResponse"/>
</wsdl:operation>
<wsdl:operation name="StudentiPerEsame" parameterOrder="1,2">
<wsdl:input message="tns:StudentiPerEsameRequest"/>
<wsdl:output message="tns:StudentiPerEsameResponse"/>
</wsdl:operation>
</wsdl:portType>
<wsdl:binding name="EsamiServiceBinding" type="tns:EsamiService">
<wsdl:operation name="EsamiStudente">
<wsdl:input portType="tns:EsamiService" message="tns:EsamiStudenteRequest"/>
<wsdl:output portType="tns:EsamiService" message="tns:EsamiStudenteResponse"/>
</wsdl:operation>
<wsdl:operation name="StudentiPerEsame">
<wsdl:input portType="tns:EsamiService" message="tns:StudentiPerEsameRequest"/>
<wsdl:output portType="tns:EsamiService" message="tns:StudentiPerEsameResponse"/>
</wsdl:operation>
</wsdl:binding>
<wsdl:service name="EsamiService">
<wsdl:port name="EsamiServicePort" binding="tns:EsamiServiceBinding">
<wsdl:address location="http://localhost:8080/EsamiService/EsamiServicePort"/>
</wsdl:port>
</wsdl:service>

```

1 INPUT                  1 OUTPUT  
2. INPUT                  1 OUTPUT

package esami;

```
import java.io.Serializable;
import java.util.HashMap;
```

```
public class EsamiService {
```

```
    HashMap<String, EsamiListType> db1;
```

```
    HashMap<String, HashMap<Integer, StudentiListType>> db2;
```

```
    public EsamiService() {
```

```
        db1 = new HashMap<String, EsamiListType>();
```

```
        db2 = new HashMap<String, HashMap<Integer, StudentiListType>>();
```

```
}
```

```
// In base alla matricola restituisce l'id degli esami sostenuti dallo studente
```

```
public EsamiListType EsamiStudente(String matricola) {
```

```
    return db1.get(matricola);
```

```
}
```

```
// In base all'ID esame e al voto, restituisce la lista di studenti con quel voto all'esame
```

```
public StudentiListType StudentiPerEsame(String IDesame, int voto) {
```

```
    return db2.get(IDesame).get(voto);
```

```
}
```

```
class Studente implements Serializable {
```

```
    private String Matricola;
```

```
    private String Nome;
```

```
    private String Cognome;
```

```
    public Studente( String Matricola, String Nome, String Cognome) {
```

```
        this.Matricola = Matricola;
```

```
        this.Nome = Nome;
```

```
        this.Cognome = Cognome;
```

```
}
```

```
}
```

```
class EsamiListType implements Serializable {
```

```
    // IDESAME VOTO
```

```
    HashMap<String, Integer> esami;
```

```
    public EsamiListType() {
```

```
        this.esami = new HashMap<String, Integer>();
```

```
}
```

```
class StudentiListType implements Serializable {
```

```
    // MATRICOLA STUDENTE
```

```
    HashMap<String, Studente> studenti;
```

```
    public StudentiListType() {
```

```
        this.studenti = new HashMap<String, Studente>();
```

```
}
```

```
}
```

**APPUNTI DI INFORMATICA**  
**GAIA BERTOLI**

Reti di calcolatori Pagina 12

# Prova d'esame svolta

martedì 30 agosto 2022 20:11

## PROVA D'ESAME

Quando da un solo server c'è una base di dati e più azioni / modifiche allora bisogna prevedere più thread per ogni client e per ogni azione.

### Esercizio 1: IndexServer

Si vuole realizzare un sistema di rete per la memorizzazione e la ricerca di file. Il sistema è composto da tre gruppi di nodi:

- 3 StorageServer utilizzati per memorizzare i file.
- 1 IndexServer utilizzato per indicizzare i file sulla base dei loro attributi e per supportare la loro ricerca.
- m Client che possono memorizzare i file sugli StorageServer o cercarli utilizzando l'IndexServer.

Ogni File è caratterizzato dai seguenti attributi di tipo String: *filename* (nome del file scelto dal Client) ed una lista di *keywords* utilizzate per classificare il contenuto del file.

Ogni StorageServer rimane in ascolto sulla porta TCP 2000 in attesa di connessioni da parte dei Client.

Quando un Client si connette, invia allo StorageServer il file da memorizzare (da simulare con un oggetto String) e gli attributi necessari per descriverlo (*filename* e *keywords*).

DIMES - UNIVERSITÀ DELLA CALABRIA  
Dipartimento di Ingegneria Informatica, Modellistica, Elettronica e Sistematica

8

Loris Belcastro, Reti di Calcolatori

### Esercizio 1: IndexServer

L'IndexServer utilizza due porte e protocolli distinti per le sue funzioni:

- Porta UDP 3000, su cui riceve da un generico StorageServer x l'informazione che x memorizza un nuovo file. A tale scopo x invia mediante datagramma tutti gli attributi del file in modo da permetterne l'indicizzazione sull'IndexServer, il quale a tale scopo utilizza idonee strutture dati.
- Porta TCP 4000, su cui riceve da un generico Client x la richiesta di cercare tutti i file, indicizzati dall'IndexServer, che soddisfano i valori degli attributi ricevuti tramite il socket. Ad esempio, la ricerca con i parametri *filename*=*song123*, *keywords*=[*classic*, "2015"] restituirà gli indirizzi IP degli StorageServer contenenti i file con nome *song123* e classificati con le keywords "classic" e "2015".

Si assume che i Client conoscano tutti gli StorageServer e che scelgano di volta in volta a caso quello su cui memorizzare un file.

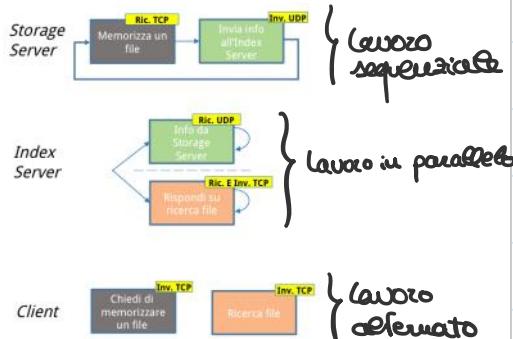
Il programma deve comprendere un main in cui un Client memorizza un file su uno StorageServer; questo file viene successivamente cercato da un altro Client rivolgendosi all'IndexServer.

DIMES - UNIVERSITÀ DELLA CALABRIA  
Dipartimento di Ingegneria Informatica, Modellistica, Elettronica e Sistematica

9

Loris Belcastro, Reti di Calcolatori

### Schema



DIMES - UNIVERSITÀ DELLA CALABRIA  
Dipartimento di Ingegneria Informatica, Modellistica, Elettronica e Sistematica

10

Loris Belcastro, Reti di Calcolatori

INGEGNERIA

ATICA

OLINO

Due funzioni:

- 1) Memorizzazione
- 2) Ricerca

## File (1) Dati scaricati

```

public class File implements Serializable {
    private String filename;
    private String[] keywords;
    private String contenuto;

    public File(String filename, String[] keywords, String contenuto) {
        super();
        this.filename = filename;
        this.keywords = keywords;
        this.contenuto = contenuto;
    }

    public String getFilename() {
        return filename;
    }

    public String[] getKeywords() {
        return keywords;
    }

    public String getContenuto() {
        return contenuto;
    }

    @Override
    public String toString() {
        return "File [filename=" + filename + ", keywords=" + Arrays.toString(keywords) +
            ", contenuto=" + contenuto + "]";
    }
}

```

metodi da implementare

DIMES - UNIVERSITÀ DELLA CALABRIA

11

Loris Belcastro, Reti di Calcolatori

1) La classe deve essere serializzabile affinché possano essere inviati dal client

## Index Server (2) Primo server

```

public class IndexServer {
    private Map<File, InetSocketAddress> data;
    private final int uPort = 3000; // Porta sulla quale riceve info da StorageServer
    private final int tPort = 4000; // Porta sulla quale riceve richieste da Client
    private ServerSocket server; TCP
    private DatagramSocket uSocket; UDP

    public IndexServer() {
        // Struttura dati thread-safe
        data = Collections.synchronizedMap(new HashMap<File, InetSocketAddress>());
        System.out.println("IndexServer in fase di avvio");
        inizia();
    }

    private void inizia() {
        try {
            server = new ServerSocket(tPort);
            System.out.println(server);
            uSocket = new DatagramSocket(uPort);
        } catch (IOException e) {
            e.printStackTrace();
        }
        new GestisciFile().start(); // Thread per la gestione richieste da StorageServer
        new GestisciClient().start(); // Thread per la gestione richieste da Client
    }
}

```

MAPPA  
SINCRONIZZATA

Inizializzazione  
socket

Inizializzazione Thread

DIMES - UNIVERSITÀ DELLA CALABRIA

12

Loris Belcastro, Reti di Calcolatori

\* assunzione fata!  
ad ogni file corrisponde  
un solo storage sever

2) Nei costruttori Bisogna creare le  
strutture dati sincronizzate

## Index Server (Gestisci File)

→ INNER CLASS → permette di avere direttamente colture

```

class GestisciFile extends Thread {
    public void run() {
        // Info da Storage Server
        while (true) {
            try {
                byte[] buf = new byte[256];
                // Riceve il pacchetto
                DatagramPacket packet = new DatagramPacket(buf, buf.length);
                System.out.println("Attesa nuovo pacchetto...");
                uSocket.receive(packet); → ricevuta bloccante
                String msg = new String(packet.getData());
                System.out.println(msg);

                separazione String[] parti = msg.split("#"); → FILE
                String[] keywords = parti[1].split(","); → KEYWORD
                Riconoscimento File file = new File(parti[0], keywords); → Si riconosce che sono word
                salvataggio System.out.format("Aggiungo il file %s inviato da %s%n", parti[0], packet.getAddress()); → indirizzo utente
                data.put(file, packet.getAddress()); // salvo file nella struttura dati
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}

```

DIMES - UNIVERSITÀ DELLA CALABRIA

13

Loris Belcastro, Reti di Calcolatori

GE

NERIA

ATICA

LINO

## Index Server (GestisciClient)

```

class GestisciClient extends Thread {
    public void run() {
        while (true) {
            try {
                // I ricevi ricerca nomefile#attributo1,attributo2
                System.out.println("Attesa nuova ricerca...");
                Socket tSocket = server.accept();
                BufferedReader br = new BufferedReader(new InputStreamReader(tSocket.getInputStream()));
                Ricezione
                String msg = br.readLine();
                System.out.println("Ricevi la ricerca " + msg);

                separazione String[] parti = msg.split("#");
                String[] keywords = parti[1].split(",");
                System.out.format("Ricevo la ricerca con filename %s e chiavi %s%n", parti[0],
                    parti[1]);
            } (Continua)
        }
    }
}

```

DIMES - UNIVERSITÀ DELLA CALABRIA

14

Loris Belcastro, Reti di Calcolatori

## Index Server (GestisciClient)

```

String ret = "";
    synchronized (data) { // Per evitare ConcurrentModificationException nel for
        for (Entry<File, InetAddress> entry : data.entrySet()) {
            File file = entry.getKey();
            if (file.getfilename().equals(parti[0])) {
                boolean trovato = true;
                for (int i = 0; i < keywords.length && trovato; i++) {
                    trovato = false;
                    for (int j = 0; j < file.getKeywords().length; j++) {
                        if (keywords[i].equals(file.getKeywords()[j])) {
                            trovato = true;
                            break;
                        }
                    }
                }
                if (trovato)
                    ret += entry.getValue().toString();
            }
        }
    } // synchronized

    // Rispondo al client
    System.out.println("Invio la risposta " + ret);
    PrintWriter pw = new PrintWriter(tSocket.getOutputStream(), true);
    pw.println(ret); → stampa sull' OutputStream

    br.close(); pw.close(); tSocket.close(); closure
} catch (IOException e) {
    e.printStackTrace();
}
}

```

DIMES - UNIVERSITÀ DELLA CALABRIA

15

Loris Belcastro, Reti di Calcolatori

## Index Server

```

// launcher del server
public static void main(String[] args) {
    IndexServer indexS = new IndexServer();
}

```

DIMES - UNIVERSITÀ DELLA CALABRIA  
Dipartimento di Ingegneria Informatica, Modellistica, Elettronica e Sistematica

16

Loris Belcastro, Reti di Calcolatori

## StorageServer ③ Secondo server

```

public class StorageServer {
    private HashMap<String, File> data; // struttura dati interna allo StorageServer

    private final int myPort = 2000; // porta di attesa connessioni da parte dei Client
    private final String indexServerAddress;
    private final int indexServerPort = 3000; // porta per invio richieste all'IndexServer

    private ServerSocket server;

    public StorageServer(String indexServerAddress) {
        this.indexServerAddress = indexServerAddress;
        data = new HashMap<String, File>();
        System.out.println("StorageServer in fase di avvio");
        inizia();
    }

    public void inizia() {
        try {
            server = new ServerSocket(myPort);
        } catch (IOException e) {
            e.printStackTrace();
        }
        while (true) {
            memorizzaFile();
        }
    }

    //Continua...
}

```

DIMES - UNIVERSITÀ DELLA CALABRIA  
Dipartimento di Ingegneria Informatica, Modellistica, Elettronica e Sistematica

17

Loris Belcastro, Reti di Calcolatori

## StorageServer

```

private void memorizzaFile() {
    try {

        // Ricevi File
        System.out.println("In attesa di memorizzare un file...");
        Socket tSocket = server.accept();
        ObjectInputStream ois = new ObjectInputStream(tSocket.getInputStream());
        File file = (File) ois.readObject();
        ois.close();
        tSocket.close();

        //1. Memorizza nella struttura dati
        data.put(file.getfilename(), file);
        System.out.println("Ho memorizzato il file " + file);

    } catch (IOException | ClassNotFoundException e) {
        e.printStackTrace();
    }
}

//Continua...

```

*put(chiave, valore)*

DIMES - UNIVERSITÀ DELLA CALABRIA  
Dipartimento di Ingegneria Informatica, Modellistica, Elettronica e Sistematica

18

Loris Belcastro, Reti di Calcolatori

## StorageServer

```

// 2 Invia info all'IndexServer
System.out.println("Invio datagramma all'IndexServer");
DatagramSocket uSocket = new DatagramSocket();

// 2.1 Prepara la stringa da inviare composta da nomeFile#attributo1,attributo2
String msg = file.getFilename()+"#";
for (String key:file.getKeywords()) {
    msg+=key+"#";
}

// 2.2 Prepara il pacchetto e lo invia
byte[] buf = msg.getBytes();
InetAddress address = InetAddress.getByName(indexServerAddress);
DatagramPacket packet = new DatagramPacket(buf, buf.length, address,
                                             indexServerPort);
uSocket.send(packet);
uSocket.close();

} catch (IOException e) {
    e.printStackTrace();
} catch (ClassNotFoundException e) {
    e.printStackTrace();
}

// launcher del server
public static void main(String[] args) {
    StorageServer storageS = new StorageServer("localhost");
}

```

DIMES - UNIVERSITÀ DELLA CALABRIA

L'invio includeva il  
costruttore a 4 parametri

19

Loris Belcastro, Reti di Calcolatori

## Client ④ Client (solo MAV)

```

public class Client {
    public static void main(String[] args) {
        try {
            //FASE 1: Memorizza file
            File file = new File("prova", new String[]{"a", "b", "c"}, "...");
            System.out.println("Salvo il file "+ file);
            Socket tSocket = new Socket("localhost", 2000); // apre socket con StorageServer
            ObjectOutputStream oos = new ObjectOutputStream(tSocket.getOutputStream());

            oos.writeObject(file);
            oos.flush(); oos.close(); tSocket.close();

            Thread.sleep(3000);

            file = new File("gatto", new String[]{"a", "b"}, "...");
            System.out.println("Salvo il file "+ file);
            tSocket = new Socket("localhost", 2000);
            oos = new ObjectOutputStream(tSocket.getOutputStream());

            oos.writeObject(file);
            oos.flush(); oos.close(); tSocket.close();

            Thread.sleep(3000);

        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}

```

DIMES - UNIVERSITÀ DELLA CALABRIA

20

Loris Belcastro, Reti di Calcolatori

→ METORIZZAZIONE

## Client

```

//FASE 2: ricerca file
try {
    //Ricerca file
    String ricerca="gatto;a,b";
    System.out.println("Ricerca tutti i file identificati da "+ricerca);
    Socket tSocket = new Socket("localhost", 4000); // apre socket con IndexServer
    PrintWriter pw = new PrintWriter(tSocket.getOutputStream(), true);
    pw.println(ricerca);

    BufferedReader br = new BufferedReader(new
        InputStreamReader(tSocket.getInputStream()));
    String risposta = br.readLine();
    System.out.println("Risultato ricerca:"+risposta);

    br.close();
    pw.close();
    tSocket.close();

} catch (Exception e) {
    e.printStackTrace();
}
}

```

DIMES - UNIVERSITÀ DELLA CALABRIA

21

Loris Belcastro, Reti di Calcolatori

BINARIO  
CARTIERI

IGEGNERIA  
ATICA  
OLINO

## Esercizio 2: VoliAeroporto-WS

Si realizzzi un *Web Service* che permette di ottenere alcune informazioni sui voli di un aeroporto.

In particolare, il servizio espone:

1. un *metodo* che, dato il nome di una città **C** e una data **D**, restituisce il codice del primo volo in partenza per **C** in data **D**.
2. un *metodo* che, dato il codice di un volo **V** e una data **D**, restituisce l'orario di partenza di **V** in data **D**.

come specificato nel file WSDL allegato.

Si implementi in Java una classe che implementa il servizio. Si chiede di scegliere le opportune strutture che: i) permettono di memorizzare i voli; ii) ottimizzano l'esecuzione delle query.

= HASH MAP

DIMES - UNIVERSITÀ DELLA CALABRIA

22

Loris Belcastro, Reti di Calcolatori

## WSDL

```

<wsdl:definitions targetNamespace="http://www.examples.com/wsdl/AirportService">
  <wsdl:types>
    <schema targetNamespace="http://DefaultNamespace">
      <import namespace="http://schemas.xmlsoap.org/soap/encoding"/>
      <complexType name="Data">
        <sequence>
          <element name="Giorno" type="xsd:int"/>
          <element name="Mese" type="xsd:int"/>
          <element name="Anno" type="xsd:int"/>
        </sequence>
      </complexType>
      <complexType name="Orario">
        <sequence>
          <element name="Ore" type="xsd:int"/>
          <element name="Minuti" type="xsd:int"/>
        </sequence>
      </complexType>
    </schema>
  </wsdl:types>

```

**Tipi**

**CLASSI** → Devo essere serializzabili e solo oggetti scambiati

**Types:** i tipi di dato che possono essere scambiati tra client e web service

## WSDL

### MESSAGGI

```

<wsdl:message name="PrimoVoloRequest">
  <wsdl:part name="in0" type="xsd:string"/>
  <wsdl:part name="in1" type="tns1:Data"/>
</message>
<wsdl:message name="PrimoVoloResponse">
  <wsdl:part name="in0" type="xsd:string"/>
</message>
<wsdl:message name="OrarioVoloRequest">
  <wsdl:part name="in0" type="xsd:string"/>
  <wsdl:part name="in1" type="tns1:Data"/>
</message>
<wsdl:message name="OrarioVoloResponse">
  <wsdl:part name="in0" type="tns1:Orario"/>
</message>

```

**PortType**

**Messages:** i messaggi che possono essere scambiati tra il web service e i client; sono definiti come composizione o aggregazione dei tipi elementari.  
NB: tns = this namespace

**PortType:** definisce i punti di connessione (operazioni) verso il WebService.

**METHODI**

## Data

```

public class Data implements Serializable{
  private int giorno, mese, anno;

  public Data(int giorno, int mese, int anno) {
    this.giorno = giorno;
    this.mese = mese;
    this.anno = anno;
  }

  public int getGiorno() {
    return giorno;
  }
  public int getMese() {
    return mese;
  }
  public int getAnno() {
    return anno;
  }

  public int hashCode() {
    return (anno+"-"+mese+"-"+giorno).hashCode();
  }

  public boolean equals(Object obj) {
    // vedi implementazione
  }

  public String toString() {
    return "Data [giorno=" + giorno + ", mese=" + mese + ", anno=" + anno + "]";
  }
}

```

Da implementare se il dato è usato come chiave dell'hashmap

## Orario

```

public class Orario implements Serializable{
  private int ore, minuti;

  public Orario(int ore, int minuti) {
    this.ore = ore;
    this.minuti = minuti;
  }

  public int getOre() {
    return ore;
  }
  public int getMinuti() {
    return minuti;
  }

  public int hashCode() {
    return (ore+"-"+minuti).hashCode();
  }

  public boolean equals(Object obj) {
    // vedi implementazione
  }

  public String toString() {
    return "Orario [ore=" + ore + ", minuti=" + minuti + "]";
  }
}

```

Client a Server  
↑  
• le richieste finiscono con Request  
e le risposte con Response  
↓

Server a Client

\* la prima parte del nome messaggio indica il nome del nome

IGEGNERIA  
ATICA  
OLINO

## OrarioComparator

```

public class OrarioComparator implements Comparator<Volo> {
    public int compare(Volo o1, Volo o2) {
        if(o1.getOrario().getOre()==o2.getOrario().getOre()
           && o1.getOrario().getMinuti()==o2.getOrario().getMinuti())
            return 0;
        if(o1.getOrario().getOre()<o2.getOrario().getOre())
            return -1;
        else if(o1.getOrario().getOre()>o2.getOrario().getOre())
            return +1;
        else if (o1.getOrario().getMinuti()<o2.getOrario().getMinuti())
            return -1;
        else if (o1.getOrario().getMinuti()>o2.getOrario().getMinuti())
            return +1;
        return 0;
    }
}

```

così o utilizzando gli if all'interno del metodo

## Volo

```

public class Volo{ → NO SERIALIZABLE (non scrive)
    private String citta;
    private Data data;
    private Orario orario;
    private String voloId;

    public Volo(String citta, Data data, Orario orario, String voloId) {
        super();
        this.citta = citta;
        this.data = data;
        this.orario = orario;
        this.voloId = voloId;
    }

    @Override
    public boolean equals(Object obj) {
        //vedi implementazione
    }

    @Override
    public int hashCode() {
        //vedi implementazione
    }

    //metodi get
    //-
}

```

## Aeroporto WEB SERVICE

```

public class Aeroporto implements AeroportoIf{
    private HashMap<Data, HashMap<String, LinkedList<Volo>>> db; → CHIAVE: DATA
    public Aeroporto(){ → Prendi data,
        db = new HashMap<Data, HashMap<String, LinkedList<Volo>>(); → ho un elenco
    } → di coppie città-volo
    public String privoVolo(String citta, Data data) { → OSERVO SIANO MAPPATI
        if(!db.containsKey(data) || !db.get(data).containsKey(citta)) → in ordine
            return "Nessun aereo";
        return db.get(data).getFirst().getVoloId();
    }

    public Orario orarioVolo(String voloId, Data data) {
        LinkedList<Volo> tmp;
        if(db.containsKey(data)){
            for (Entry<String, LinkedList<Volo>> entry: db.get(data).entrySet()) {
                tmp = entry.getValue();
                for (Volo volo2 : tmp) {
                    if(volo2.getVoloId().equals(voloId))
                        return volo2.getOrario();
                }
            }
        }
        return new Orario(-1, -1);
    }
}

```

## Esercizio 2.1 - VoliAeroporto-REST

- A partire dal WSDL fornito, si realizzi l'equivalente Web Service di tipo RESTful utilizzando le annotazioni JAX-RS utilizzate in Jersey.
- I tipi complessi (es. oggetti complessi) devono essere convertiti in oggetti di tipo application/json
- I path dei metodi dovranno essere definiti usando utilizzando il nome dell'operation definita nel metodo.

## (Richiamo) Annotazioni in Jersey

- **@Path:** si tratta di un'annotazione posta a livello della dichiarazione della classe che indica la URI di una risorsa (<http://www.mydomain.com/helloworld>). Rappresenta il path di base per tutti i metodi dell'API RESTful.
- **@Produces:** specifica uno o più MIME-Type con cui lo stato della risorsa può essere trasferito al client che ne fa richiesta.
- **@GET:** indica il metodo HTTP che deve essere usato per accedere a quel metodo (potrebbe essere sostituito con **@POST**, **@PUT**, **@DELETE**).
- **@Consumes:** viene utilizzata per specificare quali tipi di rappresentazione MIME dei media una risorsa può accettare o consumare dal client (es. `text/plain`, `application/json`, `application/xml`).

## Passaggio di parametri a REST API

```
// I parametri vengono passati nel path come /customer/12*
@GET
@Path("/customer/{id}")
public String getCustomer(@PathParam("id") final String id) {
    return "Customer ID: " + customers.get(id);
}

// I parametri vengono passati nella query string dell'url come /customer?name=Pippo"
@GET
@Path("/customer")
public List<String> getCustomer(@QueryParam("name") final String name) {
    return customers.findByName(name);
}

// I parametri vengono trasmessi nel body sotto forma di oggetto serializzato in JSON
@POST
@Path("/customer")
@Produces(MediaType.APPLICATION_JSON) // oppure @Produces("application/json")
@Consumes(MediaType.APPLICATION_JSON)
public List<String> createCustomer(final Customer c) {
    customers.add(c); return c;
}
```



## Aeroporto RESTful

```
@Path("/")
public class Aeroporto implements AeroportoIF{

    private HashMap<Data, HashMap<String, LinkedList<Volo>>> db;
    public Aeroporto(){
        db = new HashMap<Data, HashMap<String, LinkedList<Volo>>>();
    }

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String privoVolo(
            @QueryParam("citta") String citta,
            @QueryParam("data") String data) {
        if(db.containsKey(data) || !db.get(new Data(data)).containsKey(citta))
            return "Nessun aereo";
        return db.get(data).get(citta).getFirst().getVoloId();
    }

    public Orario orarioVolo(String voloId, Data data) {
        LinkedList<Volo> tmp;
        if(db.containsKey(data)){
            for (Entry<String, LinkedList<Volo>> entry: db.get(data).entrySet()) {
                tmp = entry.getValue();
                for (Volo volo2 : tmp) {
                    if(volo2.getVoloId().equals(voloId))
                        return volo2.getOrario();
                }
            }
        }
        return new Orario(-1, -1);
    }
}
```

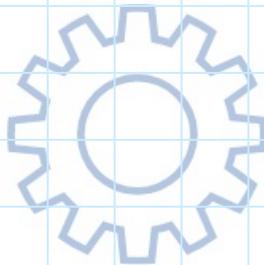
## Aeroporto RESTful

```
...
@GET
@Produces(MediaType.APPLICATION_JSON)
public Orario orarioVolo(
        @QueryParam("citta") String voloId,
        @QueryParam("data") String data) {
    LinkedList<Volo> tmp;
    if(db.containsKey(data)){
        for (Entry<String, LinkedList<Volo>> entry: db.get(data).entrySet()) {
            tmp = entry.getValue();
            for (Volo volo2 : tmp) {
                if(volo2.getVoloId().equals(voloId))
                    return volo2.getOrario();
            }
        }
    }
    return new Orario(-1, -1);
}
```

## Consigli per gli esercizi di programmazione di rete

- Studiare in modo approfondito tutti gli esercizi presentati durante le esercitazioni.
- Saper usare i Thread. → **LAMBDA**
- Gestione di connessioni multiple usando i Thread. → **LATO SERVER**
- Conoscere come «temporizzare» le scelte. → **SOCKET TIMEOUT O SLEEP DI UN THREAD**
- Conoscere le principali strutture dati.
- Conoscere le principali strutture dati sincronizzate. → **PiÙ THREAD CHE MODIFICANO LE STRUTTURE DATI**
- Saper gestire le eccezioni.

↓  
Es. ECCEZIONE  
SOCKET TIMEOUT



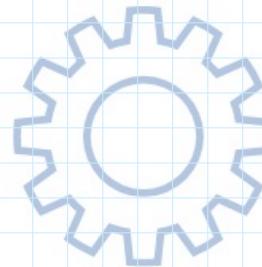
# APPUNTI DI INGEGNERIA INFORMATICA

GAIA BERTOLINO

## Domande scritte

sabato 3 settembre 2022 16:08

- 1• Si descriva, anche mediante figure opportunamente commentate, come funziona la **firma digitale**.
- 2• Si descriva, anche mediante figure opportunamente commentate, come funziona il protocollo **BitTorrent**.
- 3• Si descriva, anche mediante figure opportunamente commentate, come funziona la **risoluzione iterativa e ricorsiva del DNS**.
- 4• Si descriva, anche mediante figure opportunamente commentate, come funziona il metodo di crittografia **One-Two-Pad**.
- 5• Si descriva, anche mediante figure opportunamente commentate, come funziona il **NAT**.
- 6• Si descriva, anche mediante figure opportunamente commentate, come funziona la **frammennazione e la deframmennazione nel protocollo IP**.
- 7• Si descriva, anche mediante figure opportunamente commentate, come funziona l'**intradamento intra-AS e inter-AS**.
- 8• Si descriva, anche mediante figure opportunamente commentate, il funzionamento dei protocolli **go-back-n e selective repeat**.
- 9• Si descriva, anche mediante figure opportunamente commentate, quali sono le funzioni svolte e i vantaggi derivanti dall'uso del proxy nel **protocollo HTTP**.
- 10• Si descriva, anche mediante figure opportunamente commentate, come possono essere **individuati i file in una rete peer-to-peer**.
- 11• Si descriva, anche mediante figure opportunamente commentate, come funziona il **controllo del flusso nel protocollo TCP**.
- 12• Si descriva, anche mediante figure opportunamente commentate, il funzionamento del **protocollo HTTP**.
- 13• Si descriva, anche mediante figure opportunamente commentate, il funzionamento dei principali **protocolli di posta elettronica**.
- 14• Si descriva, anche mediante figure opportunamente commentate, perché viene effettuata e come viene realizzata la **frammennazione e il successivo riassembaggio dei datagrammi IP**.
- 15• Si descriva, anche mediante figure opportunamente commentate, come vengono effettuate la **connessione e la disconnessione nel protocollo TCP**.
- 16• Si confrontino le caratteristiche principali dei seguenti approcci **peer-to-peer**: con server centrale, basato su flooding, basato su DHT.
- 17• Si descriva, anche mediante figure opportunamente commentate, le differenze tra **crittografia a chiave pubblica e crittografia simmetrica**.



# APPUNTI DI INGEGNERIA

## ① FIRMA DIGITALE

- 1) TRE FATTORI (autenticazione, non riproducibilità, integrità)
- 2) USO DELLA CHIAVE PRIVATA
- 3) DIGEST
- 4) CA

La firma digitale è un meccanismo di firma dei documenti digitali in grado di preservare e autentificare sia il mittente che il contenuto e di rintracciare anche eventuali manomissioni. Infatti rispetta tre importanti fattori:

- 1) autenticazione univoca del mittente
- 2) non riproducibilità di mittente e contenuto
- 3) integrità del contenuto stesso

La firma digitale viene realizzata tramite principi matematici e crittografici che prevedono l'uso di una chiave privata tramite la quale il mittente crittografa i propri messaggi e una chiave pubblica che il ricevente utilizzerà per decrittare il file. Il rilascio e la verifica dei certificati legati alle chiavi pubbliche è compito degli enti certificatori (CA).

Alla base del meccanismo c'è il digest ovvero una stringa alfanumerica ricavata dall'applicazione della chiave pubblica alla firma digitale da parte del ricevente. Tale stringa definisce tutto il contenuto del file e una sola modifica cambierebbe totalmente il valore della stringa. Dunque riesce a verificare l'integrità del file ricevuto.

## ② BITTORRENT

- 1) P2P appunto alla rete
- 2) PEER - SEED - TRACKER - LEECHERS
- 3) CHUNK (256 kb)

BitTorrent è un protocollo di comunicazione Peer-2-Peer non puro utilizzato per lo scambio di file in rete in maniera (quasi) diretta. Esso infatti prevede che un nuovo host che entra sulla rete comunihi con un Tracker (un server) per venire a conoscenza della rete e della dislocazione dei dati. Lo scambio fra host è di pezzi di informazione chiamati chuck, ciascuno dei quali è pari a 256 kb. Coloro che hanno scaricato parzialmente il file sono detti Peer mentre coloro che lo hanno scaricato interamente e rimangono sulla rete sono detti seed o se escono dalla rete sono detti leechers. Infatti, un host può decidere di rimanere nella rete e fare a sua volta da nodo di trasmissione per gli altri peer che richiedono l'informazione o lasciare la rete (e in questo caso sono detti appunto leechers ovvero sanguisughe). All'interno del protocollo sono previste alcune tecniche come il rarest first che prevede l'intercettazione prima di quei pacchetti che sono meno presenti sulla rete in modo da renderli più disponibili mentre un'altra tecnica è la tit-for-tat che prevede la scelta di 4 nodi preferiti che

DISEGNO SUL LIBRO DELLA FIRMA DIGITALE (NUMERO 1) (VERSIONE EFFICIENTE)

- 2) Si descriva, anche mediante figure opportunamente commentate, come funziona il protocollo **BitTorrent**.

BitTorrent è un protocollo P2P (Peer to Peer, NON PURO dato che la sua architettura prevede un SERVER) finalizzato alla distribuzione e condivisione di file nella rete.

A differenza dei tradizionali sistemi di file sharing, l'obiettivo di BT è di fornire un sistema efficiente per distribuire lo stesso file verso il maggior numero di utenti disponibili, che possono sia prelevarlo (Download) e sia inviarlo ad altri (Upload).

Esso impone un meccanismo di coordinamento di numerosi computer, ottenendo il massimo dell'utilizzo della rete. Come ogni altro sistema di condivisione di diffusione di file, I SEED sono coloro che hanno già scaricato interamente il file e stanno continuando il loro UPLOAD per aiutare gli altri PEER.

Primitivamente esiste un TRACKER che tiene traccia di tutti i PEER (partecipanti) che partecipano alla condivisione del file.

Il file viene suddiviso in pezzi da 256 kb detti **CHUNK**.

Il file viene suddiviso in parti da 256 Kb detti CHUNKS.  
I peer possono entrare ed uscire a piacimento dal torrent, anche una volta che hanno finito, decidendo così egoisticamente di non condividere più il file.  
Una tecnica importante che si può adottare è quella del RAREST FIRST per poter scaricare le sue parti mancanti, ovvero seleziona 4 peer preferiti con cui sta condividendo a frequenza più alta, questi vengono rivalutati ogni 10 secondi.  
Invece, ogni 30 secondi viene selezionato casualmente un nuovo PEER.

- 3) Si descriva, anche mediante figure opportunamente commentate, come funziona la risoluzione iterativa e ricorsiva della rete.

Il DNS (Domain Name System) è un sistema utilizzato per assegnare nomi ai nodi della rete. Questi nomi sono utilizzabili, mediante una traduzione, di solito chiamata "Risoluzione", al posto degli indirizzi IP originali. Il sistema è realizzato grazie ad un database distribuito che viene implementato in una gerarchia di server DNS ad albero rovesciato e viene dominato. Praticamente, quando richiediamo di collegarci ad un server tramite il suo "hostname", stiamo chiedendo al server DNS di andare all'interno di questi database e tradurre quel hostname nell'indirizzo IP del server in cui viene ospitato tale sito. Un host può avere più nomi.  
La "risoluzione iterativa" del DNS avviene praticamente quando il server contattato risponde con il nome del server da contattare quindi si continuerà ad iterare fino ad arrivare al server finale per la traduzione.  
La "risoluzione ricorsiva" praticamente va ad affidare il compito di tradurre il nome al server DNS contattato che a sua volta andrà in modo ricorsivo a contattare gli altri ad ottenere le informazioni.

- 4) Si descriva, anche mediante figure opportunamente commentate, come funziona il metodo di crittografia One-Time-Pad.

Il sistema crittografico One-Time-Pad (OTP) fa parte degli algoritmi a chiave segreta. Praticamente viene chiamato così perché la chiave segreta venisse utilizzata una singola volta e poi estinta. Il testo viene codificato con una chiave della stessa lunghezza di esso. Viene definito come "algoritmo inattaccabile" proprio per queste caratteristiche.  
Il sistema si basava su di un cifrario di parole rappresentato da numeri a 4 cifre, accoppiate poi in gruppi di cinque cifre e sommate ad un numero casuale. Se l'ultimo gruppo di accoppiamento a cinque cifre non è completo (cioè ha meno di 5 cifre), la sequenza doveva essere completata a destra con dei zeri (es. 123 → 12300).

- 5) Si descriva, anche mediante figure opportunamente commentate, come funziona il NAT.

Il NAT (Network Address Translation) ovvero traduzione degli indirizzi di rete. Il NAT è implementato dai router e dai firewalls e ne esistono di diversi tipi (SNAT, DNAT, dipendentemente da quale indirizzo viene modificata SOURCE o DESTINATION). Praticamente, questa tecnica consiste nel nascondere i dettagli della rete domestica al mondo esterno tramite il router abilitato al NAT. Infatti, non è necessario allocare un intervallo di indirizzi IP da un ISP perché un unico IP è sufficiente per tutte le macchine della rete locale.

Quando un router NAT riceve il datagramma, genera per esso un nuovo numero di porta d'origine e sostituisce l'IP origine con l'IP lato WAN, ovviamente utilizzando delle tabelle di supporto interne. Infatti, esso modifica gli IP contenuti dentro gli header dei pacchetti in transitazione.

Il campo numero di porta è lungo 16 bit.

In ogni caso, questa traduzione è contestata perché i router dovrebbero elaborare pacchetti solo fino al livello 3 e quindi va a violare l'END to END tra host. Si ha anche un'interruzione con applicazioni di tipo P2P se non adeguatamente configurato.

FIGURA NUMERO 1 SUL LIBRO DEL NAT

- 6) Si descriva, anche mediante figure opportunamente commentate, come funziona la frammentazione e la deframmentazione nel protocollo IP.

Le funzioni chiave del livello rete sono l'inoltro, frammentazione e deframmentazione e l'instradamento dei pacchetti utilizzando varie tecniche. Proprio per facilitare queste funzioni, in questo livello viene introdotta una quantità detta MTU (Maximum Transmission Unit) che indica la quantità massima di dati che può essere trasmessa; essa varia in base alla rete in cui si trova a comunicare. La frammentazione avviene quando viene inserito nella rete un pacchetto di dimensione superiore alla MTU e questo deve essere trasmesso. Infatti viene frammentato in datagrammi più piccoli. I frammenti verranno inseriti ed inviati e verranno assemblati solamente raggiunta la destinazione, ovviamente per fare questo vengono utilizzati gli header dei pacchetti che contengono le informazioni chiavi per l'assemblamento. Vari flag presenti nel datagramma guideranno la frammentazione e la deframmentazione.

- 7) Si descriva, anche mediante figure opportunamente commentate, come funziona l'instradamento intra-AS e inter-AS.

Il livello rete ha diversi compiti all'interno della pila protocollare, tra cui quello del routing (instradamento), ovvero determinare il percorso ideale per la trasmissione dei dati attraverso la rete. La maggior parte dei casi, questa funzione viene svolta dinamicamente tramite appositi algoritmi (Globale, Centralizzato, Dinamico, Statico), che utilizzano informazioni provenienti dai protocolli di routing sulle condizioni della rete.  
Un AS è un Autonomous System (Sistema autonomo) ovvero un insieme di router che si trovano all'interno di questo sistema autonomo in cui vengono gestiti tramite degli appositi algoritmi e protocolli, ovvero è sotto il controllo di una singola e ben definita autorità amministrativa. Possono esistere molti AS e all'interno di essi non perforza vengono utilizzati protocolli uguali.  
Un instradamento di tipo 'Intra-AS' vuol dire che i router appartenenti ad un dato AS utilizzano lo stesso algoritmo di instradamento in modo da poter comunicare fra di loro e gestire il traffico di rete adeguatamente. I protocolli di instradamento 'Intra-AS' sono noti come IGP (Protocolli Gateway Interni).  
Un instradamento di tipo 'Inter-AS' sono tutte le tecniche utilizzate per poter far sì che i pacchetti indirizzati al di fuori della propria AS vengano gestiti in modo adeguato e quindi scegliere verso quale gateway dovranno viaggiare. Ovviamente

escendo dalla rete sono detti leechers. Infatti, un nodo può decidere di rimanere nella rete e fare a sua volta da nodo di trasmissione per gli altri peer che richiedono l'informazione o lasciare la rete (e in questo caso sono detti appunti leechers ovvero sanguisughe). All'interno del protocollo sono previste alcune tecniche come il rarest first che prevede l'intercettazione prima di quei pacchetti che sono meno presenti sulla rete in modo da renderli più disponibili mentre un'altra tecnica è la tit-for-tat che prevede la scelta di 4 nodi preferiti che cambiano ogni 10 secondi e che vengono ricalcolati in base alla velocità di interazione. Infatti, un nodo che entra per ottenere dei dati, diventa a sua volta un nodo che fornisce dati agli altri nodi.

### ③ DNS

- 1) TRADUZIONE = RISOLUZIONE
- 2) SERVIZIO
- 3) ITERATIVO VS RICORSIVO

Disegno del DNS  
iterativo e del  
DNS ricorsivo

Il DNS (o domain name system) è un sistema utilizzato per la traduzione fra hostname e indirizzi IP delle risorse. Infatti, la comunicazione fra servizi web avviene tramite IP ed è dunque necessario tradurre gli hostname quando questi vengono utilizzati per l'accesso alle risorse. Il procedimento di traduzione è detto risoluzione. In particolare, esso prevede l'esistenza di una gerarchia di server che fanno da dizionari e che sono consultabili tramite due meccanismi diversi:

- il meccanismo iterativo prevede che il server contattato alla ricerca della traduzione fornisca le indicazioni relative ad un secondo server a cui accedere per ricercare le informazioni necessarie dopodiché si itera il processo
- il meccanismo ricorsivo prevede che sia il server stesso, quello contattato per primo, a risalire la catena e a restituire l'informazione ricercata dopo aver interagito con la gerarchia di server.

### ④ ONE TIME PAD

L'OTP (One time pad) è una meccanismo inattaccabile di cifratura dei messaggi. Esso si basa su tre concetti:

- 1) uso di una chiave privata usa e getta
- 2) uso di una chiave lunga quanto il testo da criprire
- 3) uso di una chiave generata casualmente

Esso prevede di tradurre il testo da alfabetico a numero traducendo ogni lettera con il corrispettivo numero in ordine alfabetico e di sommarlo secondo l'aritmetica modulare base 10 alla chiave generata casualmente anch'essa dunque convertita in numeri. Il messaggio criptato può essere così inviato e sarà traducibile col meccanismo inverso soltanto da coloro che hanno la chiave segreta

### ⑤ NAT (NETWORK ADDRESS TRANSLATION)

Disegno

Il Network Address Translation è un meccanismo di mascherazione degli IP interni ad una rete domestica attraverso un router che si comporta da traduttore per le comunicazioni. La motivazione dell'uso di questo meccanismo risiede nel fatto che grazie ad esso tramite un singolo indirizzo IP è possibile rappresentare una intera rete; quando poi i pacchetti giungono al router, grazie a delle tabella interne esso è in grado di reinidirizzare i dati verso l'host cercato. Tuttavia, tale modalità di interazione è abbastanza criticata in quanto viola il meccanismo END-to-end e peer to peer di comunicazione.

ROUTER - COMUNICAZIONE  
|  
PC

### ⑥ FRAMMENTAZIONE E DEFRAFFMENTAZIONE

Il protocollo IP è un protocollo senza connessione e poco affidabile che prevede l'invio dei dati in pacchetti di una dimensione massimo detta MTU (Maximum Transfer Unit). Dunque, tutti i dati da inviare vengono frammentati in piccole porzioni e l'informazione è ricomposta all'arrivo tramite opportuni flag all'interno dell'header del pacchetto che indicano il numero del frammento e quanti pacchetti ancora devono giungere. La deframmentazione deve tenere conto di tutte le informazioni inviate coi pacchetti stessi e ne consente che se si perde anche solo uno di questi pacchetti allora è impossibile ottenere i dati iniziali. Inoltre, i pacchetti possono giungere per vie diverse all'host finale.

### GAIA BERTOLINO

### ⑦ AUTONOMOUS SYSTEM E COMUNICAZIONE INTRA - INTER NET

Gli Autonomous Systems sono reti di router gestite da una specifica autorità e caratterizzati da protocolli e meccanismi di scambio dei dati.

In particolare, l'instradamento è una delle mansioni affidate al livello di rete che utilizza determinati algoritmi per inoltrare l'informazione lungo la rete dal mittente al destinatario.

Le comunicazioni degli AS possono essere interne o esterne. Le comunicazioni interne (Intra-AS) sono gestite tramite IGP ovvero Protocolli Gateaway interni e servono ai router per scambiare informazioni riguardo le tabelle di routing per l'instradamento dei dati mentre le comunicazioni fra più AS dette Inter-AS sono gestite tramite protocolli BGP ovvero border gateway protocol

ci sono diversi protocolli utilizzati per poter capire quale gateway è il più adatto da utilizzare.

8) Si descriva, anche mediante figure opportunamente commentate, il funzionamento dei protocolli go-back-n e selective repeat.

- All'interno di un protocollo go-back-n il mittente può trasmettere più pacchetti senza dover attendere nessun ack; ma non può più di un numero massimo N di pacchetti consecutivi in attesa di ack all'interno della pipeline. L'intervallo di pacchetti trasmessi che non hanno ancora ricevuto l'ack può essere visto come una finestra di dimensione N. Quando il protocollo è in funzione, questa finestra trasla. All'interno di questo protocollo, l'ack è di tipo cumulativo: esso indica che tutti i pacchetti presenti in quella sequenza minore o uguale ad N sono stati ricevuti correttamente. Quando si riceve un ack, il mittente invia nuovamente tutti i pacchetti specifici che ancora non hanno ricevuto un ack.
- Il problema del go-back-n è che un errore su un singolo pacchetto può provocare un numero elevato di ritrasmissione e causare una saturazione della pipeline. Il protocollo selective-repeat evita ritrasmissioni non necessarie facendo trasmettere al mittente solo quei pacchetti in cui esistono sospetti di errori, ossia smistramento o alterazioni. Questa forma forza il destinatario all'interno di ack specifici per i pacchetti ricevuti in modo corretto. La finestra del lato mittente si sposta ogni volta che vengono ricevuti correttamente gli ack per i pacchetti inviati, prodotti dal lato ricevitore. La finestra dal lato ricevitore si sposta solo se i pacchetti vengono ricevuti in modo corretto, provenienti dal lato mittente.

9) Si descriva, anche mediante figure opportunamente commentate, quali sono le funzioni svolte e i vantaggi derivanti dall'uso del proxy nel protocollo HTTP.

- Un proxy indica un tipo di server che funge da intermediario per le richieste da parte dei client alla ricerca di risorse su altri server, disaccoppiando l'accesso al web dal browser. Un client si connette al server proxy, richiedendo qualche servizio e quest'ultimo valuta ed esegue la richiesta in modo da semplificare e gestire la sua complessità.
- I server proxy vengono utilizzati per svariati tipi di impieghi:
- ANONIMATO per poter nascondere da eventuali minacce in rete, nascondere i propri dati sensibili.
- HTTP Caching Proxy ovvero si cerca di soddisfare la richiesta del client senza coinvolgere il server di origine. Praticamente il browser trasmette tutte le richieste http alla cache, se l'oggetto si troverà già nell'interno della cache, essa fornirà semplicemente l'oggetto richiesto. Altrimenti, la cache andrà a richiedere l'oggetto al server di origine. Essa riduce i tempi di risposta alle richieste del client, riduce il traffico di collegamento ad Internet e consente ai provider scadenti di offrire un servizio efficace.
- FIREWALL (Barriera di difesa) verso il web agendo da filtro per le connessioni entranti ed uscenti.

10) Si descriva, anche mediante figure opportunamente commentate, come possono essere indicizzati i file in una rete peer-to-peer.

- In una rete di tipo P2P, i file vengono indicizzati in modo centralizzato o completamente distribuito. Un approccio completamente distribuito viene preferito per via delle sue proprietà che permettono un minor utilizzo di memoria e di rendere scalabile il sistema. Quando parliamo di approccio completamente distribuito, stiamo parlando delle tabelle di hash distribuite (DHT) che sono una classe di sistemi distribuiti decentralizzati che partizionano l'appartenenza di un set di chiavi tra i nodi partecipanti, e possono inoltre in maniera efficiente i messaggi all'unico proprietario di una determinata chiave. Ciascun nodo è l'analogo di un array slot in una hash table.
- Le DHT sono tipicamente progettate per gestire un vasto numero di nodi, anche nei casi in cui ci sono continui ingressi o improvvisi guasti di alcuni di essi. Questo tipo di infrastruttura può essere utilizzata per implementare servizi più complessi. Le caratteristiche delle DHT enfatizzano le seguenti proprietà:

Decentralizzazione  
Scalabilità  
Tolleranza ai guasti

La tecnica chiave utilizzata per raggiungere questi scopi è costituita dal fatto che ciascun nodo ha bisogno di rimanere in contatto solo con un piccolo numero di altri nodi della rete; in questo modo, la rete è sottoposta ad una minima quantità di lavoro per ogni modifica che avviene nel set di nodi che la costituiscono.

11) Si descriva, anche mediante figure opportunamente commentate, come funziona il controllo del flusso nel protocollo TCP.

- Il TCP (Transmission Control Protocol) è un protocollo di rete a pacchetto, livello di trasporto, si occupa di controllo della trasmissione ovvero rende affidabile la comunicazione dati in rete tra mittente e destinatario. L'affidabilità della comunicazione in TCP è garantita anche dal cosiddetto "Controllo di flusso" ovvero far in modo che il flusso di dati in trasmissione non superi le capacità di ricezione ovvero di memorizzazione del ricevente quindi con causa di pacchetti a maggior peso e latenze nelle successive richieste di trasmissione. Infatti, viene attuato un opportuno campo noto come "Windows Receiver", variabile dinamica, ossia che varia in base allo spazio disponibile e che quindi specifica il numero massimo di segmenti ricevibili dal destinatario. Questa variazione della finestra avviene tramite l'intervallo di ACK in cui varia il campo "WND\_RCV". TCP per evitare l'Overflow del proprio buffer, renderà noto lo spazio disponibile in esso:
- RCV\_WND = RCVBuffer - [LastByteRCV - LastByteRead] dove ovviamente per negare l'overflow si avrà RCVBuffer >= LastByteRCV - LastByteRead.

## ⑥ GO-BACK-N E SELECTIVE REPEAT

Il go back n e il selective repeat sono protocolli di ritrasmissione di pacchetti in caso di perdita o non ricezione dello stesso o della sua conferma di ricezione chiamata ack.

Il protocollo go-back-n prevede l'uso di una finestra che gestisce l'invio di pacchetti: ogni ack di ricezione è cumulativo e permette l'invio di un pacchetto successivo, facendo dunque traslare tale finestra. Tuttavia, la mancata ricezione di un ack causa il rinvio di tutti i pacchetti presenti nella finestra (se essa avrà dimensione N, dovranno essere rimandati N pacchetti) causando molte ritrasmissioni inutili.

Il protocollo selective repeat prevede una configurazione simile ma utilizza ack singoli e non cumulativi il che permette di far comunque slittare in avanti la finestra di ricezione di una posizione ma la mancata ricezione dell'ack o di un pacchetto causa il rinvio di un solo pacchetto. Esso può essere considerato come un go-back-n con finestra di dimensione pari ad 1.

## ④ PROXY

Il proxy è un tipo di server che funge da intermediario fra le richieste di un client e l'effettivo server in cui si trovano le informazioni ricercate. Esso gestisce tutte le richieste permettendo di non sovraccaricare di richieste l'effettivo server, proteggendolo dunque da attacchi di tipo Dos o Ddos che intaccano così il solo proxy così come da tutti gli attacchi malevoli finalizzati anche a prelevare dati sensibili o importanti. Oltre alla funzione di firewall, è in grado di gestire anche le richieste in maniera indipendente dal server grazie ad una cache che permette di restituire dei dati se sono già presenti in questa memoria altrimenti la richiesta viene elaborata dal server, permettendo così di velocizzare l'interazione e non sovraccaricare il server principale.

## ⑩ INDICIZZAZIONE RETE P2P

In una rete peer-to-peer, l'indicizzazione dei file può avvenire in maniera centralizzata o distribuita. Secondo un approccio centralizzato, vi è bisogno di un maggior utilizzo di memoria e risorse e un server facilita l'interazione fornendo un servizio di localizzazione dell'informazione fra i peer della rete. Nell'approccio decentralizzato, ogni nodo contiene delle informazioni parziali grazie al fatto di dover interagire con pochi altri nodi. Tale tipo di principio permette una forte scalabilità del sistema, una forte resistenza ai guasti e una forte elasticità sia per l'inserimento che la rimozione (ad esempio a causa di una perdita) di nodi appartenenti alla rete stessa.

Di conseguenza, nessun nodo è obbligato a lavorare né gli sono richieste eccessive risorse ma la potenza richiesta viene suddivisa fra i vari nodi della rete.

## ⑪ CONTROLLO DI FLUSSO DEL TCP

Il TCP (Transport Control Protocol) è un protocollo di livello trasporto che rende affidabile la connessione attraverso alcuni meccanismi di controllo e gestione dell'invio dei dati in una trasmissione.

In particolare, esso ricorre al controllo di flusso ovvero alla regolamentazione del flusso di dati fra sorgente e mittente affinché non si perdano dei pacchetti a causa della diversa velocità di elaborazione fra due enti diversi. Esso ricorre all'utilizzo di una proprietà che descrive la grandezza della finestra disponibile da parte del ricevente che comunica la propria saturazione attraverso l'invio di ack con specifici campi nel proprio header. Di conseguenza, il mittente riesce a capire se il suo invio è troppo veloce e tramite l'utilizzo di buffer la comunicazione viene gestita affinché nessun pacchetto vada perso.

Se la RCV\_WND dove essere pari a zero, il mittente continuerà a mandare segmenti di un bye per garantire la sincronizzazione tra sender e receiver.

- 12) Si descriva, anche mediante figure opportunamente commentate, il funzionamento del protocollo HTTP.

Il protocollo **HTTP** (HyperText Transfer Protocol) è un protocollo a livello applicativo usato come principale per la trasmissione d'informazioni sul web ovvero in un'architettura del tipo Client-Server. Esso utilizza il protocollo TCP per instaurare una connessione ed utilizza la porta 80. Il client (browser) manda delle richieste al server (server http) che risponderà.

HTTP è un protocollo persistente senza stato, infatti il server non mantiene informazioni sulle richieste fatte dal client.

HTTP è un protocollo con connessioni persistenti e non persistenti. Con connessione non persistente praticamente almeno un oggetto viene trasferito su una connessione TCP mentre in una connessione persistente, più oggetti possono essere trasmessi nella connessione instaurata tra Client e Server.

Un importante figura all'interno di questo protocollo sono i messaggi e i comandi che vengono utilizzati. I comandi sono "GET", "POST", "PUT", "HEAD" e "DELETE".

- 13) Si descriva, anche mediante figure opportunamente commentate, il funzionamento dei principali protocolli di posta elettronica.

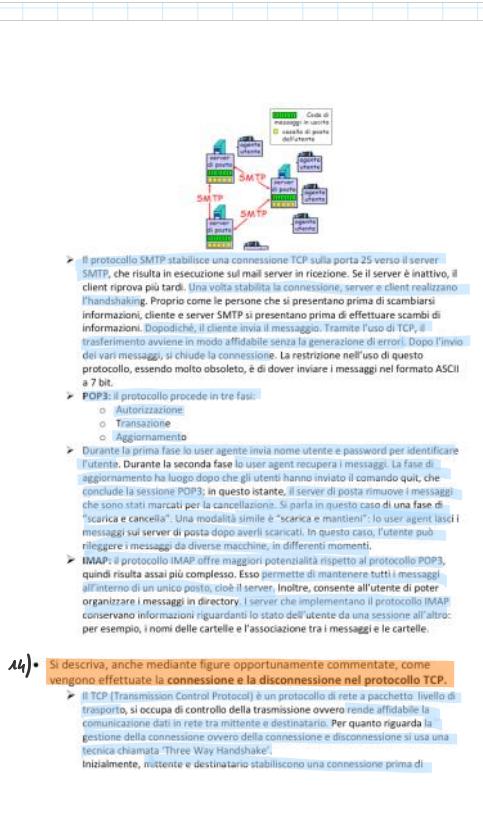
Proprio come il servizio postale tradizionale, l'e-mail rappresenta un mezzo di comunicazione asincrono; ciò vuol dire che le persone hanno la possibilità di inviare e leggere i messaggi quando vogliono, senza doversi coordinare. La differenza è che la posta elettronica è veloce, facile da distribuire e gratuita. Possiede molte caratteristiche importanti:

- o Allegati
- o Collegamenti ipertestuali
- o Testo con formattazione HTML

All'interno di un sistema postale di Internet esistono tre componenti principali:

- o gli user agent (o agenti utente)
- o i server di posta (o mail server)
- o il protocollo SMTP (simple mail transfer protocol)

Gli user agent, come ad esempio Microsoft Outlook, Thunderbird ecc., permettono agli utenti di leggere, rispondere, inoltrare, salvare e comporre i propri messaggi. Una volta che un utente ha composto il proprio messaggio, il suo user agent lo invia al server di posta, dove viene posto nella coda dei messaggi in uscita. Quando un altro utente vorrà leggere il messaggio, il suo user agent lo recupererà dalla casella di posta nel mail server.



- 14) Si descriva, anche mediante figure opportunamente commentate, come vengono effettuate la connessione e la disconnessione nel protocollo TCP.

Il TCP (Transmission Control Protocol) è un protocollo di rete a pacchetto. Livello di trasporto, si occupa di controllo della trasmissione ovvero rende affidabile la comunicazione dati in rete tra mittente e destinatario. Per quanto riguarda la gestione della connessione ovvero della connessione e disconnessione si usa una tecnica chiamata "Three Way Handshake".

Inizialmente, mittente e destinatario stabiliscono una connessione prima di

scambiare i dati, vengono inizializzate alcune variabili utili come per esempio la dimensione della finestra di invio/ricchezza. Il client avvia la connessione e ha iniziato la tecnica TWH: il client invia un segmento SYN (Synchronize) al server, il server risponde con un SYNACK ovvero se la sincronizzazione è andata a buon fine o meno ed in questo caso il server andrà ad inizializzare il buffer ed altre variabili utili alla comunicazione. Il client risponderà con un ACK se tutto è andato a buon fine, così il canale viene creato e si può iniziare a comunicare.

Per quanto riguarda invece la chiusura della connessione, il client quando ha finito dati da inviare, invia un segmento FIN, il server risponde con un segmento ACK, chiude la connessione ed invia un segmento FIN. Il client riceverà il FIN e risponderà con un ACK. Quando il server riceverà l'ACK di chiusura, chiuderà effettivamente la connessione.

Esistono anche delle varianti del Three Way Handshake.

- 15) Si confrontino le caratteristiche principali dei seguenti approcci peer-to-peer: con server centrale, basato su flooding, basato su DHT.

All'interno del sistema P2P gli utenti, chiamati appunto peers (nodi), possono connettersi tra loro per scambiarsi dati e risolvere problemi. Nel caso di sistemi centralizzati, vi è la presenza di un server che ha il compito di conservare tutti i nomi dei file che vengono utilizzati dagli utenti. I vari utenti a loro volta possono trovare l'indirizzo di un altro peer tramite l'aiuto del server formulando delle query, senza l'opera di altri intermediari. Lo scambio coinvolge esclusivamente i due utenti.

## (12) PROTOCOLLO HTTP

Il protocollo HTTP è utilizzato a livello applicativo per la gestione delle comunicazioni internet e dunque per le interazioni di tipo Client-Server.

Esso è di tipo stateless ovvero non memorizza le precedenti interazioni e richiede che con ogni richiesta siano inoltrate tutte le informazioni necessarie all'esecuzione della richiesta stessa. Tuttavia, oggi si ricorre all'uso di tecnologie come ad esempio i cookie per limitare la necessità di una grande mole di informazioni.

Esso inoltre prevede sia una connessione tramite TCp di tipo persistente che non: infatti, è possibile sia instaurare connessioni persistenti che permettano l'interazione su un unico canale sia non persistenti che prevedono la chiusura del canale al termine di ogni nuova interazione. Oggi si tende ad usare una visione ibrida delle due per velocizzare l'accesso alle informazioni, specialmente quando situate in diversi router. Le richieste client-server richiedono la specifica dell'operazione che si vuole eseguire: ad esempio, alcune di esse sono la post, la get, la delete ecc.

## (13) PROTOCOLLI DI POSTA ELETTRONICA

La posta elettronica è un tipo di messaggistica asincrona ovvero prevede che due nodi che comunicano non debbano essere entrambi collegati in contemporanea per scambiare i messaggi. Infatti, essa prevede l'uso di protocolli e server ad hoc che permettono l'invio e la lettura dei messaggi in maniera non sincrona.

Per inviare la posta elettronica intervengono anche alcune figure come ad esempio di user agent (ad esempio outlook) che comunicano con i server di posta

per scaricare o inviare dei messaggi.

I protocolli utilizzati sono diversi per l'invio e la ricezione dei messaggi: per l'invio si utilizza il SMTP che, attraverso il protocollo TCp e una comunicazione sulla porta 25, avvia una comunicazione che avviene prima tramite un handshaking a cui segue l'invio di messaggi e dopo la chiusura della connessione.

Per quanto riguarda la ricezione dei messaggi, esistono due protocolli: POP3 prevede che i messaggi scaricati da un server di posta vengano cancellati sul server stesso e dunque non siano disponibili per sessioni su altri dispositivi.

Invece, il protocollo IMAP prevede che i messaggi vengano mantenuti sul server, così come altre informazioni specifiche, e siano dunque riutilizzabili in altre sessioni su altri dispositivi.

## TI DI INGEGNERIA

### GAIA BERTOLINO

## 44 THREE-WAY HANDSHAKE CONNESSIONE E DISCONNESSIONE TCP

Il protocollo TCP (Transfer Control Protocol) è un protocollo di livello trasporto che regola il flusso della comunicazione. In particolare, è usato per rendere affidabile la connessione fra client e server.

La connessione secondo il TCP avviene attraverso lo scambio di messaggi, il cosiddetto three handshake: il client manda un messaggio detto SYNC al server con delle informazioni iniziali sulla comunicazione; il server risponde con un messaggio chiamato SYN-ACK e inizializza alcune proprietà e variabili utili alla comunicazione la quale infine inizia dopo che il client, recepito il SYN-Ack, manda a sua volta un messaggio di ACK.

La disconnessione può avvenire allo stesso modo quando la disconnessione è contemporanea, oppure può avvenire in 4 fasi. In entrambi i casi il messaggio si chiamerà FIN (e non SYNC) ma nel caso delle 4 fasi la disconnessione è operata da ciascuna parte in maniera indipendente ovvero ciascun lato manda un messaggio di FIN e attende l'ACK di risposta per poi considerare chiuso (dal suo lato) l'invio di messaggi; tuttavia, se non ha ricevuto il FIN dell'altro capo allora dovrà rimanere in attesa dei dati in arrivo da esso.

## 15 P2P CON SERVER CENTRALE, FLOODING, DHT

La comunicazione peer to peer può essere gestita in diversi modi:

- server centrale: prevede la presenza di un nodo che conserva le informazioni dei nodi della rete. ogni nodo deve accedervi per ottenere le informazioni

con server centrale, basato su flooding, basato su DHT.

- All'interno del sistema P2P gli utenti, chiamati appunto peers (nodi), possono connettersi tra di loro scambiandosi dati reciprocamente. Nel caso di sistema centralizzato, vi è la presenza di un server che ha il compito di conservare tutti i nomi dei file che vengono utilizzati dagli utenti. I vari utenti a loro volta possono trovare l'indirizzo di un altro peer tramite l'aiuto del server formulando delle query, senza l'opera di altri intermediari. Lo scambio coinvolge esclusivamente i due utenti che intendono scambiarsi i dati senza che il server conservi direttamente i files. Lo svantaggio è che ci può essere un unico punto di guasto, infatti se il server centrale si guasta non è più possibile accedere alle risorse o ricercarne altre.
- L'approccio basato su flooding prevede di avere un sistema completamente distribuito, a differenza del caso precedente che utilizzava un unico server altamente centralizzato. In questo approccio, ogni peer va ad indicizzare i file che rendono disponibili per la loro condivisione. Quando si vuole trasmettere un file, si utilizzano le connessioni TCP esistenti. Quando un generico peer riceve il messaggio lo va subito ad inoltrare sino a peer vicini se la richiesta non riguarda tale peer. Un messaggio di questo tipo viene propagato sul percorso inverso. Anche se da un lato questo evita il problema della bassa tolleranza ai guasti sollevata nel caso precedente, l'approccio basato su flooding introduce un grado di inefficienza per la ricerca delle risorse.
- L'approccio basato su DHT prevede di utilizzare delle tabelle hash distribuite, le quali vanno a partizionare un insieme di chiavi per l'accesso efficiente alle risorse. Le principali caratteristiche sono:
  - decentralizzazione: i nodi formano collettivamente il sistema senza alcun coordinamento centrale
  - scalabilità: il sistema è predisposto per un funzionamento efficiente anche con un numero elevatissimo di nodi
  - tolleranza ai guasti: il sistema dovrebbe riuscire a funzionare in modo affidabile anche al fronte di molti nodi che si aggiungono alla rete, molti nodi che escono dalla rete oppure che molti nodi sono soggetti a malfunzionamenti con elevata frequenza.

Il vantaggio principale è che ogni nodo ha bisogno di rimanere in contatto solo con un numero piccolo di altri nodi sulla rete. In questo modo la rete è sottoposta ad una quantità minima di lavoro per ogni modifica che avviene all'interno dei nodi che fanno parte della rete stessa.

16) Si descriva, anche mediante figure opportunamente commentate, le differenze tra crittografia a chiave pubblica e crittografia simmetrica.

- Con crittografia simmetrica o anche detta a chiave pubblica si intende un algoritmo di crittografia dove ogni attore coinvolto nella comunicazione ha una coppia di chiavi:

Chiave Pubblica che deve essere distribuita per diffondere l'informazione  
Chiave privata che deve essere mantenuta personale e segreta.

Il meccanismo si basa sul fatto che, con una delle due chiavi si cifra il messaggio e con l'altra viene decifrato. Di solito si usano chiavi di 1024-2048 bit e quindi sono molto lente.

Con crittografia simmetrica o anche detta a chiave privata, si intende una tecnica di cifratura dove la chiave per cifrare il testo è uguale a quella per decifrarlo. Tuttavia, si presuppone che le due parti stiano già in possesso delle chiavi. Il problema principale è proprio lo scambio di chiavi.

Di solito si usano chiavi di 64-128 bit e sono molto veloci.

17) Si descrivano gli attacchi su reti di tipo DOS, SYN Flooding e Smurfing.

- La sicurezza informatica è l'insieme dei mezzi e delle tecnologie tese alla protezione degli asset informatici. Nelle aziende le informazioni sono costituiti elementi tecnici, organizzativi, giuridici e umani. Per valutare la sicurezza è solitamente necessario individuare le minacce, le vulnerabilità e i rischi associati agli asset informatici, al fine di proteggerli da possibili attacchi che potrebbero compromettere dei dati. Infatti, per proteggersi dagli attacchi informatici si utilizzano varie tecnologie tra cui l'uso di un Firewall (Muro di fuoco) che è praticamente la combinazione di hardware e software che va ad isolare la rete interna dal resto di internet.
- Esso controlla l'accesso tra le risorse interne ed il mondo esterno, permettendo ad alcuni pacchetti di passare e bloccandone altri.
- Esso può essere di due tipi:
  - Filtraggio di pacchetto, a livello rete
  - Come Gateway a livello applicazione

Quando parliamo di un attacco di tipo "DOS" (Denial of Service) si va ad indicare un malfunzionamento dovuto ad un attacco informatico in cui si fanno esaurire deliberatamente le risorse di un sistema informatico che fornisce un servizio al Client, in modo da bloccarne l'erogazione. Gli attacchi vengono abitualmente attuati inviando molti pacchetti di richieste, di solito ad un server.

L'attacco tipo dos prevede l'inondazione di richieste per una risorsa causandone il ko.

## 16) CON SERVER CENTRALE, TOLERANZA AI GUASTI

La comunicazione peer to peer può essere gestita in diversi modi:

- server centrale: prevede la presenza di un nodo che conserva le informazioni dei nodi della rete. ogni nodo deve accedervi per ottenere le informazioni riguardo il nodo che presenta i dati ricercati. questa soluzione è poco scalabile ed è poco resistente ai guasti a causa della centralizzazione delle informazioni
- flooding: esso prevede che quando un nodo riceva un messaggio, cominci a mandare ai propri vicini l'informazione richiesta fino a quando essa non giunge al richiedente il quale manda indietro un messaggio di successo sempre in flooding. tale tecnica causa però un eccessivo consumo di risorse nonostante sia efficiente dal punto di vista della resistenza per la sua distribuzione
- dht: essa prevede l'uso di tabella di hashing contenute in ciascun nodo in modo che ognuno di essi debba comunicare solo coi nodi vicini e allo stesso tempo l'indicizzazione sia efficiente per via della struttura dati utilizzata. Inoltre, è una soluzione altamente scalabile e resistente ai guasti.

## 16) CRIPTOGRAFIA A CHIAVE PUBBLICA E SIMMETRICA



L'attacco di tipo dos prevede l'inondazione di richieste per una risorsa causandone il ko.

Il syn flooding di richieste di apertura di connessione.

Lo smurfing prevede un attacco di tipo dos dove l'indirizzo ip è camuffato con quello dell'host che si vuole attaccare che sarà inondato di risposte

TI DI INGEGNERIA  
INFORMATICA  
GAIA BERTOLINO

### → attacco socket

Quando parliamo di un attacco di tipo "SYN Flooding", stiamo parlando di un attacco di tipo DOS nel quale l'utente malevolo invia una serie di richieste SYN verso il server oggetto dell'attacco. L'attaccante inonda il server di pacchetti SYN (per inizio connessione) aventi indirizzi IP camuffati. Ovviamente, il server andrà a rispondere con pacchetti di tipo SYNACK per attivare la connessione. Ovviamente gli attaccanti non completeranno mai il Three Way Handshake e la connessione non verrà instaurata nonostante il server andrà a creare i socket e tutte le allocazioni utile.

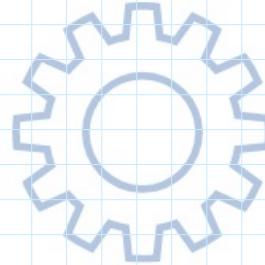
Quando parliamo di un attacco di tipo "SMURFING", stiamo parlando di un attacco di tipo DOS nel quale l'utente malevolo va a sovraccaricare i sistemi vittime con numerosi messaggi provenienti da molti altri nodi della rete, in risposta a false richieste di informazioni specificate per richieste della vittima stessa. L'attaccante invia un gran numero di pacchetti ICMP di tipo Echo Request ad un ampio numero di Host, con ovvia mente indirizzo sorgente del computer che si vuole attaccare. I pacchetti ICMP di tipo Echo Reply verranno quindi inviati alla vittima che si ritroverà bombardato di messaggi. L'attacco diventa ancora più grande se effettuato con indirizzi di Multicast.

18) Si descriva, anche mediante figure opportunamente commentate, quali sono i principi che Chord utilizza per memorizzare e per cercare una risorsa sulla rete.

In cuore di Chord, fornisce una distribuzione di calcolo veloce della funzione hash, mappando le chiavi verso i nodi corrispondenti. Chord assegna chiavi ai nodi con hashing consistente che ha diverse proprietà. Con alta probabilità, la funzione hash bilancia in carico tutti i nodi ricevono circa lo stesso numero di chiavi. Anche con alta probabilità, quando l' $N$ -esimo nodo si unisce o lascia la rete, solo una  $O(1/N)$  frazione di chiavi si muovono verso una nuova localizzazione - questo è chiaramente il minimo necessario per mantenere un carico bilanciato. Chord migliora la scalabilità dell'hashing consistente evitando il fatto che ogni nodo conosca ogni altro nodo. Un nodo Chord ha bisogno solo di una piccola quantità di informazioni di routing sugli altri nodi. Poiché queste informazioni sono distribuite, un nodo risolve la funzione hash comunicando con altri nodi. In una rete di  $N$ -nodi, ogni nodo mantiene informazioni su  $O(\log N)$  altri nodi, e una ricerca richiede  $O(\log N)$  messaggi.

Chord è un protocollo che permette di regolare l'assegnazione delle chiavi di una dht alle risorse. Esso fa uso di hashing consistente il che vuol dire che le chiavi risultano essere bilanciate all'interno della rete dei nodi e la mancanza di uno di essi causa il cambiamento di un numero limitato di chiavi nella tabella di hashing.

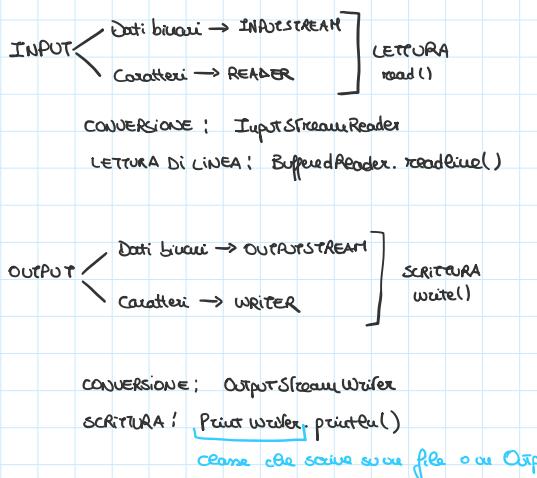
Ogni nodo mantiene le informazioni sul nodo successivo; i peer vengono disposti topologicamente lungo un anello in base ad un identificatore e viene utilizzata una funzione successor( $k$ ) che individua il primo successore di  $k$  nell'anello (che non per forza corrisponde numericamente). Quando un utente deve cercare una risorsa calcola l'hash del nome della risorsa e individua l'ip del peer che la contiene attraverso la funzione successor(hash(nome risorsa)). Per raggiungerlo, un pacchetto viene inoltrato con le informazioni necessarie fin quando non raggiunge il peer contenente le informazioni richieste.



# APPUNTI DI INGEGNERIA INFORMATICA

GAIA BERTOLINO

## CLASSI



## METODI INDIRIZZI

- getByName (host)
 

L'IP dell'host sapendo il nome
- getAllByName (host)
 

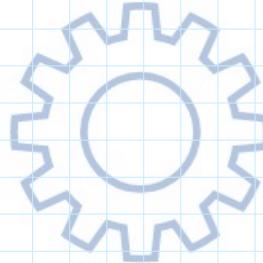
Tutti gli IP di un host
- getLocalHost()
 

IP del localhost
- getAddress()
 

IP sottoforma di array di byte
- getInetAddress()
 

IP sottoforma di stringa
- getHostByName()
 

L'hostname di un indirizzo IP



# APPUNTI DI INGEGNERIA INFORMATICA

GAIA BERTOLINO

## METODI SOCKET

- Socket (host, port)
 

HOSTNAME o IP (STRINGS)
- create a socket
- Socket (address, port)
 

INETADDRESS
- close()
 

Chiude il socket
- getInetAddress()
 

Indirizzo a cui è connesso il socket
- getPort()
 

Porta remota del socket
- getLocalPort()
 

Porta locale del socket
- getInputStream()
 

Legge i dati in ingresso; usato per leggere dal socket
- getOutputStream()
 

Legge i dati in uscita; usato per scrivere nel socket
- setSoTimeout (timeout)
 

Imposta un timer di attesa. Scaduto il timer solleva una eccezione
- toString()
 

Rappresentazione completa

### METODI SERVER SOCKET

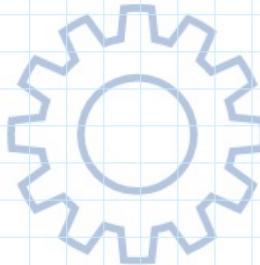
- ServerSocket (port)
- accept()
- Metodo che rimane in attesa di una connessione
- close()
- getInetAddress()
- getLocalPort()

### METODI PRINTWRITER

- flush()
  - COSTRUTTORI
- PrintWriter (OutputStream out)
- PrintWriter (OutputStream out, boolean autoFlush)

### DATAGRAMMI

- DatagramSocket()
  - DatagramSocket (port)
  - DatagramPacket (byte[], lenght) → RICEZIONE
  - DatagramPacket (byte[], lenght, address, port) → INVIO
  - getAddress()
  - getPort()
  - receive (DatagramPacket)
  - send (DatagramPacket)
  - close()
- ↓  
InetAddress a = packet.getAddress()  
(per rispondere)



### METODO MULTICAST

- MulticastSocket (port)
- joinGroup (InetAddress)
- leaveGroup (InetAddress)

# APPUNTI DI INGEGNERIA INFORMATICA

GAIA BERTOLINO

### TCP

- 1) I server devono
  - a. Implementare un ServerSocket e un Socket che memorizza l'invocazione del metodo accept del ServerSocket  
Se hanno più canali di connessione, per ognuno devono creare un thread attraverso una inner class. In questo caso i PrintWriter e il BufferedReader devono essere implementati nel run e il ciclo di while deve comprendere lo start del thread. Questo perché il server deve rimanere in ascolto all'infinito fino alla fine del thread. Inoltre, il server prevede che il while true sia solo nella zona di lettura delle stringhe in input tramite il comando readLine()
    - Differenza fra invio singolo e server sempre in ascolto:
      - Sempre in ascolto con Thread:  
ServerSocket s = new ServerSocket(80);  
while (true){  
    Socket in = new Socket(s.accept());  
    MyThread t = new MyThread(in);  
    t.start();  
} // BufferedReader e PrintWriter implementati SENZA ciclo di while nel run del thread
      - Sempre in ascolto senza Thread:  
ServerSocket s = new ServerSocket(80);  
while (true){  
    Socket in = new Socket(s.accept());  
    BufferedReader r = new BufferedReader(new InputStreamReader(in.getInputStream()));

```

PrintWriter w = new PrintWriter(in.getOutputStream(), true);
...
} // Un altro ciclo di while serve a gestire la lettura

□ Connessione singola
ServerSocket s = new ServerSocket(s);
Socket in = new Socket(s.accept());
BufferedReader r = new BufferedReader(new InputStreamReader(in.getInputStream()));
PrintWriter w = new PrintWriter(in.getOutputStream(), true);

```

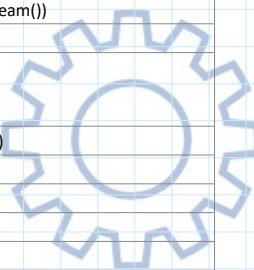
- 2) Le classi che implementano gli oggetti scambiati nella comunicazioni devono implementare Serializable
- 3) Nella catch andrebbe messa la close del socket

### UDP

- 1) Bisogna istanziare il DatagramSocket, creare il DatagramPacket con il costruttore della ricezione e invocare il metodo di receive sul DatagramSocket passandogli come argomento il DatagramPacket. Una volta ricevuto, si elabora il messaggio e la risposta prevede che il DatagramPacket venga inizializzato col costruttore di invio e venga invocato il metodo send sul DatagramSocket con argomento il DatagramPacket. Il client prevede l'inizializzazione al contrario.

### METODI DA USARE

TCP:	ServerSocket socket = new ServerSocket(InetAddressServer, portaServer);  BufferedReader r = new BufferedReader(new InputStreamReader(socket.getInputStream()))  r.readLine()  PrintWriter w = new PrintWriter(socket.getOutputStream())  w.println(stringa)  while (true) { Socket s = socket.accept() }
MULTICAST	MulticastSocket s = new MulticastSocket(portaLocale)  socket.joinGroup(gruppo) byte[] buffer = new byte[dimensione]
UDP:	buffer = dati.getBytes()  DatagramSocket socket = new DatagramSocket(buffer, lunghezzaBuffer, gruppo, portaLocale) INVIO  socket.send(pacchetto) DatagramSocket pacchetto = new DatagramSocket(buffer, lunghezzaBuffer)  socket.receive(pacchetto)
INETADDRESS	InetAddress gruppo = new InetAddress.getByName("123.45.67.89")
CALENDAR	Calendar now = Calendar.getInstance();  serverSocket.setSoTimeout((int).limite.getTimeInMillis() - now.getTimeInMillis())



### TCP

#### Cient

**Socket(String host, int port):** crea un socket e lo connette all'host ed alla porta specificati; host è un hostname (per es. "www.unical.it") oppure un indirizzo IP (per es. "160.97.4.26").

**Socket(InetAddress address, int port):** crea un socket e lo connette all'indirizzo ed alla porta specificati.

**void close():** chiude il socket.

**InetAddress getInetAddress():** restituisce l'indirizzo a cui è connesso il socket.

**int getPort():** restituisce la porta remota a cui è connesso il socket.

**int getLocalPort():** restituisce la porta locale del socket.

**InputStream getInputStream():** restituisce lo stream di input del socket; questo stream è utilizzato per leggere i dati provenienti dal socket.

**OutputStream getOutputStream():** restituisce lo stream di output del socket; questo stream è utilizzato per scrivere dati nel socket.

**void setSoTimeout(int timeout):** imposta il timeout per operazioni di lettura dal socket; se il tempo specificato trascorre genera una InterruptedIOException.

**String toString():** restituisce una rappresentazione del socket del tipo "Socket[addr=hostname/192.168.90.82,port=3575,localport=1026]"

### SERWER SOCKET

#### Server

**ServerSocket(int port):** crea un server socket che controlla una porta.

**Socket accept():** rimane in attesa di una connessione, e restituisce un socket tramite il quale si effettua la comunicazione.

**void close():** chiude il server socket.

**InetAddress getInetAddress():** restituisce l'indirizzo locale di questo server socket.

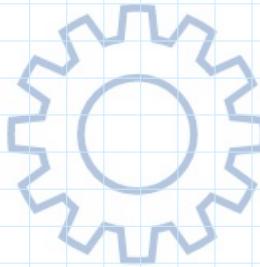
**int getLocalPort():** restituisce la porta locale di questo server socket.

### UDP

### MULTICAST

- **DatagramSocket () throws SocketException**  
Crea un DatagramSocket e lo collega alla porta qualsiasi sulla macchina locale. Solleva eccezione se non si hanno i permessi necessari.
- **DatagramSocket (int port)**  
Crea un DatagramSocket e lo collega alla porta specificata sulla macchina locale.
- **void receive (DatagramPacket p)**  
Riceve un DatagramPacket da questo socket.
- **void send (DatagramPacket p)**  
Invia un DatagramPacket su questo socket.
- **void close ()**  
Chiude questo DatagramSocket.
  
- **DatagramPacket (byte[] buf, int length)**  
Crea un DatagramPacket per ricevere pacchetti di lunghezza length.
- **DatagramPacket (byte[] buf, int length, InetAddress address, int port)**  
Crea un DatagramPacket per inviare pacchetti di lunghezza length all'host ed alla porta specificati.
- **InetAddress getAddress ()**  
Restituisce l'indirizzo IP della macchina alla quale questo DatagramPacket deve essere inviato o da cui è stato ricevuto.
- **int getPort ()**  
Restituisce la porta della macchina alla quale questo DatagramPacket deve essere inviato o da cui è stato ricevuto.

- **MulticastSocket (int port)**  
Crea un MulticastSocket e lo collega alla porta specificata sulla macchina locale.
- **void joinGroup (InetAddress mcastaddr)**  
Si collega ad un multicast group.
- **void leaveGroup (InetAddress mcastaddr)**  
Abbandona un multicast group.
- **void receive (DatagramPacket p) 230.0.0.1**  
Riceve un DatagramPacket da questo socket.
- **void send (DatagramPacket p)**  
Riceve un DatagramPacket da questo socket.
- **void close ()**  
Chiude questo MulticastSocket.



## APPUNTI DI INGEGNERIA INFORMATICA

GAIA BERTOLINO

**TCP** → **SERVER SOCKET**

↑  
inizio

↑  
fine

SI DEFINISCE UN SERVER SOCKET E UN PRINTWRITER E UN BUFFEREDWRITER CHE GESTISCONO LA COMUNICAZIONE.

UN CICLO DI WHILE GESTISCE LA RICEZIONE CONTINUA

L'INIZIO SI FA TRAMITE UNA PRINTLN()

```
public static void main(String[] args) {
    try {
        ServerSocket s = new ServerSocket(8190); → SOCKET LATO SERVER
        Socket incoming = s.accept(); // stop in attesa di una connessione → METODO BLOCCANTE

        BufferedReader in = new BufferedReader (new InputStreamReader(incoming.getInputStream())); → READER PER LE COMUNICAZIONI IN INGRESSO SUL CANALE
        PrintWriter out = new PrintWriter (incoming.getOutputStream(), true /* autoFlush */); → WRITER PER LE COMUNICAZIONI IN USCITA SUL CANALE

        ↓
        CICLO
        boolean done = false; → COMMUNICAZIONE RANDATA AL CLIENT
        while (!done) {
            String line = in.readLine(); → LETTURA LINEA IN INGRESSO

            if (line == null) { → USCITA DAL CICLO
                done = true; } → (QUANDO L'INPUT È VUOTO)
            else {
                out.println("Echo: " + line); → SCRITTURA IN USCITA
                if (line.trim().equals("BYE")) done = true; } → INTERRUZIONE DEL CICLO
                (QUANDO RICEVO BYE)
            } // while
        incoming.close(); }
        catch (Exception e) { System.err.println(e); }
    } // main
}
```

READER PER LE COMUNICAZIONI IN INGRESSO SUL CANALE  
WRITER PER LE COMUNICAZIONI IN USCITA SUL CANALE

#### METODI LATO SERVER (NO THREAD)

- SERVER SOCKET
- SOCKET `Socket incoming = s.accept();`
- BUFFEREDREADER `BufferedReader in = new BufferedReader (new InputStreamReader(incoming.getInputStream()));`
- PRINTWRITER `PrintWriter out = new PrintWriter (incoming.getOutputStream(), true /* autoFlush */);`
- MESSAGGIO `out.println("Hello! Enter BYE to exit.");`
- CICLO DI WHILE
- READLINE DI INPUT `String line = in.readLine();`
- CLOSE



# APPUNTI DI INGEGNERIA INFORMATICA

GAIA BERTOLINO

#### CONNESSIONI MULTIPLE

USO DI THREAD CHE IMPLEMENTANO LA GESTIONE DELL'INTERAZIONE

IL BUFFEREDREADER E IL PRINTWRITER SONO ISTANZIATI NEL RUN DEL THREAD

```
public class ConnessioniMultiple {

    // SOCKET E CICLI VANNO SPECIFICATI IN UN MAIN. TUTTI GLI ALTRI METODI AL DI FUORI
    public static void main(String[] args) {
        ServerSocket s = new ServerSocket(1234);
        while (true) { // Nel ciclo di while si opera l'accept e si fanno partire i thread
            Socket incoming = s.accept(); // Chiamata bloccante in attesa di una connessione
            ThreadedHandler t = new ThreadedHandler(incoming);
            t.start();
        }
    }

    // INNER CLASS PER I THREAD LANCIATI
    class ThreadedHandler extends Thread {

        Socket incoming;
        public ThreadedHandler(Socket s) {
            incoming = s;
        }

        public void run () {
            try {
                BufferedReader in = new BufferedReader (new InputStreamReader(incoming.getInputStream()));
                PrintWriter out = new PrintWriter (incoming.getOutputStream(), true);
                incoming.close();
            } catch (Exception e) {}
        }
    }
}
```

### UDP → DATAGRAM SOCKET

IL SOCKET È DI TIPO DATAGRAMSOCKET E IL MESSAGGIO VIENE MANDATO O RICEVUTO TRAMITE UN DATAGRAMPACKET. LA RICEZIONE SI FA COL METODO SOCKET. RECEIVE(PACKET) MENTRE L'INVIIO CON SOCKET. SEND(PACKET)

#### SERWER

```
public class TimeServer {  
    public static void main(String[] args) {  
        DatagramSocket socket = null;  
        try {  
            socket = new DatagramSocket(3575);  
            int n = 1;  
            while (n <= 10) {  
                byte[] buffer = new byte[256];  
                DatagramPacket dati = new DatagramPacket(buffer, buffer.length); // Connessione di ricezione  
                socket.receive(dati); // Metodo bloccante in attesa dell'arrivo di dati in ingresso.  
                // In questo caso corrisponde ad una richiesta qualsiasi  
  
                INFO DEL RITTORENTE {  
                    InetAddress address = dati.getAddress(); // Oggetto indirizzo da passare al messaggio in uscita  
                    int port = dati.getPort();  
                    dati = new DatagramPacket(buffer, buffer.length, address, port); // Connessione di risposta  
                    socket.send(dati);  
                    n++;  
                }  
                socket.close();  
            } catch (IOException e) { e.printStackTrace(); socket.close(); }  
        }  
    }  
}
```



#### CLIENT

```
public class TimeClient {  
    public static void main(String[] args) throws IOException {  
        String hostname = "localhost";  
        DatagramSocket socket = new DatagramSocket(); // invia la richiesta  
  
        byte[] buffer = new byte[256];  
        InetAddress address = InetAddress.getByName(hostname);  
  
        DatagramPacket dati = new DatagramPacket(buffer, buffer.length, address, 3575); // Pacchetto di invio  
        socket.send(dati); // La send si fa sul socket col pacchetto di dati  
        dati = new DatagramPacket(buffer, buffer.length); // Pacchetto di ricezione della risposta  
        socket.receive(dati); // visualizza la risposta  
  
        String received = new String(dati.getData());  
        socket.close();  
    }  
}
```

### MULTICAST

IL SERWER MANDA UN MESSAGGIO TRAMITE UN MULTICAST SOCKET  
USA UN DATAGRAMSOCKET (PERCHÉ È UNA CONNESSIONE UDP)

#### SERWER

```

public class MulticastTimeServer {
    public static void main(String[] args) {
        MulticastSocket socket = null;
        try {
            socket = new MulticastSocket(3575); → PORTA DEL SERVER
            while (true) {
                byte[] buffer = new byte[256];
                // Non prevede il metodo di receive che rimane in attesa di ricezione
                String dati = new Date().toString();
                buffer = dati.getBytes();

                InetAddress groupAddress = InetAddress.getByName("230.0.0.1"); ← Indirizzi di tutti i PC della LAN
                DatagramPacket packet = new DatagramPacket(buffer, buffer.length, groupAddress, 3575); → PACCHETTO DI INIZIO
                socket.send(packet);
            }
            socket.close();
        } catch (Exception e) { e.printStackTrace(); socket.close(); }
    }
}

```

### CURRENT

```

public class MulticastTimeClient {
    @SuppressWarnings("deprecation")
    public static void main(String[] args) throws IOException {
        MulticastSocket socket = new MulticastSocket(3575);
        InetAddress group = InetAddress.getByName("230.0.0.1");
        socket.joinGroup(group); → SI UNISCE AL GRUPPO
        DatagramPacket packet;

        for (int i = 0; i < 100; i++) {
            byte[] buf = new byte[256];
            packet = new DatagramPacket(buf, buf.length);
            socket.receive(packet);
            String received = new String(packet.getData());
        }
        socket.leaveGroup(group); → LASCIA IL GRUPPO
        socket.close(); → CHIOSURA SOCKET
    }
}

```



# APPUNTI DI INGEGNERIA INFORMATICA

GAIA BERTOLINO