



Spring 2020

**Computer networks (CS- 330)**  
**Section: 171**

**Project**  
**Ping-pong using UDP**

**Submitted By**

Amr AL Jamal	(438021843)
Anas Markhouss	(438021838)
Ibrahim Faiz Babgi	(438011611)
AbdulrahmanSaadAlsakran	(438011736)
Abdullah Sulaiman Alsudais	(438012355)

**Supervisor**

Dr.Eisa Aleisa

**Date: April 8, 2020**

## Table of Contents

1.Intoduction .....	3
2.Ping-Pong.....	3
Part1: standard ping .....	3
Part2: UDP Ping .....	5
3.Project code.....	7
3.1 Client .....	7
3.2 Server .....	8
3.3 Output for Server .....	8
3.4 Output for Client .....	9
4.References.....	9

# 1.Introduction

To be able to send and receive from different devices a connection is required, this project consist of two parts, the first one will show how to know if the server (Receiver) **is connected or can be reached** by sending ping message using Command Prompt, the second one is an **UDP Ping-Pong application** which consist of client/server classes, the client send 10 ping messages to server that replay with pong messages, considering the timeout and that a packet may get lost, also calculating the RTT (Round Trip Time) for each packet.

## 2.Ping-Pong

### Part1: standard ping

To check that a server can receive messages from client using command prompt several steps are taking:

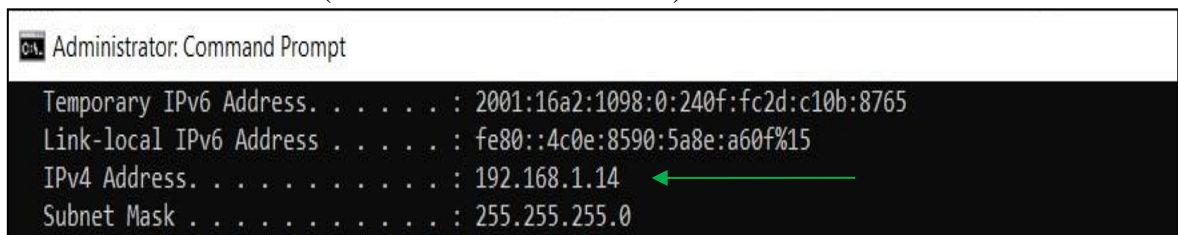
- 1- Open the cmd on the server device and enter the following command: **ipconfig** which display all the IP address information of the device.



```
Administrator: Command Prompt
Microsoft Windows [Version 10.0.18362.720]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\windows\system32>ipconfig
```

- 2- Look for IPv4 address (in our case **192.168.1.14**).



```
Administrator: Command Prompt

Temporary IPv6 Address. . . . . : 2001:16a2:1098:0:240f:fc2d:c10b:8765
Link-local IPv6 Address . . . . . : fe80::4c0e:8590:5a8e:a60f%15
IPv4 Address. . . . . : 192.168.1.14
Subnet Mask . . . . . : 255.255.255.0
```

- 3- Repeat the steps for the client device, then look for IPv4 address (in our case 192.168.1.11).

```
C:\ Select Command Prompt
Temporary IPv6 Address. . . . . : 2001:16a2:1098:0:78e8:53e0:49b8:799a
Link-local IPv6 Address . . . . . : fe80::1438:d80:7226:8e60%11
IPv4 Address. . . . . : 192.168.1.11
Subnet Mask . . . . . : 255.255.255.0
```

- 4- From the client device enter the following command: `ping 192.168.1.14` (this command will send 4 packets to the specify IP address).

```
C:\ Command Prompt
C:\Users\TOSHIBA>ping 192.168.1.14
```

- ✦ If the server is reachable or connected replay message is received with packets statistics (send, receive, loss).

```
C:\ Command Prompt
Pinging 192.168.1.14 with 32 bytes of data:
Reply from 192.168.1.14: bytes=32 time=6ms TTL=128
Reply from 192.168.1.14: bytes=32 time=3ms TTL=128
Reply from 192.168.1.14: bytes=32 time=4ms TTL=128
Reply from 192.168.1.14: bytes=32 time=19ms TTL=128

Ping statistics for 192.168.1.14:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 3ms, Maximum = 19ms, Average = 8ms
```

- ✦ If the server is not reachable or not connected there will be a timeout for all packets.

```
C:\ Command Prompt
C:\Users\TOSHIBA>ping 192.168.1.14

Pinging 192.168.1.14 with 32 bytes of data:
Reply from 192.168.1.11: Destination host unreachable.
Reply from 192.168.1.11: Destination host unreachable.
Reply from 192.168.1.11: Destination host unreachable.
Reply from 192.168.1.11: Destination host unreachable.

Ping statistics for 192.168.1.14:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
```

## Part2: UDP Ping

### Client

#### InetAddress

The `InetAddress` class which represents both IPv4 and IPv6, doesn't have public constructors so, we create an instance "`address`" using one of its methods:

- `getLocalHost()`: returns the address of the local host on the device.
- `getHostAddress()`: returns the IP address in text.

#### DatagramSocket

To provide a UDP connection between two computers in a network we used `DatagramSocket` the constructor is used to create a server that binds to the specific port number, so the clients know how to connect to using.

- `DatagramSocket socket = new DatagramSocket(3333)` -
- `send(DatagramPacket p)`: sends a datagram packet.
- `receive(DatagramPacket p)`: receives a datagram packet.

`setSoTimeout(int timeout)`: sets timeout in milliseconds, limiting the waiting time when receiving data. If the timeout expires, a `SocketTimeoutException` is raised. These methods can throw `IOException`, `SocketTimeoutException` so, it must be catch or rethrow.

#### DatagramPacket

`DatagramPacket(byte[] buf, int length)`: the data must be in the form of an array of bytes. The first constructor is used to create a `DatagramPacket` to be received.

`DatagramPacket(byte[] buf, int length, InetAddress address, int port)`: constructor creates a **`DatagramPacket`** to be sent, so you need to specify the address and port number of the destination host.

The parameter length specifies the amount of data in the byte array to be used, usually is the length of the array (`buf.length`).

## Packet loss

The loss occurs randomly using the code below:

```
// random loss of packets
if (random.nextFloat() <= loss_rate) {
    lossCounter++;
    System.out.println("packet loss");
    continue;}
}
```

## RTT calculate

We will start a timer before we send the packet and stop after receiving it, finally get the difference:

```
Date now = new Date();
long msSend = now.getTime();
```

## Server

Basically, the server has the same structure as the client with:

- [InetAddress](#) class which used to specify the IP address using: [getLocalHost\(\)](#)
  - create a server that binds to the specific port number (9999) using: [DatagramSocket](#)
  - create packet to be receive using: [DatagramPacket\(byte\[\] buf, int length\)](#) - The packet received using: [receive\(DatagramPacket p\)](#)

Also, these methods:

- [getAddress\(\)](#); returns the IP address of the client.
- [getPort\(\)](#): return port number of the client.
- Create packet to be send using: [DatagramPacket\(byte\[\] buf, int length, InetAddress address, int port\)](#)
- After that we send the packet [send\(DatagramPacket p\)](#)

As we mentioned in the client these methods can throw Exceptions.

## 3. Project code

### 3.1 Client

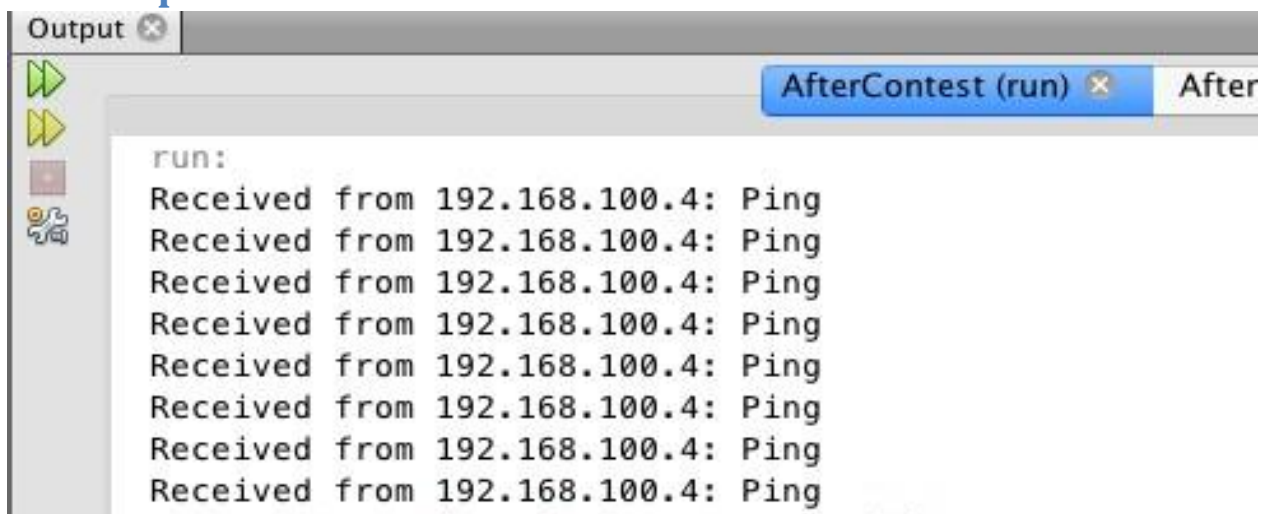
```
1 import java.io.*;
2 import java.net.*;
3 import java.util.*;
4
5 /**
6  * implement a UDP client program.
7  */
8
9 public class Client {
10     private static final int MAX_TIMEOUT = 1000;    // milliseconds
11     public static void main(String[] args) throws Exception {
12
13         Random random = new Random();
14         double loss_rate = 0.455535;
15         int lossCounter = 0;
16         int ServerPort = 9999;
17         int count = 0;
18         try {
19             InetAddress address = InetAddress.getLocalHost();
20             DatagramSocket socket = new DatagramSocket(3333);
21             byte[] message = ("ping").getBytes();
22             while (count < 10) {
23                 Date now = new Date();
24                 long msSend = now.getTime();
25                 DatagramPacket request = new DatagramPacket(message, message.length, address, ServerPort);
26                 count++;
27
28                 // random loss of packets
29                 if (random.nextFloat() <= loss_rate) {
30                     lossCounter++;
31                     System.out.println("packet loss");
32                     continue;
33                 }
34
35                 socket.send(request);
36                 try {
37                     // Set up the timeout 1000 ms = 1 sec
38                     socket.setSoTimeout(MAX_TIMEOUT);
39                     // Set up an UDP packet for receiving
40                     DatagramPacket response = new DatagramPacket(new byte[1024], 1024);
41                     // Try to receive the response from the ping
42                     socket.receive(response);
43                     // If the response is received, the code will continue here, otherwise it will continue in the catch
44                     // timestamp for when we received the packet
45                     now = new Date();
46                     long msReceived = now.getTime();
47                     // Print the packet and the RTT
48                     printData(response, msReceived - msSend);
49                 } catch (IOException e) {
50                     // Print which packet has timed out
51                     System.out.println("Request Timeout for packet no. " + count);
52                 }
53                 System.out.println("");
54                 System.out.println("Ping statistics for: " + address.getHostAddress());
55                 System.out.println("    packets: sent = "+10+"    Received = "+(10-lossCounter)+
56                    "    Lost = "+lossCounter+" ("+(lossCounter*100)/10+"% loss");
57
58                 socket.close();
59
60             } catch (SocketTimeoutException ex) {
61                 System.out.println("Timeout error: " + ex.getMessage());
62                 ex.printStackTrace();
63             } catch (IOException ex) {
64                 System.out.println("Client error: " + ex.getMessage());
65                 ex.printStackTrace();
66             }
67         }
```



## 3.2 Server

```
1 import java.io.*;
2 import java.net.*;
3
4
5 /**
6  * implement a UDP server program.
7  *
8  */
9 public class Server {
10     public static void main(String[] args) throws Exception {
11         int count = 5;
12         try{
13             int port = 9999;
14             DatagramSocket socket = new DatagramSocket(port);
15             System.out.println("server waiting...");
16
17             while (count<10) {
18                 // Create a datagram packet to hold incoming UDP packet.
19                 DatagramPacket request = new DatagramPacket(new byte[1024], 1024);
20
21                 // Block until the host receives a UDP packet.
22                 socket.receive(request);
23
24                 // Print the recieved data.
25                 printData(request);
26
27                 // Send reply.
28                 InetAddress clientHost = request.getAddress();
29                 int clientPort = request.getPort();
30                 byte[] buf = ("pong").getBytes();
31
32                 DatagramPacket reply = new DatagramPacket(buf, buf.length, clientHost, clientPort);
33                 count++;
34                 socket.send(reply); }
35
36         } catch (SocketTimeoutException ex) {
37             System.out.println("Timeout error: " + ex.getMessage());
38             ex.printStackTrace();
39         } catch (IOException ex) {
40             System.out.println("Client error: " + ex.getMessage());
41             ex.printStackTrace();}
42     }
43 }
44
```

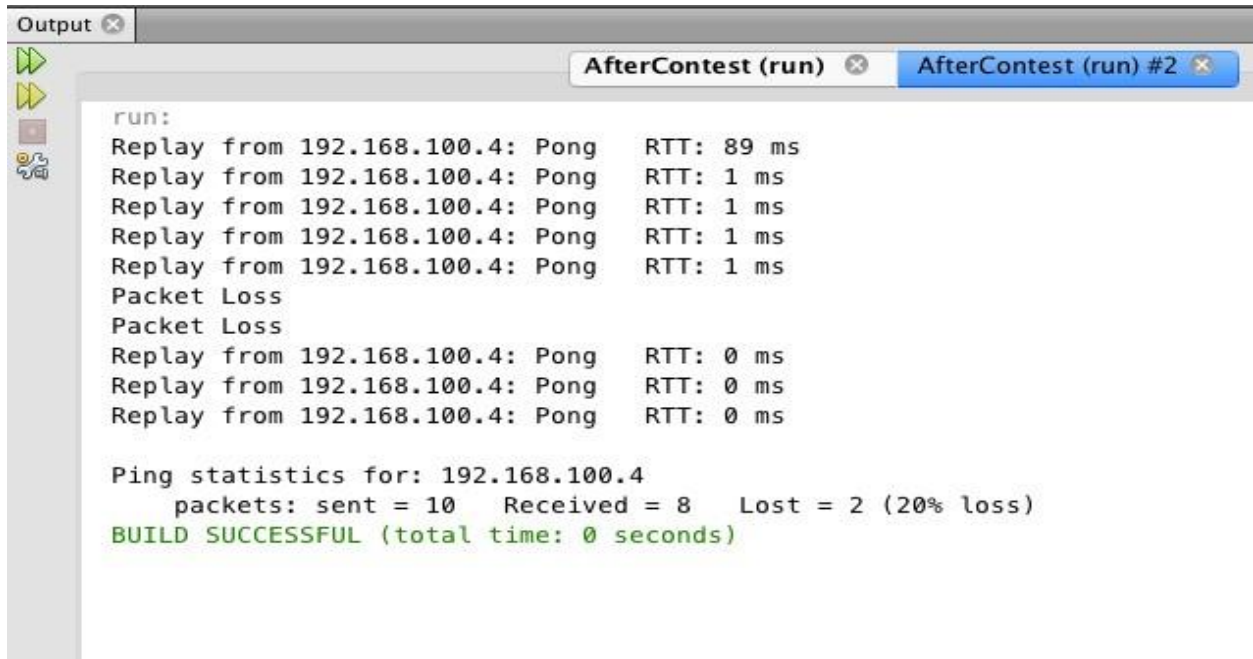
## 3.3 Output for Server



```
run:
Received from 192.168.100.4: Ping
Received from 192.168.100.4: Ping
Received from 192.168.100.4: Ping
Received from 192.168.100.4: Ping
Received from 192.168.100.4: Ping
Received from 192.168.100.4: Ping
Received from 192.168.100.4: Ping
Received from 192.168.100.4: Ping
```



## 3.4 Output for Client



```
run:
Replay from 192.168.100.4: Pong    RTT: 89 ms
Replay from 192.168.100.4: Pong    RTT: 1 ms
Replay from 192.168.100.4: Pong    RTT: 1 ms
Replay from 192.168.100.4: Pong    RTT: 1 ms
Replay from 192.168.100.4: Pong    RTT: 1 ms
Packet Loss
Packet Loss
Replay from 192.168.100.4: Pong    RTT: 0 ms
Replay from 192.168.100.4: Pong    RTT: 0 ms
Replay from 192.168.100.4: Pong    RTT: 0 ms

Ping statistics for: 192.168.100.4
    packets: sent = 10    Received = 8    Lost = 2 (20% loss)
BUILD SUCCESSFUL (total time: 0 seconds)
```

## 4. References

- 1- <https://www.geeksforgeeks.org/pinging-ip-address-java/>
- 2- <https://www.codejava.net/java-se/networking/java-udp-client-server-program-example>
- 3- [https://github.com/sariebeary/UDP\\_Protocol\\_Networks](https://github.com/sariebeary/UDP_Protocol_Networks)
- 4- <https://www.youtube.com/watch?v=SFrWdodD3hs>