Hebron George
Andrew Knutson
Computer Networks I
Dr. Miguel Labrador

TCP Echo and UDP Pinger

Version 1.00

October 10, 2012

This project is about TCP and UDP echoing and pinging, respectively. A client sends an echo and a ping to a TCP and UDP server, respectively.

History of document

- **Version 1.00** (October 10, 2012) – TCP Echo and UDP Pinger

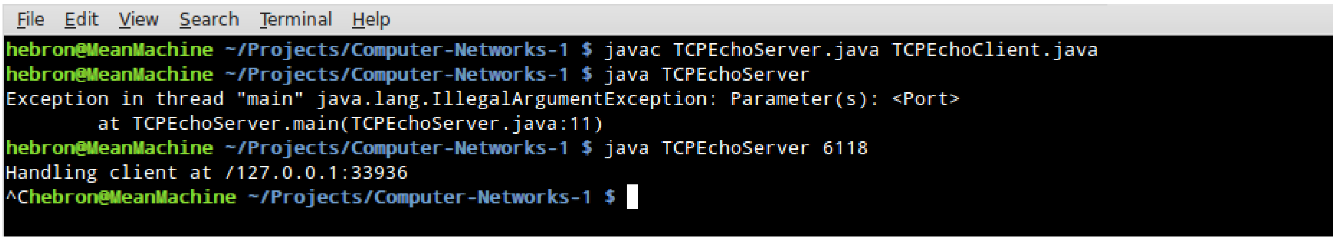
Table of Contents

Title	Page #
1. Introduction	1
2. History of Document	2
3. Table of Contents	3
4. Table of Figures	4
5. TCP Echo	5
6. UDP Pinger	6 -7

Table of Figures

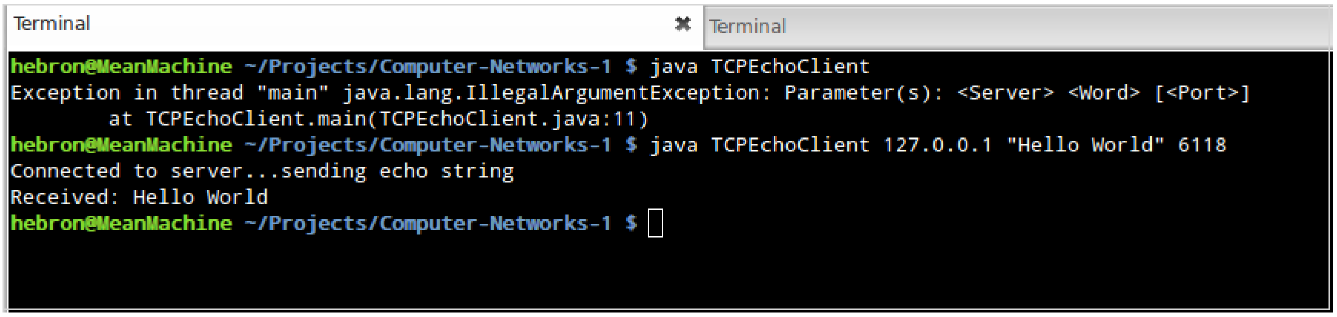
Title	Page #
Figure 5.1 – TCPEchoServer	5
Figure 5.2 – TCPEchoClient	5
Figure 6.1 – PingServer	6
Figure 7.1 – PingClient	7

TCP Echo



```
hebron@MeanMachine ~/Projects/Computer-Networks-1 $ javac TCPEchoServer.java TCPEchoClient.java
hebron@MeanMachine ~/Projects/Computer-Networks-1 $ java TCPEchoServer
Exception in thread "main" java.lang.IllegalArgumentException: Parameter(s): <Port>
        at TCPEchoServer.main(TCPEchoServer.java:11)
hebron@MeanMachine ~/Projects/Computer-Networks-1 $ java TCPEchoServer 6118
Handling client at /127.0.0.1:33936
^Chebron@MeanMachine ~/Projects/Computer-Networks-1 $
```

Figure 5.1 – TCPEchoServer

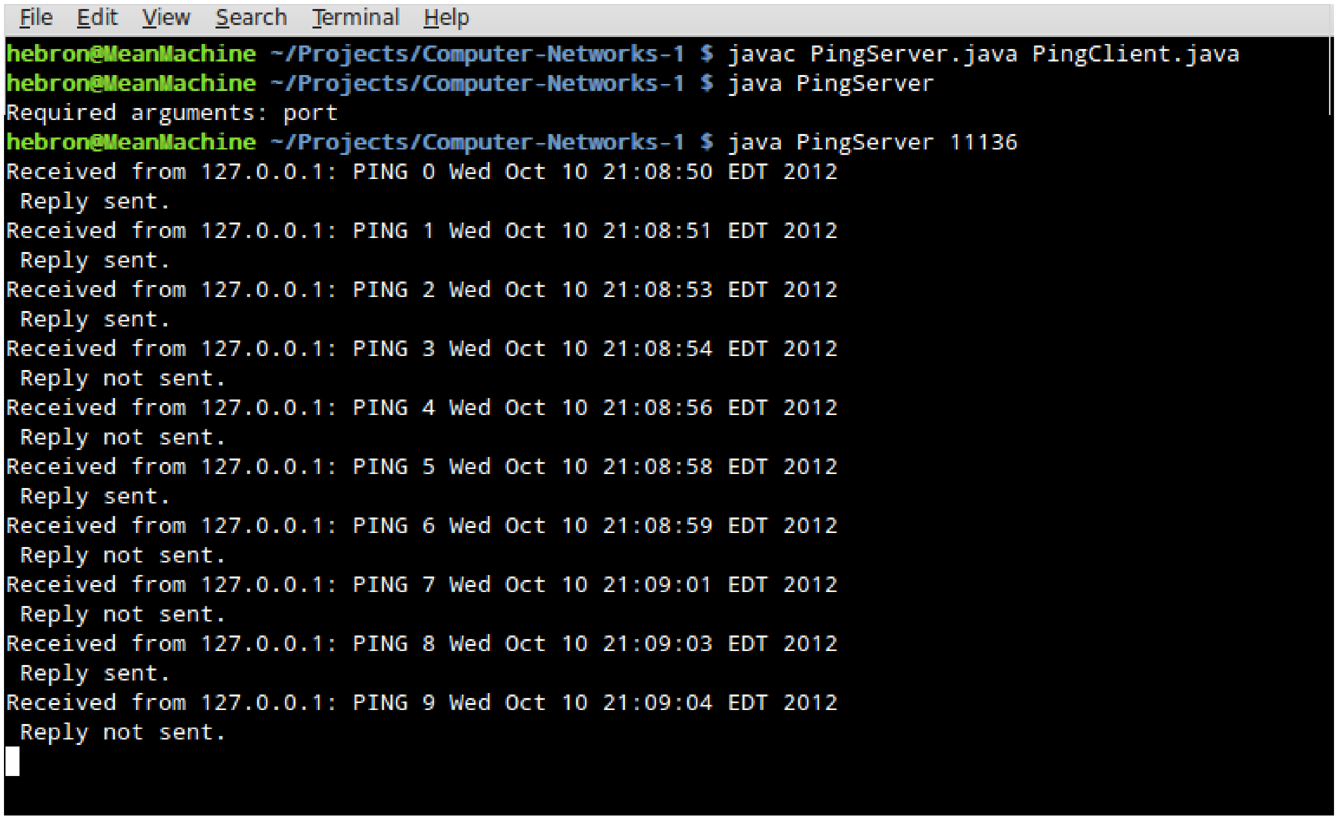


```
hebron@MeanMachine ~/Projects/Computer-Networks-1 $ java TCPEchoClient
Exception in thread "main" java.lang.IllegalArgumentException: Parameter(s): <Server> <Word> [<Port>]
        at TCPEchoClient.main(TCPEchoClient.java:11)
hebron@MeanMachine ~/Projects/Computer-Networks-1 $ java TCPEchoClient 127.0.0.1 "Hello World" 6118
Connected to server...sending echo string
Received: Hello World
hebron@MeanMachine ~/Projects/Computer-Networks-1 $
```

Figure 5.2 – TCPEchoClient

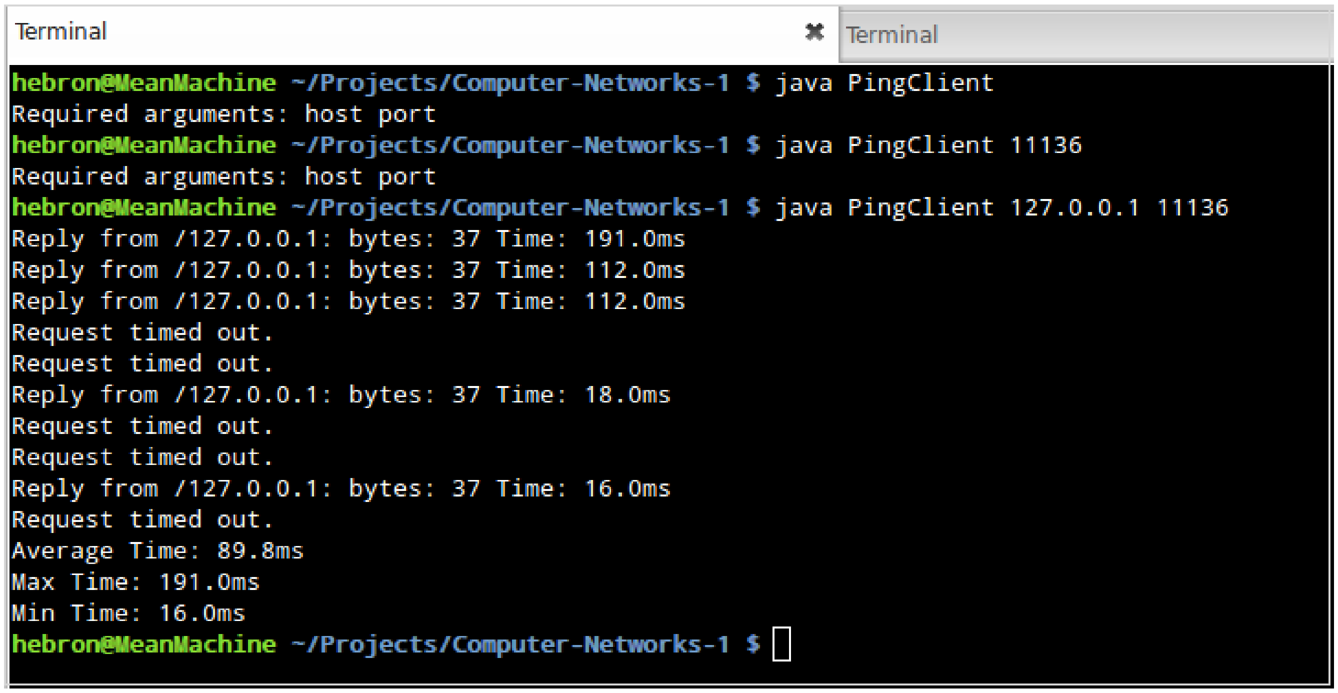
Figure 5.1 shows how to compile the TCPEchoServer.java and TCPEchoClient.java files and how to start the server. Figure 5.2 shows how to start the client. This implementation uses a 'reliable' connection, so that the connection between the server and client sends messages over and over until the counterpart confirms that it received the message.

UDP Pinger



```
hebron@MeanMachine ~/Projects/Computer-Networks-1 $ javac PingServer.java PingClient.java
hebron@MeanMachine ~/Projects/Computer-Networks-1 $ java PingServer
Required arguments: port
hebron@MeanMachine ~/Projects/Computer-Networks-1 $ java PingServer 11136
Received from 127.0.0.1: PING 0 Wed Oct 10 21:08:50 EDT 2012
 Reply sent.
Received from 127.0.0.1: PING 1 Wed Oct 10 21:08:51 EDT 2012
 Reply sent.
Received from 127.0.0.1: PING 2 Wed Oct 10 21:08:53 EDT 2012
 Reply sent.
Received from 127.0.0.1: PING 3 Wed Oct 10 21:08:54 EDT 2012
 Reply not sent.
Received from 127.0.0.1: PING 4 Wed Oct 10 21:08:56 EDT 2012
 Reply not sent.
Received from 127.0.0.1: PING 5 Wed Oct 10 21:08:58 EDT 2012
 Reply sent.
Received from 127.0.0.1: PING 6 Wed Oct 10 21:08:59 EDT 2012
 Reply not sent.
Received from 127.0.0.1: PING 7 Wed Oct 10 21:09:01 EDT 2012
 Reply not sent.
Received from 127.0.0.1: PING 8 Wed Oct 10 21:09:03 EDT 2012
 Reply sent.
Received from 127.0.0.1: PING 9 Wed Oct 10 21:09:04 EDT 2012
 Reply not sent.
```

Figure 6.1 – PingServer



```
hebron@MeanMachine ~/Projects/Computer-Networks-1 $ java PingClient
Required arguments: host port
hebron@MeanMachine ~/Projects/Computer-Networks-1 $ java PingClient 11136
Required arguments: host port
hebron@MeanMachine ~/Projects/Computer-Networks-1 $ java PingClient 127.0.0.1 11136
Reply from /127.0.0.1: bytes: 37 Time: 191.0ms
Reply from /127.0.0.1: bytes: 37 Time: 112.0ms
Reply from /127.0.0.1: bytes: 37 Time: 112.0ms
Request timed out.
Request timed out.
Reply from /127.0.0.1: bytes: 37 Time: 18.0ms
Request timed out.
Request timed out.
Reply from /127.0.0.1: bytes: 37 Time: 16.0ms
Request timed out.
Average Time: 89.8ms
Max Time: 191.0ms
Min Time: 16.0ms
hebron@MeanMachine ~/Projects/Computer-Networks-1 $
```

Figure 7.1 - PingClient

Figure 6.1 shows how to compile the PingServer.java and PingClient.java files. These two programs implement an 'unreliable' connection. When the client or server sends a message it only waits for a certain amount of time for a reply acknowledgement and then it times out and doesn't 'care' if their counterpart got the packet. It's up to the Application layer programmer to implement any error checking at that point.