Assignment 1

Computer Networks - an introduction

Author: John Herrlin
Email: jh222jx@student.lnu.se
Semester: VT2016
Area: Computer Science
Coursecode: 1DV701

Contents

1 Introduction

All virtual machines are running in Docker containers, Dockers internal network is 172.17.0.0. Base host is addressed with 172.17.0.1. The container used is the official Java, and the specific version is OpenJDK / OpenJRE 8.

2 Assignment 1.1

This assignment is based of setting up a virtual environment and send icmp (ping) packages.

Two Docker containers is set up, 172.17.0.2 and 172.17.0.3. 172.17.0.2 sends 5 icmp requests to 172.17.0.3 and gets 5 icmp replays back from target.

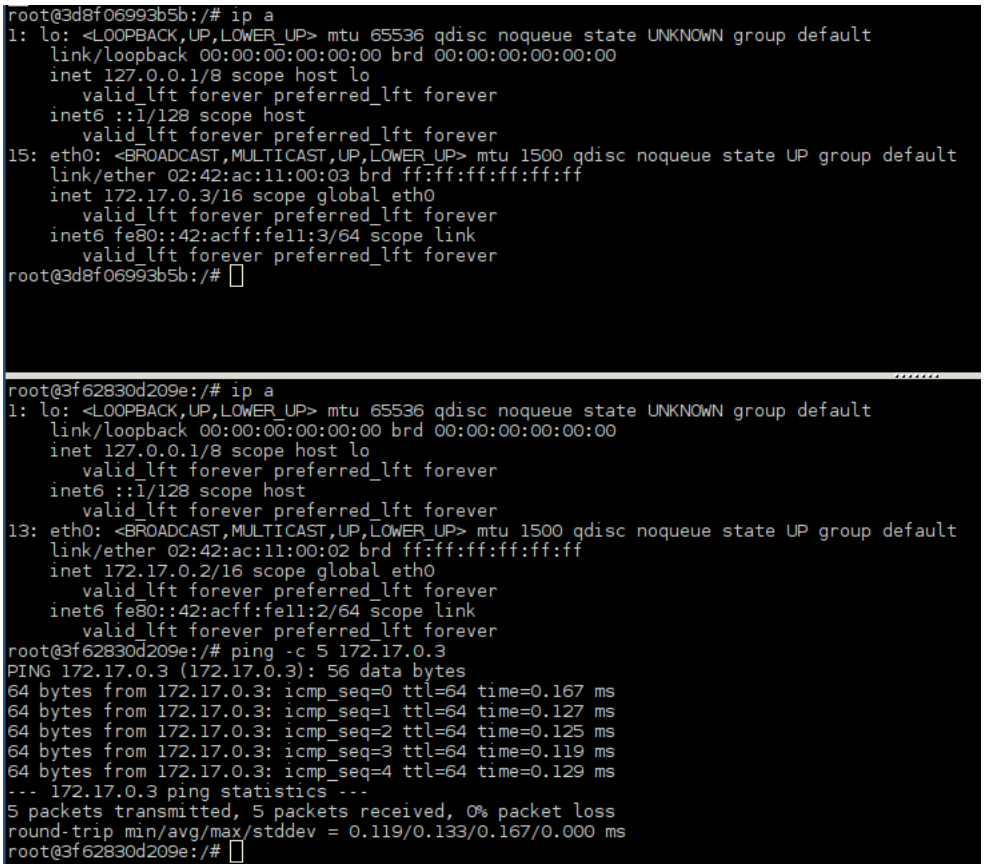

Figure 1: ping -c 5

3 Assignment 1.2

To handle commandline arguments i used Apache Commons CLI lib. This provides some good features for handeling cli inputs. I had to write some exception handling when types didnt match. All of this is done in the abstract class Host and is then applied to all of the hosts by inheritence, see 3.1.2.

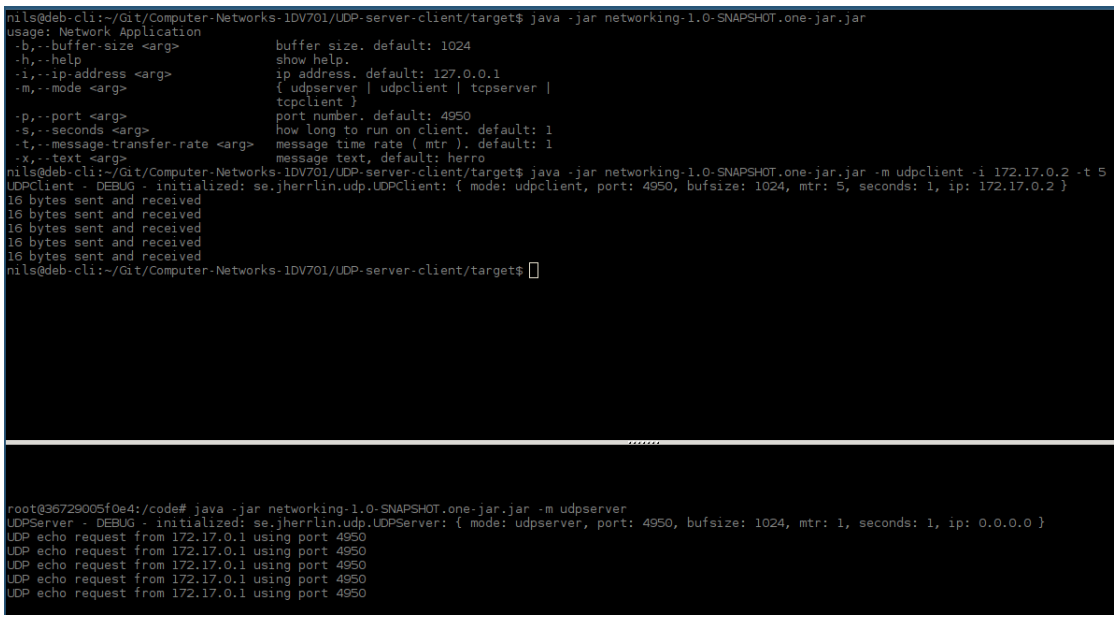

Figure 2: Send 5 UDP package in 1 sec

3.1 VG-tasks

Clairification on vg-tasks.

3.1.1 Task 1

Not implemented

3.1.2 Task 2

In the implementation there is a abstract class called Host. All of the hosts (UDPServer, UDPClient, TCPServer, TCPClient) extends Host. Host gives functionallity to validate that all input parameters are correct. It also provides a run method. All hosts override the run method to implement the specific logic for the specific host.

4 Assignment 1.3

In 4 there is a screenshot of 4 clients creating TCP connections to the server.

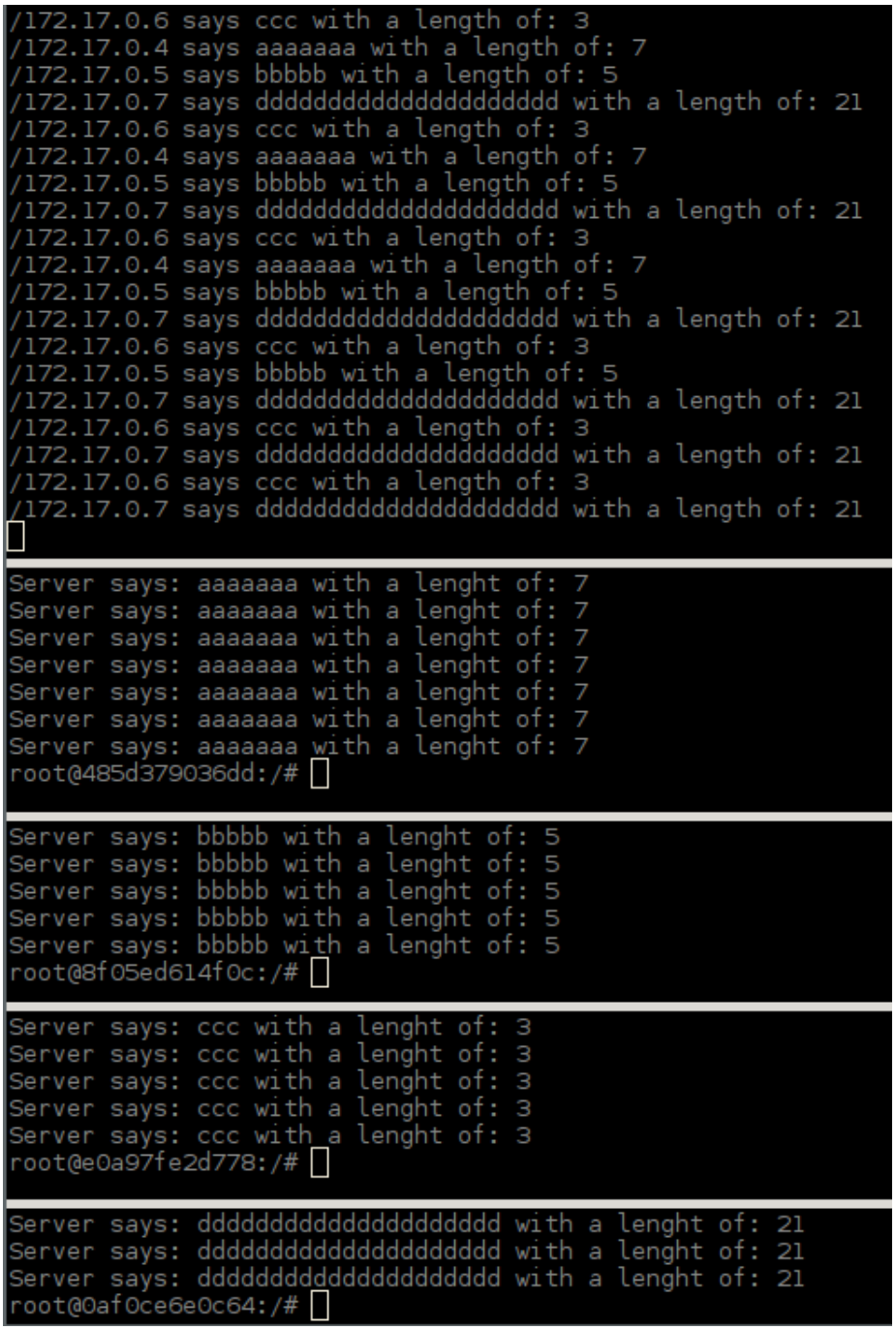

Figure 3: TCP server with threads, multi client connection

4.1 TCP server with small bufsize

In 4.1 there is a screenshot of an TCP server with a buffer size of 5. The client is sending a message with a size of 10. The server is handling this in a good way and assembles the stream. Returning a message that is the size of 10.

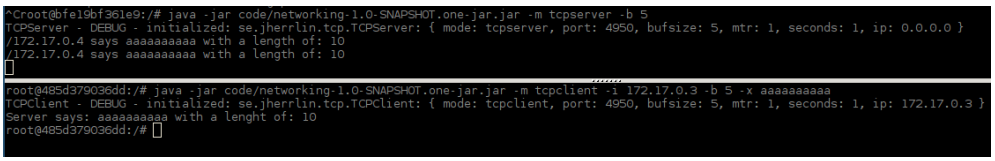

Figure 4: TCP where bufsize is smaller than message length

4.2 UDP server with small bufsize

In 4.2 there is a screenshot of an UDP server with a buffer size of 5. The client is sendning a message with a size of 10. The UDP is not handeling this good, it cuts the message and sends back half the message to the client.

Figure 5: UDP where bufsize is smaller than message length

4.3 Conclusion

The reason that TCP server is handeling this in a good way it that is uses a stream. It can be refered to open a file and read or write a steam. In UDP we just get a package and nothing more, if we dont take care of the package UDP wont help us doing it.

5 Assignment 1.4, Wireshark

5.1 UDP

UDP client / server in Wireshark

5.1.1 To small buffersize

In 5.1.1 we see that the length is bigger on the way to the server than on the way back to the client

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.17.0.4	172.17.0.1	UDP	58	Source port: 4950 Destination port: 4950
2	0.000196000	172.17.0.1	172.17.0.4	UDP	47	Source port: 4950 Destination port: 4950

Figure 6: UDP server with small buffer size

5.1.2 Enought buffersize

In 5.1.2 we see that the length is the same on both ways

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.17.0.4	172.17.0.1	UDP	58	Source port: 4950 Destination port: 4950
2	0.000086000	172.17.0.1	172.17.0.4	UDP	58	Source port: 4950 Destination port: 4950

Figure 7: UDP server with enought buffer size

5.2 TCP

TCP client / server in Wireshark

5.2.1 To small buffersize

In 5.2.1 we see that there is alot of packages going back forth. TCP have the Three way handshake or Syn-Ack when esatblishing and finishing a connection stream. For every package that is send, the reciever returns an Ack to the sender that tells the sender that the reciever have got the package and the data inside is correct. If we compair the number of packages in 5.2.1 and 5.2.2 we see that 5.2.2 have less packages. The reason for this is that the buffer size is big enought to handle the data in fewer packets.

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.17.0.4	172.17.0.1	TCP	74	38336→4950 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=13933987 TSecr=0 WS
2	0.000047000	172.17.0.1	172.17.0.4	TCP	74	4950→38336 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=13933987
3	0.000064000	172.17.0.4	172.17.0.1	TCP	66	38336→4950 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=13933987 TSecr=13933987
4	0.001060000	172.17.0.4	172.17.0.1	TCP	67	38336→4950 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=1 TSval=13933988 TSecr=13933987
5	0.001090000	172.17.0.1	172.17.0.4	TCP	66	4950→38336 [ACK] Seq=1 Ack=2 Win=29056 Len=0 TSval=13933988 TSecr=13933988
6	0.001136000	172.17.0.4	172.17.0.1	TCP	67	38336→4950 [PSH, ACK] Seq=2 Ack=1 Win=29312 Len=1 TSval=13933988 TSecr=13933988
7	0.001144000	172.17.0.1	172.17.0.4	TCP	66	4950→38336 [ACK] Seq=1 Ack=3 Win=29056 Len=0 TSval=13933988 TSecr=13933988
8	0.001164000	172.17.0.4	172.17.0.1	TCP	67	38336→4950 [PSH, ACK] Seq=3 Ack=1 Win=29312 Len=1 TSval=13933988 TSecr=13933988
9	0.001170000	172.17.0.1	172.17.0.4	TCP	66	4950→38336 [ACK] Seq=1 Ack=4 Win=29056 Len=0 TSval=13933988 TSecr=13933988
10	0.001178000	172.17.0.4	172.17.0.1	TCP	67	38336→4950 [PSH, ACK] Seq=4 Ack=1 Win=29312 Len=1 TSval=13933988 TSecr=13933988
11	0.001182000	172.17.0.1	172.17.0.4	TCP	66	4950→38336 [ACK] Seq=1 Ack=5 Win=29056 Len=0 TSval=13933988 TSecr=13933988
12	0.001190000	172.17.0.4	172.17.0.1	TCP	67	38336→4950 [PSH, ACK] Seq=5 Ack=1 Win=29312 Len=1 TSval=13933988 TSecr=13933988
13	0.001194000	172.17.0.1	172.17.0.4	TCP	66	4950→38336 [ACK] Seq=1 Ack=6 Win=29056 Len=0 TSval=13933988 TSecr=13933988
14	0.001201000	172.17.0.4	172.17.0.1	TCP	67	38336→4950 [PSH, ACK] Seq=6 Ack=1 Win=29312 Len=1 TSval=13933988 TSecr=13933988
15	0.001205000	172.17.0.1	172.17.0.4	TCP	66	4950→38336 [ACK] Seq=1 Ack=7 Win=29056 Len=0 TSval=13933988 TSecr=13933988
16	0.001213000	172.17.0.4	172.17.0.1	TCP	67	38336→4950 [PSH, ACK] Seq=7 Ack=1 Win=29312 Len=1 TSval=13933988 TSecr=13933988
17	0.001217000	172.17.0.1	172.17.0.4	TCP	66	4950→38336 [ACK] Seq=1 Ack=8 Win=29056 Len=0 TSval=13933988 TSecr=13933988
18	0.001225000	172.17.0.4	172.17.0.1	TCP	67	38336→4950 [PSH, ACK] Seq=8 Ack=1 Win=29312 Len=1 TSval=13933988 TSecr=13933988
19	0.001229000	172.17.0.1	172.17.0.4	TCP	66	4950→38336 [ACK] Seq=1 Ack=9 Win=29056 Len=0 TSval=13933988 TSecr=13933988
20	0.001236000	172.17.0.4	172.17.0.1	TCP	67	38336→4950 [PSH, ACK] Seq=9 Ack=1 Win=29312 Len=1 TSval=13933988 TSecr=13933988
21	0.001240000	172.17.0.1	172.17.0.4	TCP	66	4950→38336 [ACK] Seq=1 Ack=10 Win=29056 Len=0 TSval=13933988 TSecr=13933988
22	0.001248000	172.17.0.4	172.17.0.1	TCP	67	38336→4950 [PSH, ACK] Seq=10 Ack=1 Win=29312 Len=1 TSval=13933988 TSecr=13933988
23	0.001252000	172.17.0.1	172.17.0.4	TCP	66	4950→38336 [ACK] Seq=1 Ack=11 Win=29056 Len=0 TSval=13933988 TSecr=13933988
24	0.400960000	172.17.0.1	172.17.0.4	TCP	76	4950→38336 [PSH, ACK] Seq=1 Ack=11 Win=29056 Len=10 TSval=13934087 TSecr=13933988
25	0.401071000	172.17.0.4	172.17.0.1	TCP	66	38336→4950 [ACK] Seq=11 Ack=11 Win=29312 Len=0 TSval=13934088 TSecr=13934087
26	0.401520000	172.17.0.4	172.17.0.1	TCP	66	38336→4950 [FIN, ACK] Seq=11 Ack=11 Win=29312 Len=0 TSval=13934088 TSecr=13934087
27	0.441057000	172.17.0.1	172.17.0.4	TCP	66	4950→38336 [ACK] Seq=11 Ack=12 Win=29056 Len=0 TSval=13934098 TSecr=13934088

Figure 8: TCP server with small buffer size

5.2.2 Enought buffersize

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	172.17.0.4	172.17.0.1	TCP	74	38339→4950 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=13948717 TSecr=0 WS
2	0.000040000	172.17.0.1	172.17.0.4	TCP	74	4950→38339 [SYN, ACK] Seq=0 Ack=1 Win=28960 Len=0 MSS=1460 SACK_PERM=1 TSval=13948717
3	0.000057000	172.17.0.4	172.17.0.1	TCP	66	38339→4950 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=13948717 TSecr=13948717
4	0.001081000	172.17.0.4	172.17.0.1	TCP	67	38339→4950 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=1 TSval=13948718 TSecr=13948717
5	0.001099000	172.17.0.1	172.17.0.4	TCP	66	4950→38339 [ACK] Seq=1 Ack=2 Win=29056 Len=0 TSval=13948718 TSecr=13948718
6	0.001129000	172.17.0.4	172.17.0.1	TCP	67	38339→4950 [PSH, ACK] Seq=2 Ack=1 Win=29312 Len=1 TSval=13948718 TSecr=13948718
7	0.001146000	172.17.0.1	172.17.0.4	TCP	66	4950→38339 [ACK] Seq=1 Ack=3 Win=29056 Len=0 TSval=13948718 TSecr=13948718
8	0.001154000	172.17.0.4	172.17.0.1	TCP	67	38339→4950 [PSH, ACK] Seq=3 Ack=1 Win=29312 Len=1 TSval=13948718 TSecr=13948718
9	0.001168000	172.17.0.1	172.17.0.4	TCP	66	4950→38339 [ACK] Seq=1 Ack=4 Win=29056 Len=0 TSval=13948718 TSecr=13948718
10	0.400758000	172.17.0.1	172.17.0.4	TCP	69	4950→38339 [PSH, ACK] Seq=1 Ack=4 Win=29056 Len=3 TSval=13948818 TSecr=13948718
11	0.400808000	172.17.0.4	172.17.0.1	TCP	66	38339→4950 [ACK] Seq=4 Ack=4 Win=29312 Len=0 TSval=13948818 TSecr=13948818
12	0.401276000	172.17.0.4	172.17.0.1	TCP	66	38339→4950 [FIN, ACK] Seq=4 Ack=4 Win=29312 Len=0 TSval=13948818 TSecr=13948818
13	0.440157000	172.17.0.1	172.17.0.4	TCP	66	4950→38339 [ACK] Seq=4 Ack=5 Win=29056 Len=0 TSval=13948828 TSecr=13948818

Figure 9: TCP server with enought buffer size

6 Instructions

Install, compile and run instructions.

6.1 Install dependencies

This is a Maven project. Install Maven3 and OpenJDK on a Debian based system.

```
apt-get install openjdk-7-jdk
apt-get install maven
```

6.2 Compile and run

6.2.1 Compile

Navigate to project root, where the file pom.xml is located. Package the project with:

```
mvn package -e -DskipTests
```

6.2.2 Execute

Execute the code:

```
java -jar target/networking-1.0-SNAPSHOT.one-jar.jar
```

6.2.3 Docker, Optional

The code needs to be compiled with Maven before running in Docker, see 6.2.1.

```
apt-get install docker
docker run -i -t --rm -v $PWD:/code java:8 /bin/bash
java -jar code/networking-1.0-SNAPSHOT.one-jar.jar
```