Programmieraufgabe

TX.c

- UDP Transmitprogramme on C. Sended message: 'Hello from TX.c'
Usage:
  ./TX <port> <amount of packets> <delay>
<port> - port on which packets are sent. (IP address is always '127.0.0.1')
<amount of packets> - amount of UDP packets to send.
<delay> - delay in microseconds
without arguments uses port 4711, send 100 packets with 0 delay

RX.c

- UDP Recieveprogramme(Server) on C.
Usage:
  ./RX <port> <buffer size>
<port> - port to listen.
<buffer size> - size of recieve buffer in Bytes
without arguments uses port 4711 and buffer size 1024 B

TX.java

- UDP Transmitprogramme on Java. Sended message: 'Hello frome TX.java'
Usage:
  java TX <port> <amount of packets> <delay>
<port> - port on which packets are sent. (IP address is always '127.0.0.1')
<amount of packets> - amount of UDP packets to send.
<delay> - delay in microseconds
without arguments uses port 4711, send 100 packets with 0 delay

RX.java

- UDP Recieveprogramme(Server) on Java.
Usage:
  ./RX <port> <buffer size>
<port> - port to listen
<buffer size> - size of recieve buffer in Bytes
without arguments uses port 4711 and buffer size 1024 B

Testing algorithm

For every paar of programs:
- RX.c + TX.c
- RX.c + TX.java
- RX.java + TX.java
- RX.java + TX.c

Try to send 100, 1000, 1000 UDP packets without delay between sends.
if there were packet loss, we tried to increase the delay to such a value that there would be no packet loss
For every conditions make 10 tests and measure average packet Loss (absolute and %)

TX.c -> RX.c

C to C

# test	# packets	buffer size	delay	Packet loss(avg)	Lost in %
10	100	1024	0	0	0.00
10	1000	1024	0	8.2	0.82
10	10000	1024	0	2154.3	21.54
10	100	1024	100	0	0.00
10	1000	1024	100	6.9	0.69
10	10000	1024	100	2086.6	20.87
10	100	1024	300	0	0.00
10	1000	1024	300	0	0.00
10	10000	1024	300	1426.3	14.26
10	100	1024	500	0	0.00
10	1000	1024	500	0	0.00
10	10000	1024	500	0	0.00

TX.c -> RX.java

C to Java

# test	# packets	buffer size	delay	Packet loss(avg)	Lost in %
10	100	1024	0	0	0.00
10	1000	1024	0	618.3	61.83
10	10000	1024	0	8629.2	86.29
10	100	1024	100	0	0.00
10	1000	1024	100	600	60.00
10	10000	1024	100	8455.8	84.56
10	100	1024	500	0	0.00
10	1000	1024	500	367.8	36.78
10	10000	1024	500	6193.3	61.93
10	100	1024	1000	0	0.00
10	1000	1024	1000	50.5	5.05
10	10000	1024	1000	3103.4	31.03
10	100	1024	2000	0	0.00
10	1000	1024	2000	0	0.00
10	10000	1024	2000	0	0.00

TX.java -> RX.c

Java to C

# test	# packets	buffer size	delay	Packet loss(avg)	Lost in %
10	100	1024	0	0	0.00
10	1000	1024	0	0	0.00
10	10000	1024	0	646.1	6.46
10	100	1024	100	0	0.00
10	1000	1024	100	0	0.00
10	10000	1024	100	0	0.00

TX.java -> RX.java

Java to Java

# test	# packets	buffer size	delay	Packet loss(avg)	Lost in %
10	100	1024	0	0	0.00
10	1000	1024	0	269.9	26.99
10	10000	1024	0	6906.1	69.06
10	100	1024	100	0	0.00
10	1000	1024	100	0	0.00
10	10000	1024	100	1807.1	18.07
10	100	1024	500	0	0.00
10	1000	1024	500	0	0.00
10	10000	1024	500	1777.9	17.78
10	100	1024	1000	0	0.00
10	1000	1024	1000	0	0.00
10	10000	1024	1000	1591.8	15.92
10	100	1024	2000	0	0.00
10	1000	1024	2000	0	0.00
10	10000	1024	2000	0	0.00

Folgerungen:

Über alle Test hinweg lässt sich eindeutig erkennen das der Delay zwischen den Paketen zu insgesamt zu weitaus weniger Paketverlusten führt. Mittels vergrößerung des Delays lässt sich bei allen Tests, außer C zu Java, der Paketverlust bei 100,1000 und auch 10000 Paketen auf bis zu 0% verringern.

Als schlechtester Fall sticht C->Java hervor, selbst mit einem Delay von 1000 findet trotzdem noch ein Paketverlust von 31% der 10000 Pakete, dies liegt daran das C viel schneller die Pakete produziert als Java sie konsumieren kann.Ohne Delay kommt man sogar auf einen Verlust von ganzen 86% der 10000 Pakete womit fast keine erkennbare Datei mehr ankommen kann.Bei ganz wenig Paketen kommt es noch zu keinem allzu großen Overflow bei Java weshalb keine Pakete verloren gehen.

Als bester Fall lässt sich Java->C erkennen, da bereits bei einem Delay von 100 keine Pakete mehr verloren gehen. Wie vorhin erwähnt arbeitet Java langsamer als C, daher kommt es zu wenig Paketverlusten da kein Overflow beim RX.c entsteht was zu einem geringem Paketverlust führt.