ICSI 516 Fall 2018 —Project 1

Due October 26th at 11:59PM via Blackboard

Objectives:

There are a number of objectives to this assignment. The first is to make sure you have some experience developing a network-based socket application. Second, because you are allowed to use any references you find on the Internet (including copies of existing code!), this assignment will help you see just how many network programming aids are available. Third, you will get a first-hand experience in comparative evaluation of protocol performance. Finally, having just a bit of practical experience will put a lot of the protocol concepts we learn into perspective.

Reading:

I would recommend you go over Chapter 2.7 before you begin your implementation. You can also use various online resources on socket programming.

Important NOTE on Formatting and Code Compliance:

Your code should run on our course VM csi516-fa18.arcc.albany.edu. This is where your submissions will be automatically graded. You may loose points if your code does not comply with the assignment's formatting requirements and does not run on the server.

Assignment Overview:

This assignment asks you to develop and evaluate a web-based FTP (File Transfer Protocol) application for simple FTP commands. You will complete this assignment in two phases. In the first phase you will develop two versions of the FTP:

- One that uses TCP as a transport protocol, and one that uses UDP for transport and implements basic stop-and-wait reliability at the application layer.
- In the second phase you will evaluate and compare the performance of the two implementations in terms of achieved throughput and delay.

Upon successful completion of this assignment, you need to submit your code from phase 1, Wireshark traces from phase 2 and a report on your implementation and findings. Pay close attention to the assignment details and submission instructions below.

Assignment Details:

========== Phase 1 -- Implementation ==========

The goal of this assignment is to implement a simple client-server system. You will need to use Java. The basic functionality of the system is implementation of FTP using a remote server.

The File Transfer Protocol (FTP) is a standard network protocol used for the transfer of computer files between a client and a server using the Internet. FTP uses a client-server application architecture with a separate control and data connections between the client and the server. The control connection is used to transmit commands from the client to the server and command output from the server back to the client, whereas the data connection is used to exchange files.

Your programs will begin with a startup request sent by the client to the server. This request establishes the control and data path. The client can then send commands and receive responses to/from the server.

Your clients must implement the following 4 commands:

`cd <dir>` :- Change directory

`ls <dir>` :- List the contents of the current directory

`put <file>` :- Copy a file from client to server

`get <file>` :- Copy a file from server to client

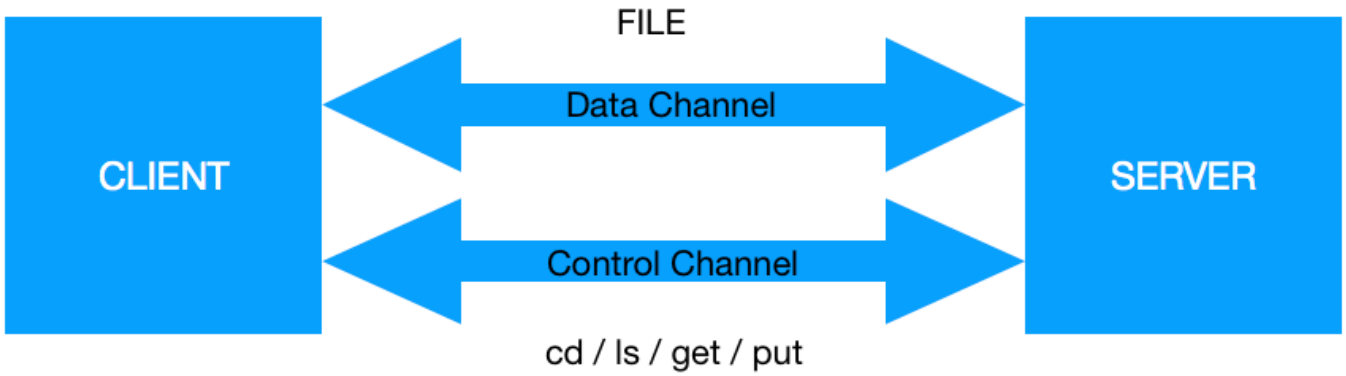


Fig. 1: FTP on Client Server Architecture

The commands `cd, ls, get, put` will be sent through the control channel and the data from the `<file>` for `get` and `put` will be sent on a separate data channel (refer fig. 1)

Your server will start in a "passive mode", in other words, it will listen to a specified port for instructions from the client. Separately, the client will be started and will connect to the server on a given host name or IP address plus a port number. The client should take as input from the user the host name or IP address plus the port number and an command to be executed on the server as follows:

1. Enter Server name or IP address:
2. Enter port:
3. Enter the command:

and should accept a single line for each query. The client-server connection should be retained after each command execution until the user inputs `exit` upon which the link is terminated.

The client should make sure that the specific port is valid, in other words, the client should check if the port is in the range from 0 to 655535 and if not then display an error message: **"Invalid port Number, Terminating"**

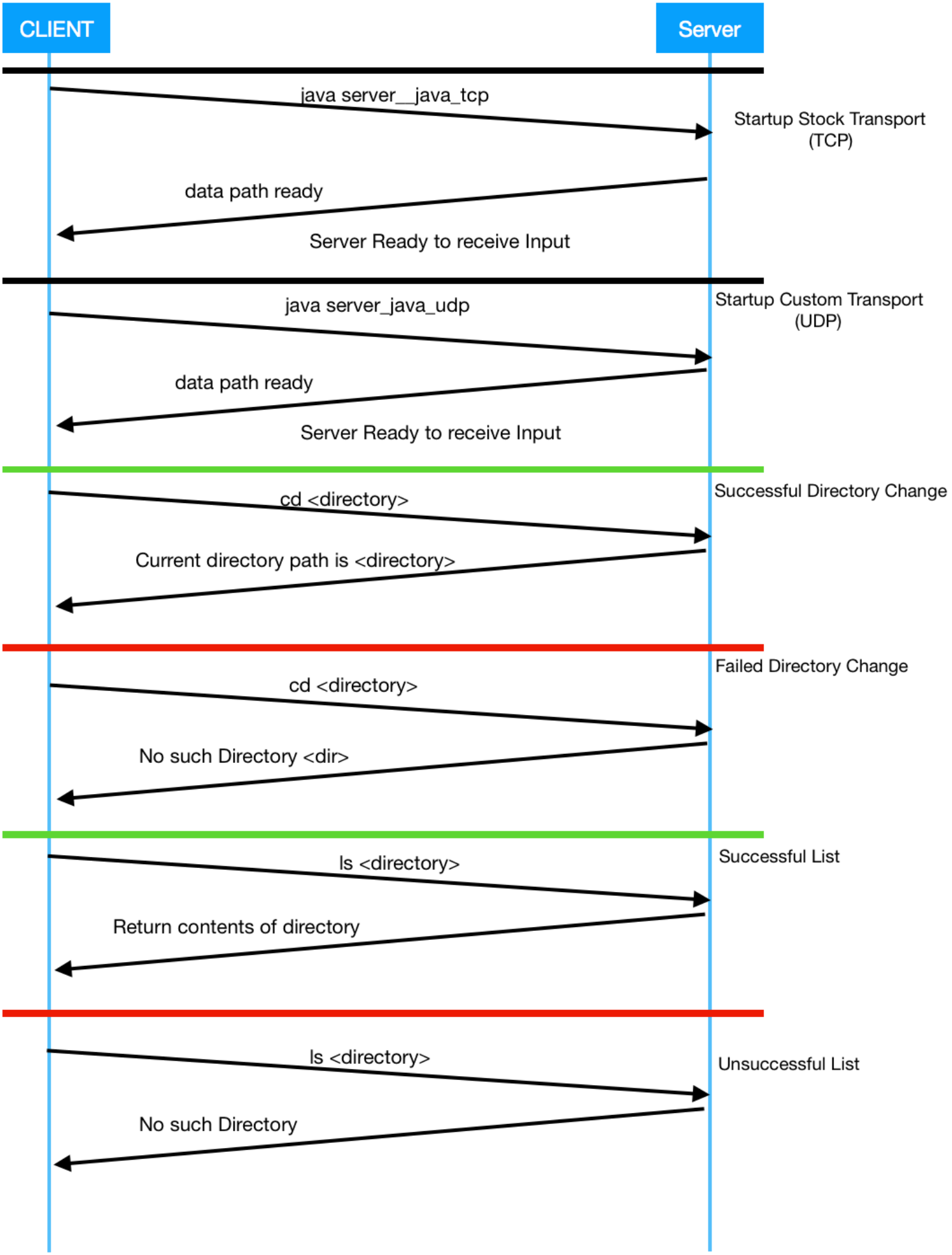
The client, then, will try to connect to the specified server on the specified port and transmit the command to the server.

The interaction flow between the client and server is as follows:

1. The server starts and waits for a connection to be established by the client
2. When an command is received, the server will:
   - Execute the command and send the results back to the client
3. Finally, the client will receive the results and display it to the user.

The interaction flow for all the 4 FTP commands can be referred from Fig. 2

Note that you have to handle application errors as detailed in Figure 2. You should use the exact same error message formatting, and same syntax to startup your programs to ensure that your code will pass our automatic grading.


CLIENT
Server
java server__java_tcp
Startup Stock Transport
(TCP)
data path ready
Server Ready to receive Input
java server_java_udp
Startup Custom Transport
(UDP)
data path ready
Server Ready to receive Input
cd <directory>
Successful Directory Change
Current directory path is <directory>
cd <directory>
Failed Directory Change
No such Directory <dir>
ls <directory>
Successful List
Return contents of directory
ls <directory>
Unsuccessful List
No such Directory

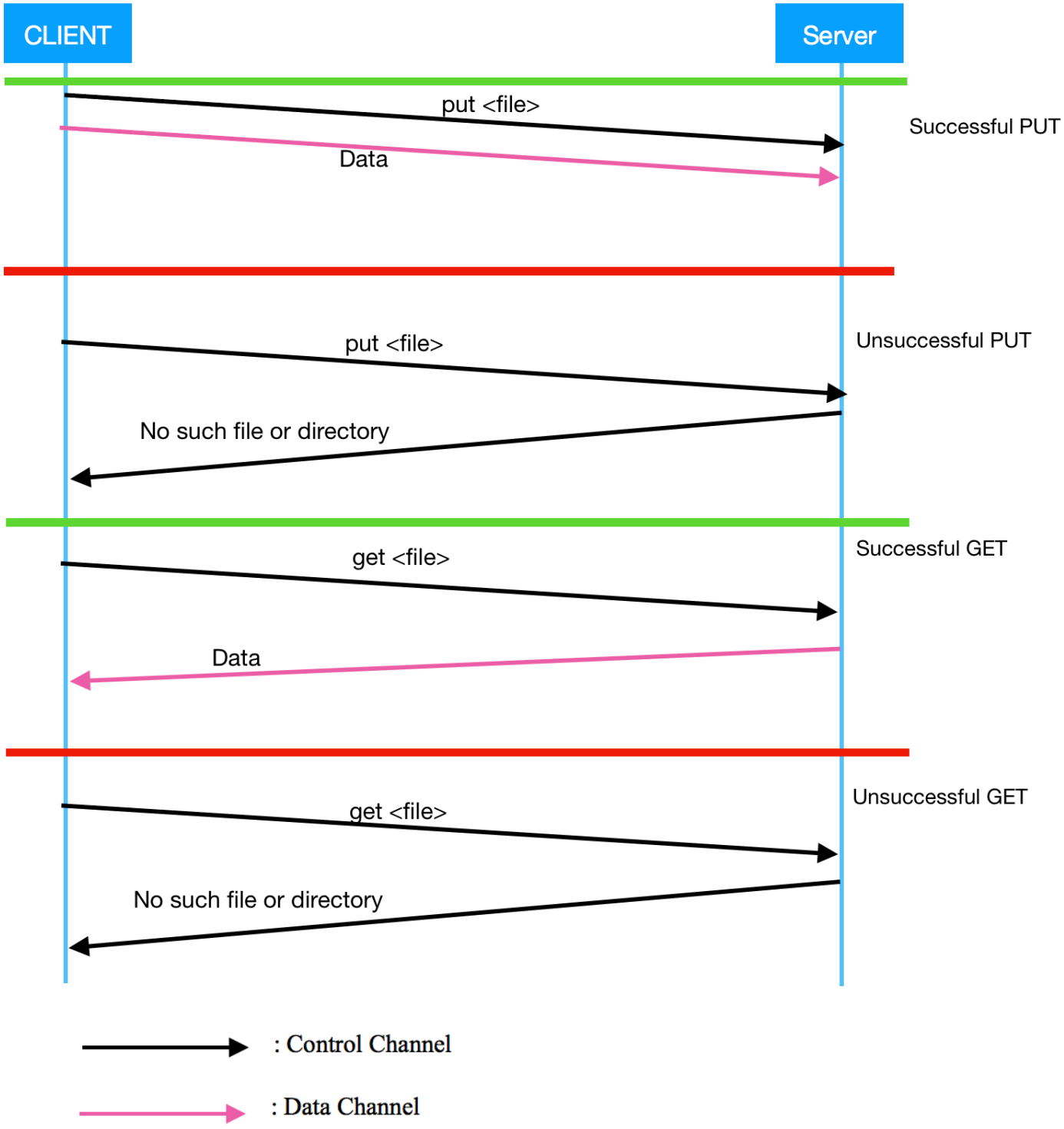


Fig. 2 : Application Level Flows

TCP and Reliable UDP:

You will implement your client and server using both TCP and UDP. This means you will be writing four different programs. (The clients and servers using TCP and UDP won't be interoperable so you won't be able to use the TCP client with the UDP server and vice versa.)
Your TCP implementation should be fairly straightforward since TCP provides connection setup, reliability, in-order delivery, etc. UDP, however, does not include any of these features so you will have to add a simple form of reliability in your application. You will do this by employing the stop-and-wait principle. Since we are working at the application level and there is no concept of packets, we will implement our stop and wait reliability at the level of message chunks. The client and server using Reliable UDP will operate as follows for a sample SUCCESSFUL PUT: (also refer toFig. 3)

1. Your client will calculate the message length and send to the server(a “length” message). Then, in a separate frame transmission, the client will send the message chunk (Chunk of file’s data, in this example)
2. Upon receiving of the length message, the server will wait up to 500 milliseconds for that number of bytes (or a fraction of it) to be sent. If it receives the correct number of bytes, it will respond with a string containing the characters “ACK” (a common abbreviation for acknowledgment). If it fails to receive the correct number of bytes by the end of timeout period, the server will give up, and display the following text on server console, and exit: “**Did Not Receive valid data from Client. Terminating.”**
3. If your client does not receive the ACK within 1 second, it will resend the length message the data chunk.
4. Your client should send, up to 3 times before giving up, closing the connection and terminating. Upon closing connection, the client should display following message: **“Failed to send data to server. Terminating.”**
5. The server will implement reliability in a similar fashion. First, it will compute the size of the result it has to send and then will send it to the client in a "length" message. The server will then send messages containing the output one by one and will wait for up to 1 second to receive an ACK for each message sent, before sending the next one. Similar to the client, the server will retry sending a single message up to three times before giving up. If the server did not receive an ACK after three attempts, it should display the following text on the server console and exit: "**Result transmission failed. Terminating.**”
6. Finally, upon successful reception of the last chunk, the server will send the total length of received message back to the client, where client will display it on its console.

NOTE: When sending large amounts of data over UDP, you might need to transmit multiple portions. This is because the UDP buffer fills up before the total amount of data is sent. Let's look at an example - say that your server needs to send a total of 5120 bytes but the UDP buffer is only 512 bytes. In this case, even though your server said that it is sending 5120 bytes, the UDP on the client delivers the data in 512 byte chunks because of the buffer size. A program that sends an ACK only if the received number of bytes is the same as the expected number of bytes

will fail to send an ACK in this case, which eventually will cause the connection to timeout and the server will fail to send result to the client. To avoid such problems, your programs will have to send an ACK for each received portion of data, keep track of how much of the entire data file had been received and display the result only if the total amount received is equal to the value sent in the length message.
You will want to test your system with large enough results to confirm that it works correctly. Ideally, you should test with the files provided with the assignment all the time.

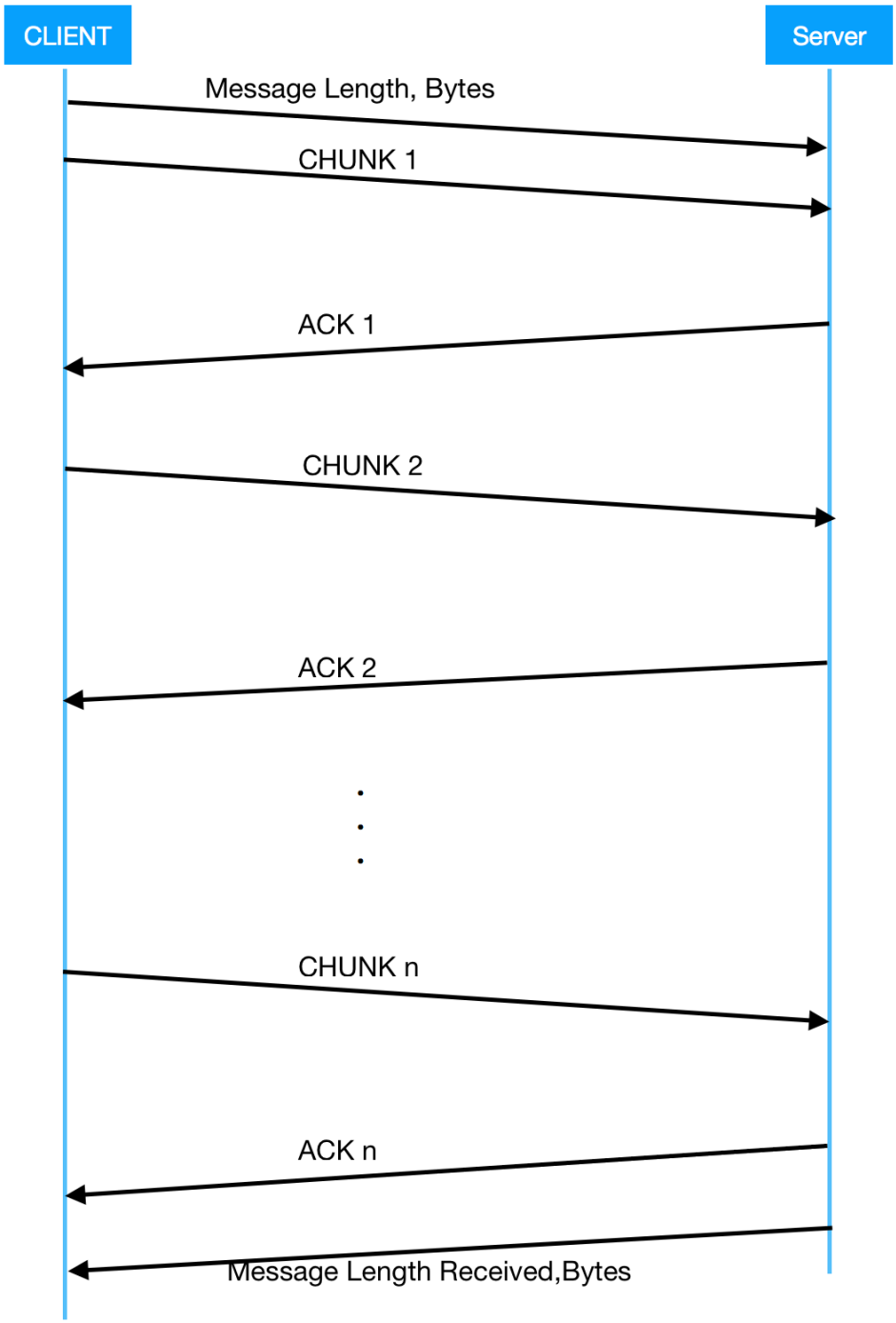


Fig. 3: Transport level Flow

Examples:

Below are some output examples, to help you format your programs display. The format of both the TCP and UDP programs should be the same - that is why we don't have separate examples for TCP and UDP. The only difference between running TCP and UDP programs is going to be the names of the programs to be run (see section "Language Choice and File Names" for naming conventions).

Starting the server:

You should start the server by issuing a command in a terminal with the following syntax (see below for correct naming of executables):

csi516-fa18% java server 3300

where "server" is the executable for the TCP or UDP program you wrote and "3300" is the port number you are going to use to establish connection

Client Input/Output Example:

You should start the client by issuing a command in a terminal with the following syntax (see below for correct naming of executables):

csi516-fa18% java client

where "client" is the executable for the TCP or UDP program you wrote

Here are some other examples of what the client might display depending on the user input.

```
csi516-fa18% java client_java_tcp
Enter server name or IP address: arcc.alb
Could not connect to server. Terminating.
csi516-fa18%

 * or *
csi516-fa18% java client_java_tcp
 Enter server name or IP address: 128.163.7.19
 Enter port: 99999
 Invalid port number. Terminating.
 csi516-fa18%

 * or *
csi516-fa18%  java client_java_tcp
Enter server name or IP address: 127.0.0.1
Enter port: 3300
Enter Command: cd ~/ok/CCN1
Failed to send data to Server. Terminating.
```

* or *

```
csi516-fa18% java client_java_tcp
Enter server name or IP address: 128.163.7.19
Enter port: 3300
Enter command: cd ~/ok/CCN1
Could not fetch result. Terminating.
csi516-fa18%
```

* or *

```
csi516-fa18% java client_java_tcp
Enter server name or IP address: localhost
Enter port: 3300
Enter command: ls ~/ok

        Applications  Users  dev   net   usr   Library
        Volumes       etc    private     var   Network
        bin     home  sbin   vm    System      cores  tmp
csi516-fa18%
```

Server Output Examples:

```
csi516-fa18% java server_java_tcp 3300
    Result transmission failed. Terminating.
csi516-fa18%
```

* or *

```
csi516-fa18% java server_java_tcp 3300
csi516-fa18%  Server Ready to receive Input
```

Language Choice and File Names:

You will use Java for this assignment. You must turn in exactly four programs.
Your programs should be named as indicated below and should run with NO errors and warning on the course VM csi516-fa18.arcc.albany.edu

For Java, the program names should be:

Client in Java using TCP: client_java_tcp.java
Server in Java using TCP: server_java_tcp.java
Client in Java using UDP: client_java_udp.java
Server in Java using UDP: server_java_udp.java

NOTE: Pay attention to all of these directions carefully. Make sure that you name your files and format your messages as specified in the assignment. An automated program will be used for grading and if there are any deviations, you will lose points!

========== Phase 2 -- Wireshark evaluation ==========

In this phase you will perform a comparative evaluation of your implementations in terms of overall delay and achieved throughput. We define overall delay as the relative time difference between the last and the first packet exchanged within a single program invocation. We define the achieved throughput as the total sum of bits exchanged within a single program invocation divided by the overall delay for that invocation. You will run your server and client implementations on different physical machines in order to account for a realistic Internet scenario. Specifically, you will run your server program on csi516-fa18.arcc.albany.edu (our course VM) and your client program on your personal computer. You will also run Wireshark on your personal computer to be able to record a packet trace for each program invocation. You will need to record four packet traces for each of the TCP and UDP implementation (so eight altogether) for four different files, as specified in the tables below. Once you have the traces, you need to process them offline and determine the overall delay and achieved throughput for each invocation.

Command / Implementation	File 1 (16KB)	File 2 (32KB)	File 3 (48KB)	File 4 (62KB)
Overall delay (TCP), sec				
Overall delay (UDP), sec				

Command / Implementation	File 1 (16KB)	File 2 (32KB)	File 3 (48KB)	File 4 (62KB)
Achieved Throughput (TCP), bps				
Achieved Throughput (UDP), bps				

You will use your '`get <FILE>`' command implementation in order to transmit the files from the server to the client and capture Wireshark Traces in this evaluation phase. The four files agains which you should evaluate are provided as a part of this assignment.

Note: You should run your experiments from the same network. For example, if you run your UDP experiments from campus and your TCP experiments from home, different delay characteristics of the campus and your home network will skew your results.

Preparing your report :
You need to submit a brief report (not more than 4 pages) on your implementation and findings. Your report should also include:

Results from Wireshark evaluation:

- *Your name and email address.*
- *A description of your methodology. How did you process the Wireshark results to calculate the above metrics. Did you use a program or did you do it manually?*
- *Two tables, using the same format as the ones above, with filled out values for overall delay and achieved throughput, calculated in your Wireshark analysis.*
- *A description of the trends you see in your results along with a justification of these trends.*

Grading Guidelines :

IMPORTANT NOTE ON FORMATTING AND CODE COMPLIANCE : Your code (both client and server) should run on our course Virtual machine csi516-fa18.arcc.albany.edu. This is where your code will be automatically graded. Points may be taken out if your code does not comply with the formatting requirements and does not run on the server.

You may use pieces of code from the Internet to help you do this assignment (e.g. basic socket code). However, this is just like citing a passage from a book, so if you copy code, you must cite it. To do this, put a comment at the beginning of your code that explains exactly what you have copied, who originally wrote it, and where it came from.
Below is a breakdown of points for this assignment. In addition to correctness, part of the points count towards how well code is written and documented. NOTE: good code/documentation does not imply that more is better. The goal is to be efficient, elegant and succinct!

- *35 pts: TCP implementation*
- *25 pts: UDP implementation*
- *20 pts: Wireshark evaluation*
- *15 pts: Report*
- *5 pts: Code documentation/Proper references*

Final Warning:
This is an assignment you definitely want to start on early. The design of the assignment is such that it is nearly impossible to provide all of the details you need.

Instead of assuming things should be done a particular way, **ask questions!** Use every opportunity to meet with the instructor and TA and send them emails with questions. Answers that are relevant to everyone in class will be posted on the Blackboard discussion forum.

Assignment Turn-in:

The assignment should be submitted using the course Blackboard page. You need to submit a total of 13 files as follows:

The programs for your UDP and TCP implementation (4 files).
Report (1 file).
Your pcap traces from wireshark(8 files).

Because the web site only allows one file to be submitted, you should use zip to combine all your files into a single archive of the format <lastname_firstname>.zip and submit this archive for grading.

Cheating Policy :

Cheating is not tolerated. Please, read the university Community Rights and Responsibilities for more information on cheating. Students caught cheating will receive 0 points for the assignment and will be reported. Of particular relevance to this assignment is the need to **properly cite material you have used**. Failure to do so constitutes plagiarism.