Directions

A. Due Date: 27 Nov 2024 at 11:55pm

B. The homework may be done in groups of up to two students.

C. All programs must be written in Java.

D. Answer all non-programming questions in Hebrew or English.

E. Use exact values for GB, MB, KB, Kbps, Mbps, and Gbps.

F. The GitHub repository includes several GitHub Actions scripts that will compile your code and perform test runs on it. You can see the output from the actions scripts in the Actions tab on GitHub. Those outputs will be the basis for your grade.

What to turn in

G. For the programming sections, turn in all related source code along with any headers or build files needed to compile the code in the GitHub repository.

H. For non-programming sections, turn in all answers in PCAP and TXT files as appropriate.

I. In addition, turn in a README.md file with the following:

- Names and TZ numbers of all students in the group
- Total number of hours spent on each part of the assignment
- Date of submission
- Comments or feedback on the assignment's requirements or complexity (optional)

J. Submissions missing any of the above will be penalized 5 points.

How to submit

K. Turn in your submission via the starter repositories opened for you on GitHub via GitHub Classroom.

L. All submissions **must** be made in the starter GitHub repository for the assignment that is opened via GitHub classroom. Code placed anywhere else or submitted in any other manner - including via email or in unrelated repositories - will not be graded.

M. Place the files for the analysis questions in the repository root.

N. Indicate that your work is complete by performing a commit with the comment "Submitted for grading" on each repository. The grade for the assignment will be based on the first commit with that comment, so create a submission with that comment only when you are finished.

O. All group members must perform least one substantive commit to the repository to receive a grade.

Network Sending

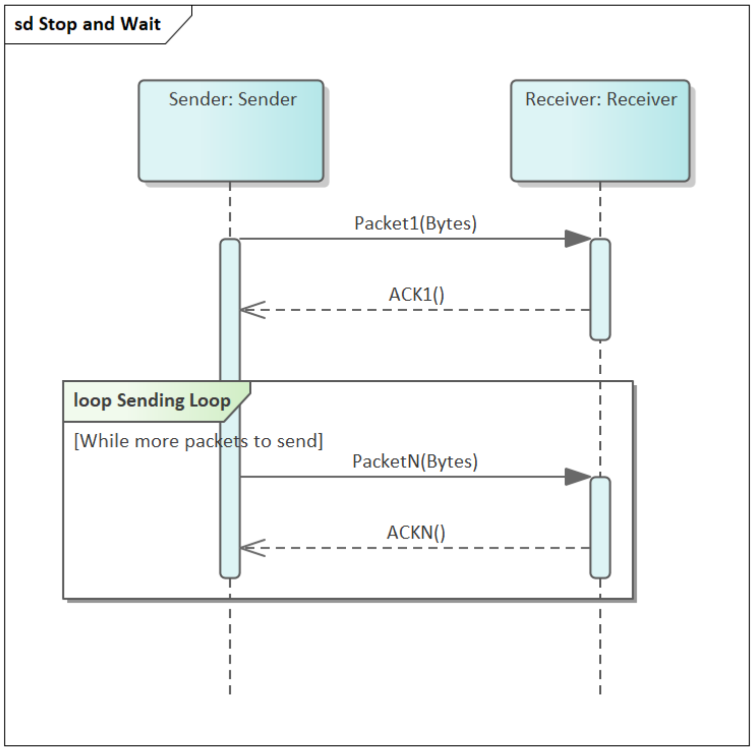
In this assignment you will create tools that test the behavior of networks in sending packets. You first will create a calculator that calculates the theoretical time it should take to send a file using a simple sending method. You will then create tools that truly send files across a network. You will thereby be able to compare the theoretical behavior with actual network behavior.

1 Sending with ACKs Calculator (30 points)

First you will build a command line calculator tool to compute the theoretical time to send a file using the stop and wait protocol with acknowledgements. The calculator must accept the following parameters:

- The size of the file in bytes, Kilobytes (2^{10}B), Megabytes (2^{20}B), Gigabytes (2^{30}B)
- The size of the data packets for the file in bytes or Kilobytes (2^{10}B)
- The size of the ACK packets in bytes
- The bandwidth of the link in Mbps (megabits per second) or Gbps (gigabits per second)
- The round trip time in milliseconds

The tool will calculate how long it should take to send the file as a series of packets using the "stop and wait" protocol under ideal conditions. This tool does not actually send the file anywhere. The stop and wait protocol with acks behaves as shown below:



The output of the tool must be the total time that it should **theoretically** take to send the file, from the time the first packet leaves until the last ACK finishes arriving at the sender.

1.1 Requirements

The tool must meet the following requirements:

1. It must accept its inputs as command line arguments
2. It must output the final result (seconds) to the command line
3. If an input value is missing, badly formed, or illegal the tool must print an error message and quit
4. The file size, data packet size, ACK packet size, bandwidth, and RTT may be provided as non-negative real numbers (*e.g.* 1.5Mbps for bandwidth, 0.75ms for RTT).
5. For the file size, data packet size, ACK packet size, bandwidth, and RTT numbers, there **will not be** a space between the number of units and the unit label. For instance:
   - OK: 5Mbps, 0.1ms, 1400B, 10.5MB
   - NO: 5 Mbps, 0.1 ms, 1400 B, 10.5 MB
6. The tool must use the following codes for units:

   File Size B (bytes), KB (kilobytes), MB (megabytes), GB (gigabytes)

   Data Packet Size B (bytes), KB (kilobytes)

   ACK Packet Size B (bytes)

   Bandwidth mbps (megabits per second), gbps (gigabits per second)

   RTT ms (milliseconds)
7. The inputs will be provided are:

   filesize Total size of the file to send

   packetsize Size of the data packet

   acksize ACK packet size

   bandwidth Bandwidth of the connection

   rtt Round trip time between the sides
8. The tool must be written in Java

1.2 Test Cases (25 points total)

The following are sample inputs and outputs assuming the tool is called `SendingWithAcksJava-5784.jar`. All tests must be preceded by `java -jar`, but they are elided for brevity for all but the first test case.

1.
   ```
   java -jar SendingWithAcksJava-5784.jar -filesize=40B -packetsize=10B -acksize=10B
   -bandwidth=100mbps -rtt=52ms
   ```
   - Output: 0.208006 s

2.
   ```
   SendingWithAcksJava-5784.jar -filesize=10KB -packetsize=10B -acksize=10B -bandwidth=100mbps
   -rtt=51ms
   ```
   - Output: 52.225638 s

3.
   ```
   SendingWithAcksJava-5784.jar -filesize=23MB -packetsize=10B -acksize=10B -bandwidth=100mbps
   -rtt=53ms
   ```
   - Output: 127825.283760 s

4. `SendingWithAcksJava-5784.jar -filesize=45.5GB -packetsize=10B -acksize=10B -bandwidth=100mbps -rtt=154ms`
   - Output: 752378713.040479 s
5. `SendingWithAcksJava-5784.jar -filesize=40B -packetsize=10B -acksize=10B -bandwidth=100mbps -rtt=45ms`
   - Output: 0.180006 s
6. `SendingWithAcksJava-5784.jar -filesize=10KB -packetsize=1KB -acksize=10B -bandwidth=100mbps -rtt=32ms`
   - Output: 0.320827 s
7. `SendingWithAcksJava-5784.jar -filesize=23MB -packetsize=52B -acksize=10B -bandwidth=100mbps -rtt=200ms`
   - Output: 92761.100415 s
8. `SendingWithAcksJava-5784.jar -filesize=45.5GB -packetsize=0.1KB -acksize=10B -bandwidth=100mbps -rtt=16ms`
   - Output: 7639787.619111 s
9. `SendingWithAcksJava-5784.jar -filesize=40B -packetsize=10B -acksize=10B -bandwidth=100mbps -rtt=11ms`
   - Output: 0.044006 s
10. `SendingWithAcksJava-5784.jar -filesize=10KB -packetsize=1KB -acksize=10B -bandwidth=100Gbps -rtt=89ms`
    - Output: 0.890001 s
11. `SendingWithAcksJava-5784.jar -filesize=23MB -packetsize=150B -acksize=10B -bandwidth=10.5Gbps -rtt=189ms`
    - Output: 30387.817600 s
12. `SendingWithAcksJava-5784.jar -filesize=45.5GB -packetsize=0.1KB -acksize=10B -bandwidth=0.5Gbps -rtt=101ms`
    - Output: 48199935.528022 s
13. `SendingWithAcksJava-5784.jar -filesize=40KB -packetsize=1KB -acksize=50B -bandwidth=10mbps -rtt=7ms`
    - Output: 0.314368 s
14. `SendingWithAcksJava-5784.jar -filesize=10MB -packetsize=1000B -acksize=100B -bandwidth=100gbps -rtt=150ms`
    - Output: 1572.900923 s
15. `SendingWithAcksJava-5784.jar -filesize=10MB -bandwidth=100gbps -rtt=50ms`

    Missing parameter. Expected output:

    ```
    Usage: -filesize=f -packetsize=p -acksize=a -bandwidth=b -rtt=r
    f can be of units B, KB, MB, GB
    p can be of units B, KB
    a can be of unit B
    ```

```
b can be of units mbps, gbps
r can be of unit ms
```

16. `SendingWithAcksJava-5784.jar -filesize=40B -packetsize=40B -rtt=40ms`

    Missing parameter. Expected output:

    ```
    Usage: -filesize=f -packetsize=p -acksize=a -bandwidth=b -rtt=r
    f can be of units B, KB, MB, GB
    p can be of units B, KB
    a can be of unit B
    b can be of units mbps, gbps
    r can be of unit ms
    ```

1.3 Documentation (5 points)

Add Javadoc documentation to every method and class. The documentation for methods must include:

- A 1-2 sentence summary of the method's purpose
- @param entries for all parameters, including what they are used for
- @return entry with a 1-2 sentence description of the return value
- @throws entries for any exceptions thrown.

The documentation for classes must includes:

- A 2-3 sentence description of the class' purpose
- @author the code's author
- @version a version for the class. Update on every commit to the repository.

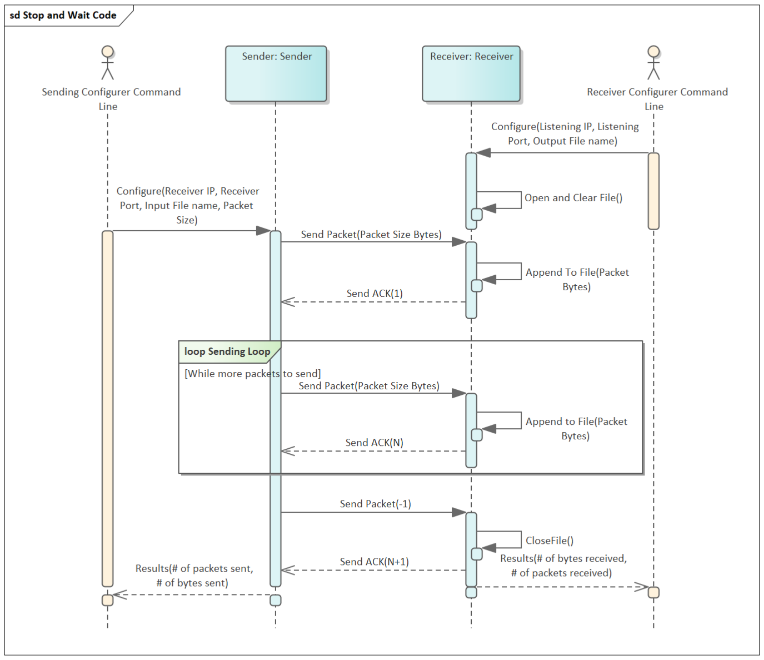
2 Actual Network Sending with UDP (50 points)

Having prepared tools that can do theoretical network timing calculations, we will next build a tool that actually does network sending over a LAN and compare it performance to the theoretical times that our calculator came up with.

The networking tool will consist of two programs:

1. A sending program that sends a file to the receiving program using UDP
2. A receiving program the receives a file from the sender using UDP

The tool will implement the "stop and wait" protocol as shown in the following diagram. Assume the file needs N packets to be sent:


sd Stop and Wait Code
Sending Configurer Command Line
Sender: Sender
Receiver: Receiver
Receiver Configurer Command Line
Configure(Listening IP, Listening Port, Output File name)
Open and Clear File()
Configure(Receiver IP, Receiver Port, Input File name, Packet Size)
Send Packet(Packet Size Bytes)
Append To File(Packet Bytes)
Send ACK(1)
loop Sending Loop
[While more packets to send]
Send Packet(Packet Size Bytes)
Append to File(Packet Bytes)
Send ACK(N)
Send Packet(-1)
CloseFile()
Results(# of bytes received, # of packets received)
Send ACK(N+1)
Results(# of packets sent, # of bytes sent)

Note that the sender tool indicates that the file is done by sending a packet with a single byte -1.

We next specify the behavior of the sending and receiving tools.

2.1 Sending Tool

The sending tool shall be a command line tool that accepts the following command line parameters:

dest The IP address of the recipient tool

port The listening port of the recipient uses

f The name of the file to send

packetsize The data packet size to use in bytes

2.1.1 Requirements

The parameters must meet the following requirements:

1. The IP address must be in the standard IPv4 four byte dot separated format (*e.g.* 10.10.0.10, 127.0.0.1, 45.48.23.250).
   (a) Otherwise, the tool must quit with an error message and a usage message.
2. The port must be an integer between 1,025 and 65,535.
   (a) Otherwise, the tool must quit with an error message and a usage message.
3. The name of the file must be a relative or absolute path on the sending computer.
4. If the file name isn't found or can't be opened, the tool must quit with a usage message.
5. The data packet size must be an integer larger than 0 and less than the maximum integer representable on the computer (MAX_INT).
   (a) Otherwise, the tool must quit with an error message and a usage message.
6. If a parameter is missing, the tool must quit with a usage message.
7. If all parameters are valid, the sending tool must start to send the file to the destination address and port using UDP packets of the size given (in bytes).
8. After sending a packet, the sending tool must wait for a response (ACK) from the recipient before continuing to the next packet.
9. After finishing sending all data packets, the sending tool must send a packet with only -1 inside.
10. The tool must output a log of each packet sent (including packet number), each packet ACKed (including packet number), and a final success message (final packet and file name). Example outputs are shown below.

2.1.2 Test Cases (45 points total)

The following are test cases assuming the tool is called `UdpStopWaitClient-5784.jar` for files file1.pdf, file2.txt (provided in the starter repository). All tests must be preceded by `java -jar`, but they are elided for brevity for all but the first test case.

1. `java -jar UdpStopWaitClient-5784.jar -dest=localhost -port=5000 -packetsize=150 -f=file1.pdf`

   There are 1991 data packets followed by the final packet. Expected output:

   ```
   Sent packet 1
   Received ACK on 1
   Sent packet 2
   Received ACK on 2
   Sent packet 3
   Received ACK on 3
   ...
   Sent packet 1990
   Received ACK on 1990
   Sent packet 1991
   Received ACK on 1991
   Sent final packet for file1.pdf.
   Received ACK on 1992
   ```

2. `UdpStopWaitClient-5784.jar -dest=localhost -port=5001 -packetsize=200 -f=file1.pdf`

   There are 1493 data packets followed by the final packet. Expected output:

```
Sent packet 1
Received ACK on 1
Sent packet 2
Received ACK on 2
...
Sent packet 1492
Received ACK on 1492
Sent packet 1493
Received ACK on 1493
Sent final packet for file1.pdf.
Received ACK on 1494
```

3. `UdpStopWaitClient-5784.jar -dest=localhost -port=5002 -packetsize=250 -f=file1.pdf`

   There are 1195 data packets followed by the final packet. Expected output:

   ```
   Sent packet 1
   Received ACK on 1
   ...
   Sent packet 1195
   Received ACK on 1195
   Sent final packet for file1.pdf.
   Received ACK on 1196
   ```

4. `UdpStopWaitClient-5784.jar -dest=localhost -port=5003 -packetsize=300 -f=file1.pdf`

   There are 996 data packets followed by the final packet. Expected output:

   ```
   Sent packet 1
   Received ACK on 1
   ...
   Sent packet 996
   Received ACK on 996
   Sent final packet for file1.pdf.
   Received ACK on 997
   ```

5. `UdpStopWaitClient-5784.jar -dest=localhost -port=5004 -packetsize=1000 -f=file1.pdf`

   There are 299 data packets followed by the final packet. Expected output:

   ```
   Sent packet 1
   Received ACK on 1
   ...
   Sent packet 299
   Received ACK on 299
   Sent final packet for file1.pdf.
   Received ACK on 300
   ```

6. `UdpStopWaitClient-5784.jar -dest=127.0.0.1 -port=5005 -packetsize=1500 -f=file1.pdf`

   There are 200 data packets followed by the final packet. Expected output:

   ```
   Sent packet 1
   Received ACK on 1
   ...
   Sent packet 200
   Received ACK on 200
   ```

```
Sent final packet for file1.pdf.
Received ACK on 201
```

7. `UdpStopWaitClient-5784.jar -dest=localhost -port=5006 -packetsize=152 -f=file2.txt`

   There are 6899 data packets followed by the final packet. Expected output:

   ```
   Sent packet 1
   Received ACK on 1
   ...
   Sent packet 6899
   Received ACK on 6899
   Sent final packet for file2.txt.
   Received ACK on 6900
   ```

8. `UdpStopWaitClient-5784.jar -dest=localhost -port=5007 -packetsize=201 -f=file2.txt`

   There are 5217 data packets followed by the final packet. Expected output:

   ```
   Sent packet 1
   Received ACK on 1
   ...
   Sent packet 5217
   Received ACK on 5217
   Sent final packet for file2.txt.
   Received ACK on 5218
   ```

9. `UdpStopWaitClient-5784.jar -dest=localhost -port=5006 -packetsize=352 -f=file2.txt`

   There are 2979 data packets followed by the final packet. Expected output:

   ```
   Sent packet 1
   Received ACK on 1
   ...
   Sent packet 2979
   Received ACK on 2979
   Sent final packet for file2.txt.
   Received ACK on 2780
   ```

10. `UdpStopWaitClient-5784.jar -dest=localhost -port=5005 -packetsize=1790 -f=file2.txt`

    There are 586 data packets followed by the final packet. Expected output:

    ```
    Sent packet 1
    Received ACK on 1
    ...
    Sent packet 586
    Received ACK on 586
    Sent final packet for file2.txt.
    Received ACK on 587
    ```

11. `UdpStopWaitClient-5784.jar -dest=localhost -port=5004 -packetsize=2020 -f=file2.txt`

    There are 520 data packets followed by the final packet. Expected output:

    ```
    Sent packet 1
    Received ACK on 1
    ...
    Sent packet 520
    ```

```
Received ACK on 520
Sent final packet for file2.txt.
Received ACK on 521
```

12. `UdpStopWaitClient-5784.jar -dest=localhost -port=5003 -packetsize=2048 -f=file2.txt`

    There are 512 data packets followed by the final packet. Expected output:

    ```
    Sent packet 1
    Received ACK on 1
    ...
    Sent packet 512
    Received ACK on 512
    Sent final packet for file2.txt.
    Received ACK on 513
    ```

13. `java -jar UdpStopWaitClient-5784.jar -dest=localhost -port=5 -packetsize=2048 -f=file2.txt`

    ```
    Error: port must be between 1025 and 65535
    Syntax: UdpStopWaitClient-5784 -dest=ip -port=p -f=filename -packetsize=s
    ```

14. `java -jar UdpStopWaitClient-5784.jar -dest=1.2.3.4.5 -port=5003 -packetsize=2048 -f=file2.txt`

    ```
    Error parsing destination address: 1.2.3.4.5: Name or service not known
    Syntax: UdpStopWaitClient-5784 -dest=ip -port=p -f=filename -packetsize=s
    ```

15. `java -jar UdpStopWaitClient-5784.jar -dest=localhost -port=5003 -packetsize=5.6 -f=file2.txt`

    ```
    Error parsing packet size: For input string: "5.6"
    Syntax: UdpStopWaitClient-5784 -dest=ip -port=p -f=filename -packetsize=s
    ```

16. `UdpStopWaitClient-5784.jar -dest=localhost -port=5003 -packetsize=2048`

    ```
    Syntax: UdpStopWaitClient-5784 -dest=ip -port=p -f=filename -packetsize=s
    ```

17. `UdpStopWaitClient-5784.jar -port=5003 -packetsize=2048 -f=file2.txt`

    ```
    Syntax: UdpStopWaitClient-5784 -dest=ip -port=p -f=filename -packetsize=s
    ```

18. `UdpStopWaitClient-5784.jar -dest=localhost -packetsize=2048 -f=file2.txt`

    ```
    Syntax: UdpStopWaitClient-5784 -dest=ip -port=p -f=filename -packetsize=s
    ```

19. `UdpStopWaitClient-5784.jar -dest=localhost -port=5003 -f=file2.txt`

    ```
    Syntax: UdpStopWaitClient-5784 -dest=ip -port=p -f=filename -packetsize=s
    ```

2.2 Receiving Tool

The receiving tool shall be a command line tool that accepts the following command line parameters in the following order:

ip The IP address to listen on

port The port to listen on

outfile The name of the file to write out to

2.2.1 ACK Packet Format

The receiver must send ACK packets to the sender after receiving data packets. You may design the ACK packet however you want, but a recommended format for the packet is just to send the Packet Number received as an integer.

2.2.2 Requirements

The parameters must meet the following requirements:

1. The IP address must be in the standard IPv4 four byte dot separated format (*e.g.* 10.10.0.10, 127.0.0.1, 45.48.23.250).
   (a) Otherwise, the tool must quit with an error message and usage message.
2. If the recipient tool can't listen on the provided address (forbidden, not available), the tool must quit with an error message.
3. The port must be an integer between 1,025 and 65,535.
   (a) Otherwise, the tool must quit with an error message and usage message.
4. If the recipient tool can't listen on the port provided, the tool must quit with an error message.
5. The name of the file must be a valid file name without illegal characters that can be written to.
   (a) Otherwise, the tool must quit with an error message and usage message.
6. If all parameters are valid, the receiving tool must listen on the address and port for UDP packets.
7. If parameters are missing, print a usage message.
8. After receiving a packet, the receiving tool must send an ACK to the sender.
9. After receiving a packet containing only the -1 byte, the receiving tool must send an ACK message to the sender, and close the file being written to, then quit.
10. The tool must output a log of each packet received and ACKed (including packet number) and a final success message (final packet and file name). Example outputs are shown below.

2.2.3 Test Cases (45 points total)

The following are test cases assuming the tool is called `UdpStopWaitServer-5784.jar`. The test cases correspond to the tests for the client program (test 1 for test 1, test 2 for test 2, etc.). The number of packets here assumes the tests were run in accordance with the packet sizes for the sender tests in Section 2.1.2. All tests must be preceded by `java -jar`, but that is elided for brevity for all but the first test case.

1.
```
java -jar UdpStopWaitServer-5784.jar -ip=0.0.0.0 -port=5000 -outfile=outfiles/Test1-file1.pdf
```
   1992 packets should be received. Expected output:
```
Listening...
Received and acked: 1
Listening...
Received and acked: 2
. . .
Listening...
Received and acked: 1991
Listening...
Received and acked: 1992
File outfiles/Test1-file1.pdf completed. Received 1992 packets.
```

2. `UdpStopWaitServer-5784.jar -ip=0.0.0.0 -port=5001 -outfile=outfiles/Test2-file1.pdf`

   1494 packets should be received. Expected output:

   ```
   Listening...
   Received and acked: 1
   Listening...
   Received and acked: 2
   . . .
   Received and acked: 1493
   Listening...
   Received and acked: 1494
   File outfiles/Test2-file1.pdf completed. Received 1494 packets.
   ```

3. `UdpStopWaitServer-5784.jar -ip=0.0.0.0 -port=5002 -outfile=outfiles/Test3-file1.pdf`

   1196 packets. Output as above.

4. `UdpStopWaitServer-5784.jar -ip=0.0.0.0 -port=5003 -outfile=outfiles/Test4-file1.pdf`

   997 packets. Output as above.

5. `UdpStopWaitServer-5784.jar -ip=0.0.0.0 -port=5004 -outfile=outfiles/Test5-file1.pdf`

   300 packets. Output as above.

6. `UdpStopWaitServer-5784.jar -ip=0.0.0.0 -port=5005 -outfile=outfiles/Test6-file1.pdf`

   201 packets. Output as above.

7. `UdpStopWaitServer-5784.jar -ip=0.0.0.0 -port=5006 -outfile=outfiles/Test7-file2.txt`

   6900 packets. Output as above.

8. `UdpStopWaitServer-5784.jar -ip=0.0.0.0 -port=5007 -outfile=outfiles/Test8-file2.txt`

   5218 packets. Output as above.

9. `UdpStopWaitServer-5784.jar -ip=0.0.0.0 -port=5006 -outfile=outfiles/Test9-file2.txt`

   2980 packets. Output as above.

10. `UdpStopWaitServer-5784.jar -ip=0.0.0.0 -port=5005 -outfile=outfiles/Test10-file2.txt`

    587 packets. Output as above.

11. `UdpStopWaitServer-5784.jar -ip=0.0.0.0 -port=5004 -outfile=outfiles/Test11-file2.txt`

    521 packets. Output as above.

12. `UdpStopWaitServer-5784.jar -ip=0.0.0.0 -port=5003 -outfile=outfiles/Test12-file2.txt`

    513 packets. Output as above.

13. `UdpStopWaitServer-5784.jar -ip=0.0.0.0 -port=5 -outfile=outfiles/Test12-file2.txt`

    ```
    Error: port must be between 1025 and 65535
    Syntax: UdpStopWaitServer-5784 -ip=ip -port=p -outfile=f
    ```

14. `UdpStopWaitServer-5784.jar -ip=0.0.0.0 -port=200000 -outfile=outfiles/Test12-file2.txt`

    ```
    Error: port must be between 1025 and 65535
    Syntax: UdpStopWaitServer-5784 -ip=ip -port=p -outfile=f
    ```

15. `UdpStopWaitServer-5784.jar -ip=1.2.3.4.5 -port=5000 -outfile=outfiles/Test12-file2.txt`

    Error parsing listening address: 1.2.3.4.5: Temporary failure in name resolution
    Syntax: UdpStopWaitServer-5784 -ip=ip -port=p -outfile=f

16. `UdpStopWaitServer-5784.jar -ip=0.0.0.0 -port=5.6 -outfile=outfiles/Test12-file2.txt`

    Error parsing port: For input string: "5.6"
    Syntax: UdpStopWaitServer-5784 -ip=ip -port=p -outfile=f

17. `UdpStopWaitServer-5784.jar -ip=0.0.0.0 -outfile=outfiles/Test12-file2.txt`

    Syntax: UdpStopWaitServer-5784 -ip=ip -port=p -outfile=f

18. `UdpStopWaitServer-5784.jar -port=123 -outfile=outfiles/Test12-file2.txt`

    Syntax: UdpStopWaitServer-5784 -ip=ip -port=p -outfile=f

19. `UdpStopWaitServer-5784.jar -ip=0.0.0.0 -port=123`

    Syntax: UdpStopWaitServer-5784 -ip=ip -port=p -outfile=f

2.3 Documentation (5 points)

Add Javadoc documentation to every method and class. The documentation for methods must include:

- A 1-2 sentence summary of the method's purpose
- @param entries for all parameters, including what they are used for
- @return entry with a 1-2 sentence description of the return value
- @throws entries for any exceptions thrown.

The documentation for classes must includes:

- A 2-3 sentence description of the class' purpose
- @author the code's author
- @version a version for the class. Update on every commit to the repository.

3 Sending Experiments (20 points)

For this part of the assignment you will need to use two physical computers. That can be two laptops or a laptop and a desktop computer. Put stop and wait server on one computer and the client on the second computer. Perform the following experiments and report your results in the files as mentioned below

All of the experiments use the file1.pdf file provided in the starter repository.

3.1 Experiment setup

Download iperf (Linux tool) and use it to measure the bandwidth between the computers. Use ping to measure the RTT between the two computers.

Record the outputs of iperf in a file called iperf.txt. Record the outputs of ping in a file called ping.txt.

3.2 Experiment 1: Small packets

3.2.1 Sending on the network

Use the sending tool to send file1.pdf with packets of size 100B. Use the Linux `time` program to calculate the total sending time.

Use Wireshark to record the whole conversation. Save the UDP packets from the conversation in a PCAP file. Use the recording to calculate the size of the ACKs sent by the server. Write down all of the numbers that you get from the experiment.

3.2.2 Calculating sending times

Use the sending with ACKs calculator to calculate how long the sending should take between the machines using the following parameters:

- RTT as measured using ping
- Bandwidth as measured
- Packet size of 100B
- ACK size based on the actual ACK size you used in your network sending tool and as recorded by Wireshark.

3.2.3 Analysis

Answer the following questions about the experiment:

Q1: Are the actual and calculated sending times close or identical?

(1-2 sentences)

Q2: If they are not identical or close, explain why they differ.

(50-60 words.)

3.3 Experiment 2: Medium packets

We will now perform the experiment above again with same file, but with a larger packet size.

3.3.1 Sending on the network

Use the sending tool to send file1.pdf with packets of size 1KB. Use the Linux `time` program to calculate the total sending time.

Use Wireshark to record the whole conversation. Save the UDP packets from the conversation in a PCAP file. Use the recording to calculate the size of the ACKs sent by the server. Write down all of the numbers that you get from the experiment.

3.3.2 Calculating sending times

Use the second calculator above to calculate how long the sending should take between the machines using the following parameters:

- RTT as measured
- Bandwidth as measured
- Packet size of 1KB

- ACK size based on the actual ACK size you used in your network sending tool. Use Wireshark to figure out how large your ACK packets are.

3.3.3 Analysis

Answer the following questions about the experiment:

Q1: Are the actual and calculated sending times close or identical?

(1-2 sentences)

Q2: If they are not identical or close, explain why they differ.

(50-60 words.)

3.4 Experiment 3: Giant packets

We will now perform the experiment above again with same file, but with a larger packet size.

3.4.1 Sending on the network

Use the sending tool to send file1.pdf with 10KB packets. Use the Linux `time` program in to calculate the total sending time.

Use Wireshark to record the whole conversation. Save the UDP packets from the conversation in a PCAP file. Use the recording to calculate the size of the ACKs sent by the server. Write down all of the numbers that you get from the experiment.

3.4.2 Calculating sending times

Use the second calculator above to calculate how long the sending should take between the machines using the following parameters:

- RTT as measured
- Bandwidth as measured
- Packet size of 10KB
- ACK size based on the actual ACK size you used in your network sending tool. Use Wireshark to figure out how large your ACK packets are.

3.4.3 Analysis

Answer the following questions about the experiment:

Q1: Are the actual and calculated sending times close or identical?

(1-2 sentences)

Q2: If they are not identical or close, explain why they differ.

(50-60 words.)

3.5 What to turn in for experiments

Turn in the following files in the repository:

- iperf.txt - output from iperf

- ping.txt - output from ping
- Wireshark PCAP recordings from Experiment 1, Experiment 2, Experiment 3.
- 331-Assignment2-Section3-Questions.txt - template file filled in with responses to questions above. Fill in the template file given in the starter repository. I marked [Fill me in] in the template file where you must fill in information or text.