Reti e Laboratorio III
Modulo Laboratorio III
AA. 2023-2024

docente: Laura Ricci

laura.ricci@unipi.it

Correzione Assignment 10
”Java Pinger Server”
11/12/2023

ASSIGNMENT 10: PING CLIENT

- PING è una utility per la valutazione delle performance della rete utilizzata per verificare la raggiungibilità di un host su una rete IP e per misurare il round trip time (RTT) per i messaggi spediti da un host mittente verso un host destinazione.
- lo scopo di questo assignment è quello di implementare un server PING ed un corrispondente client PING che consenta al client di misurare il suo RTT verso il server.
- la funzionalità fornita da questi programmi deve essere simile a quella della utility PING disponibile in tutti i moderni sistemi operativi. La differenza fondamentale è che si utilizza UDP per la comunicazione tra client e server, invece del protocollo ICMP (Internet Control Message Protocol).
- inoltre, poichè l'esecuzione dei programmi avverrà su un solo host o sulla rete locale ed in entrambe i casi sia la latenza che la perdita di pacchetti risultano trascurabili, il server deve introdurre un ritardo artificiale ed ignorare alcune richieste per simulare la perdita di pacchetti

ASSIGNMENT 10: PING CLIENT

- accetta due argomenti da linea di comando: nome e porta del server. Se uno o più argomenti risultano scorretti, il client termina, dopo aver stampato un messaggio di errore del tipo ERR -arg x, dove x è il numero dell'argomento.
- utilizza una comunicazione UDP per comunicare con il server ed invia 10 messaggi al server, con il seguente formato:

  `PING segno timestamp`

  in cui *segno* è il numero di sequenza del PING (tra 0-9) ed il timestamp (in millisecondi) indica quando il messaggio è stato inviato
- non invia un nuovo PING fino che non ha ricevuto l'eco del PING precedente, oppure è scaduto un timeout.

ASSIGNMENT 10: PING CLIENT

- stampa ogni messaggio spedito al server ed il RTT del ping oppure un * se la risposta non è stata ricevuta entro 2 secondi
- dopo che ha ricevuto la decima risposta (o dopo il suo timeout), il client stampa un riassunto simile a quello stampato dal PING UNIX

---- PING Statistics ----
10 packets transmitted, 7 packets received, 30% packet loss
round-trip (ms) min/avg/max = 63/190.29/290

- il RTT medio è stampato con 2 cifre dopo la virgola

ASSIGNMENT 10: PING SERVER

- è essenzialmente un echo server: rimanda al mittente qualsiasi dato riceve

- accetta un argomento da linea di comando: la porta, che è quella su cui è attivo il server + un argomento opzionale, il seed, un valore long utilizzato per la generazione di latenze e perdita di pacchetti. Se uno qualunque degli argomenti è scorretto, stampa un messaggio di errore del tipo ERR -arg x, dove x è il numero dell'argomento.

- dopo aver ricevuto un PING, il server determina se ignorare il pacchetto (simulandone la perdita) o effettuarne l'eco. La probabilità di perdita di pacchetti di default è del 25%.

- se decide di effettuare l'eco del PING, il server attende un intervallo di tempo casuale per simulare la latenza di rete

- stampa l'indirizzo IP e la porta del client, il messaggio di PING e l'azione intrapresa dal server in seguito alla sua ricezione (PING non inviato, oppure PING ritardato di x ms).

ASSIGNMENT 10: PING CLIENT

```
public class Client {
    // Numero di richieste (pacchetti UDP) inviate dal client.
    public static final int numPing = 10;
    // Tempo massimo di attesa prima di considerare un pacchetto come perso.
    public static final long timeout = 2000;
    // Dimensione del buffer usato per la ricezione.
    public static final int bufferSize = 1024;
    // Array di indicatori contenenti informazioni sullo stato dei pacchetti inviati.
    public static PacketMonitor[] monitors = new PacketMonitor[numPing];
    // Parametri usati per il calcolo delle statistiche.
    public static long rttTot = 0;
    public static long rttMin = Long.MAX_VALUE;
    public static long rttMax = Long.MIN_VALUE;
    public static int numReceived = 0;
```

ASSIGNMENT 10: PING CLIENT

```
public static void main(String[] args) {
    if (args.length < 2) {
        System.err.println("Esegui come: Client <indirizzo> <porta>");
        System.exit(1);
    }
    String address = args[0];
    int port = Integer.parseInt(args[1]);
    // Inizializzo l'array con i monitor.
    for (int i = 0; i < numPing; i++) monitors[i] = new PacketMonitor();
    // Inizializzo la socket per inviare le richieste.
    // La socket viene chiusa automaticamente all'uscita dal blocco.
    // Di conseguenza, anche il thread listener che si trova bloccato
    // sulla receive() puo' terminare.
    try (DatagramSocket socket = new DatagramSocket()) {
        // Avvio il thread listener che attende i pacchetti di risposta.
        Thread listener=new Thread(new ClientListener(socket, monitors, bufferSize));
        listener.start();
        // Ottengo l'indirizzo del server.
        InetAddress addr = InetAddress.getByName(address);
```

ASSIGNMENT 10: PING CLIENT

```
for (int i = 0; i < numPing; i++) {
    long startTime = System.currentTimeMillis();
    String contentStr = String.format("PING %d %d", i, startTime);
    byte[] content = contentStr.getBytes();
    DatagramPacket pkt = new DatagramPacket(content, content.length, addr, port);
    // Invio il pacchetto.
    socket.send(pkt);
    System.out.printf("[CLIENT] Inviato: id=%d\n", i);
    // Attendo l'arrivo sul monitor i-esimo per un massimo di `timeout` millisecondi.
    if (monitors[i].get(timeout)) {
        System.out.printf("[CLIENT] Ricevuto: id=%d\n", i);
        long rtt = System.currentTimeMillis() - startTime;
        rttTot += rtt;
        rttMin = Long.min(rttMin, rtt);
        rttMax = Long.max(rttMax, rtt);
        numReceived++;
        }
        // Se sono qui, significa che non ho ricevuto il pacchetto
    else System.out.printf("[CLIENT] Non ricevuto: id=%d\n", i); }
```

ASSIGNMENT 10: PING CLIENT - STATS

```
/**
* Stampa su schermo le statistiche relative ai pacchetti.
*/
private static void printStats() {
    System.out.printf("---- PING Statistics ----\n%d packets transmitted, %d packets
received, %d%% packet loss\n",
    numPing, numReceived, 100 * (numPing - numReceived) / numPing);
    if (numReceived != 0) {
        System.out.printf("round-trip (ms) min/avg/max = %d / %.2f / %d\n",
        rttMin, ((double) rttTot / (double) numReceived), rttMax);
    }
}
```

ASSIGNMENT 10: CLIENT THREAD

```
public class ClientListener implements Runnable {
    private DatagramSocket socket;
    private PacketMonitor[] monitors;
    private int bufferSize;

    /**
     * Costruttore della classe ClientListener.
     * @param socket riferimento alla DatagramSocket su cui arrivano i pacchetti
     * @param monitors riferimento all'array contenente i monitor per i pacchetti
     * @param bufferSize dimensione del buffer usato per la ricezione dei pacchetti
     */
    public ClientListener(DatagramSocket socket, PacketMonitor[] monitors, int
bufferSize) {
        this.socket = socket;
        this.monitors = monitors;
        this.bufferSize = bufferSize;
    }
```

ASSIGNMENT 10: CLIENT THREAD

```
public void run() {
    System.out.println("[CLIENT] Listener avviato");
    while (true) {
    byte[] buffer = new byte[bufferSize];
    DatagramPacket rpkt = new DatagramPacket(buffer, buffer.length);
    try {socket.receive(rpkt);}
    // Questa Exception viene sollevata quando la socket viene chiusa dall'esterno
    // NOTA: oltre al timeout, questo e' l'unico modo per sbloccare
    // un thread che e' bloccato sulla receive().
    catch (IOException e) {break;}
    // Ricavo l'identificativo del pacchetto appena arrivato.
    int i = Integer.parseInt((new String(rpkt.getData())).split(" "))[1]);
    // Aggiorno l'indicatore i-esimo segnalando al thread in attesa
    // l'arrivo del pacchetto con questo identificativo.
    if (0 <= i && i < monitors.length) monitors[i].set();
    }
    System.out.println("[CLIENT] Listener terminato");
}
```

ASSIGNMENT 10: CLIENT - PACKET MONITOR

```
/**
 * La classe PacketMonitor rappresenta lo stato corrente della ricezione
 * di un determinato pacchetto. La classe contiene una variabile booleana
 * che vale true se e solo se il pacchetto e' stato ricevuto.
 * La classe contiene inoltre due metodi synchronized, denominati rispettivamente
 * set() e get(). Il primo viene invocato dal thread ClientListener per
 * settare a true la variabile indicatrice e segnalare l'arrivo di
 * un pacchetto. Il secondo invece viene eseguito dal thread main
 * della classe Client per attendere l'arrivo del pacchetto.
 * I due metodi usano i meccanismi di wait() e notify() invocati
 * sull'oggetto corrente per sospendere e risvegliare i thread.
 * In particolare, la wait() che usiamo in questa soluzione e' una
 * variante di quella vista a lezione, nel senso che prevede un ulteriore
 * parametro, detto timeout, che consente di specificare il tempo
 * massimo di attesa. In questo modo, il thread che esegue la wait()
 * rimane quindi sospeso finche' un altro thread non esegue la notify()
 * oppure fino allo scadere del timeout.
 * */
```

ASSIGNMENT 10: CLIENT - PACKET MONITOR

```
public class PacketMonitor {
    /**
     * Variabile booleana che vale true se e solo se il pacchetto e' stato
ricevuto.
     */
    boolean received = false;

    /**
     * Imposta a true il flag per l'arrivo del pacchetto,
     * risvegliando anche eventuali thread in attesa.
     */
    public synchronized void set() {
        received = true;
        notify();
    }
```

ASSIGNMENT 10: CLIENT - PACKET MONITOR

```
/**
* Metodo bloccante che restituisce il valore del flag per l'arrivo del pacchetto.
* Il thread che lo invoca resta sospeso al piu' per timeout millisecondi. */
public synchronized boolean get(long timeout) throws InterruptedException {
    long expiration = System.currentTimeMillis() + timeout;
    while (!received) {
        // Controllo quanto mi rimane da aspettare.
        long remaining = expiration - System.currentTimeMillis();
        // Se il valore dei millisecondi da attendere e' <= 0,
        // significa che il tempo massimo e' scaduto.
        if (remaining <= 0) break;
            // Altrimenti, mi metto in attesa sul monitor per il
            // tempo che mi rimane, usando una wait() con un timeout.
            // Si tratta di una variante della wait() vista a lezione.
            // Come al solito, se un thread mi risveglia con una notify(),
            // controllo la guardia del while e verifico se e' arrivato
            // il pacchetto che aspettavo. In tal caso, posso terminare.
            wait(remaining);
        }
        return received;
    }
```

ASSIGNMENT 10: SERVER

```
public class Server {
    // Porta su cui attendere l'arrivo dei pacchetti.
    public static int port;
    // Parametro per il generatore pseudocasuale.
    public static long seed;
    // Dimensione del buffer per la ricezione e l'invio dei pacchetti.
    public static final int bufferSize = 1024;
    // Tempo massimo di attesa (in millisecondi) prima di inviare una risposta.
    public static final int maxDelay = 5000;
    // Pool di thread per servire le richieste in arrivo.
    public static ExecutorService pool = Executors.newCachedThreadPool();
```

ASSIGNMENT 10: SERVER

```
public static void main(final String[] array) {
        // Leggo i parametri (porta e seed) da riga di comando.
        // Il parametro relativo al seed e' opzionale.
        if (array.length < 1) {
            System.err.println("Esegui come: Server <porta> [<seed>]");
            System.exit(1);
        }
        // Eseguo il parsing dei parametri.
        port = Integer.parseInt(array[0]);
        if (array.length > 1) seed = Long.parseLong(array[1]);
        // Se il seed non e' stato fornito all'avvio, uso il timestamp corrente.
        else seed = System.currentTimeMillis();
        // Apro la socket per la comunicazione.
```

ASSIGNMENT 10: SERVER

```
try (DatagramSocket socket = new DatagramSocket(port)) {
    // Inizializzo il generatore di numeri pseudocasuali
    // usando il seed passato dall'utente, altrimenti il timestamp
    Random random = new Random(seed);
    System.out.printf("[SERVER] pronto sulla porta %s\n", port);
    // Entro in un ciclo infinito in cui ricevo un pacchetto
    // dal client e lo gestisco in un thread apposito.
    while (true) {
        byte[] buffer = new byte[bufferSize];
        DatagramPacket pkt = new DatagramPacket(buffer, buffer.length);
        socket.receive(pkt);
        pool.execute(new ServerWorker(socket, pkt, random.nextLong(), bufferSize,
maxDelay));
    }
}
catch (Exception e) {
    System.err.printf("[SERVER] Errore: %s\n", e.getMessage());
    e.printStackTrace();
}
finally { pool.shutdown(); }
```

ASSIGNMENT 10: SERVER WORKER

```
public class ServerWorker implements Runnable {
    // Dimensione (in byte) del buffer usato per l'invio dei pacchetti.
    public final int bufferSize;
    // Tempo massimo di attesa (in millisecondi) prima di inviare una risposta.
    public final int maxDelay;
    // Socket per comunicare con il client.
    private DatagramSocket socket;
    // Pacchetto ricevuto dal client.
    private DatagramPacket packet;
    // Generatore di numeri pseudocasuali.
    private Random gen;

    public ServerWorker(DatagramSocket socket, DatagramPacket packet, long seed,
int bufferSize, int maxDelay) {
        this.socket = socket;
        this.packet = packet;
        this.gen = new Random(seed);
        this.bufferSize = bufferSize;
        this.maxDelay = maxDelay;
    }
```

ASSIGNMENT 10: SERVER WORKER

```
public void run() {
    // Estraggo l'identificativo associato al pacchetto ricevuto.
    int id = Integer.parseInt(new String(packet.getData()).split(" ")[1]);
    System.out.printf("[WORKER] Ricevuto: id=%d\n", id);
    // Decido se scartare o tenere il pacchetto. Per farlo, genero
    // un intero nell'intervallo [1, 100] e controllo se Ã¨ minore o
    // uguale a 25. In caso affermativo, scarto il pacchetto.
    if (gen.nextInt(100) + 1 <= 25) {
        System.out.printf("[WORKER] Scartato: id=%d\n", id);
        return;
    }
    // A questo punto, decido quanto attendere prima di inviare il
    // pacchetto di risposta al client. Scelgo un tempo casuale
    // nell'intervallo [0, maxDelay].
    long delay = gen.nextInt(maxDelay + 1);
    try {Thread.sleep(delay);}
    catch (InterruptedException e) {
        System.err.println("[WORKER] Interruzione durante l'attesa");
        return;
    }
```

ASSIGNMENT 10: SERVER WORKER

```
        // Preparo il pacchetto di risposta e lo invio.
        long timestamp = System.currentTimeMillis();
        byte[] content = String.format("PONG %d %d", id, timestamp).getBytes();
        DatagramPacket reply = new DatagramPacket(
            content, content.length, packet.getAddress(), packet.getPort()
        );
        try {
            socket.send(reply);
            System.out.printf("[WORKER] Inviato: id=%d dopo %d ms\n", id, delay);
        }
        catch (IOException e) {
            System.err.printf("[WORKER] Errore di I/O: %s\n", e.getMessage());
            e.printStackTrace();
        }
    }
```