Networks and Distributed Systems	Project 2: Simple Transport Protocol
CS3700 Spring 2015	September 21, 2014

This project is due at 11:59:59pm on March 5, 2015 and is worth 15% of your grade. You must complete it with a partner. You may only complete it alone or in a group of three if you have the instructor's explicit permission to do so for this project.

Note that there is a milestone deadline for this project, at 11:59:59pm on February 23, 2015. More details are in the Milestone section below.

1 Description

You will design a simple transport protocol that provides reliable datagram service. Your protocol will be responsible for ensuring data is delivered in order, without duplicates, missing data, or errors. Since the local area networks at Northeastern are far too reliable to be interesting, we will provide you with access to a machine that will emulate an unreliable network.

For the assignment, you will write code that will transfer a file reliably from between two nodes (a sender and a receiver). You do NOT have to implement connection open/close etc. You may assume that the receiver is run first and will wait indefinitely, and the sender can just send the data to the receiver.

2 Requirements

You have to design your own packet format and use UDP as a carrier to transmit packets. Your packet might include fields for packet type, acknowledgement number, advertised window, data, etc. This part of the assignment is entirely up to you. Your code MUST:

- The sender must accept data from STDIN, sending data until EOF is reached
- The sender and receive must work together to transmit the data reliably
- The receiver must print out the received data to STDOUT in order and without errors
- The send and receiver must print out specified debugging messages to STDERR
- Your sender and receiver must gracefully exit
- Your code must be able to transfer a file with any number of packets dropped, damaged, duplicated, and delayed, and under a variety of different available bandwidths and link latencies

You may implement any reliability algorithm(s) you choose. However, more sophisticated algorithms (i.e., those which perform better) will be given higher credit. For example, some desired properties include (but are not limited to):

- Fast: Require little time to transfer a file.

- Low overhead: Require low data volume to be exchanged over the network, including data bytes, headers, retransmissions, acknowledgements, etc.

Regardless, correctness matters most; performance is a secondary concern. We will test your code and measure these two performance metrics; better performance will result in higher credit. Remember that network-facing code should be written defensively. Your code should check the integrity of every packet received. We will test your code by corrupting packets, reordering packets, delaying packets, and dropping packets; you should handle these errors gracefully, recover, and *not* crash.

3 Your programs

For this project, you will submit two programs: a *sending* program `3700send` that accepts data and sends it across the network, and a *receiving* program `3700recv` that receives data and prints it out in-order. You must use C to implement both programs, and we will give you basic starter code (see below). You may *not* use any transport protocol libraries in your project (such as TCP); you must use UDP. You must construct the packets and acknowledgements yourself, and interpret the incoming packets yourself.

3.1 Starter code

Very basic starter code for the assignment is available in `/course/cs3700sp15/code/project2`. You must use this code as a basis for your project. Provided is a simple implementation that sends one packet at a time; it does not handle any packet retransmissions, delayed packets, or duplicated packets. So, it will work if the network is perfectly reliable. Moreover, if the latency is significant, the implementation will use very little of the available bandwidth.

To get started, you should copy down this directory into your own local directory (i.e., `cp -r /course/cs3700sp15/code/project2 ~/`). You can compile your code by running `make`. You can also delete any compiled code and object files by running `make clean`.

3.2 Requirements

The command line syntax for your sending is given below. The client program takes command line argument of the remote IP address and port number, and the name of the file to transmit. The syntax for launching your sending program is therefore:

```
./3700send <recv_host>:<recv_port>
```

`recv_host` (Required) The IP address of the remote host in `a.b.c.d` format.

`recv_port` (Required) The UDP port of the remote host.

To aid in grading and debugging, your sending program should print out messages to the console: When a sender sends a packet (including retransmission), it should print the following to STDERR:

```
<timestamp> [send data] start (length)
```

where `timestamp` is a timestamp (down to the microsecond), `start` is the beginning offset of the data sent in the packet, and `length` is the amount of the data sent in that packet. When your 3700send receives an acknowledgement, you should also print to STDERR

```
<timestamp> [recv ack] end
```

where `end` is the last offset that was acknowledged. You may also print some messages of your own to indicate timeouts, etc, depending on your design, but make it concise and readable; a function `mylog(char *fmt, ...)` is provided for this purpose.

The command line syntax for your receiving program is given below. The client program will start up and will bind to a local port—once bound, it will print out the following to STDERR:

```
<timestamp> [bound] port
```

The syntax for launching your sending program is therefore:

```
./3700recv
```

To aid in grading and debugging, your receiving program should print out messages to STDERR: When the receiver receives a valid data packet, it should print

```
<timestamp> [recv data] start (length) status
```

where `start` is the beginning offset of the data sent in the packet, and `length` is the amount of the data sent in that packet, and `status` is one of `ACCEPTED (in-order)`, `ACCEPTED (out-of-order)`, or `IGNORED`. If a corrupt packet arrives, it should print to STDERR

```
<timestamp> [recv corrupt packet]
```

Similar to 3700send, you may add your own output messages.

Both the sender and the receiver should print out a message to STDERR after completion of file transfer, and then exit:

```
<timestamp> [completed]
```

You should develop your client program on the CCIS Linux machines (ideally `cs3600tcp.ccs.neu.edu`), as these have the necessary compiler and library support. You are welcome to use your own Linux/OS X/Windows machines, but you are responsible for getting your code working, and your code *must* work when graded on `cs3600tcp.ccs.neu.edu`. If you do not have a CCIS account, you should get one ASAP in order to complete the project.

4 Testing your code

In order for you to test your code over an unreliable network, we have set up a machine that will configurably emulate a network that will drop, reorder, damage, duplicate, and delay your packets. This machine is part of the CCIS network, and you can log in to it with your CCIS username and credentials. If you have any problems accessing the machine, please post on Piazza or email `cs3700sp15-staff@ccs.neu.edu`.

4.1 Emulating a network

The machine is `cs3600tcp.ccs.neu.edu`; you should make sure you are able to `ssh` to the machine and run your code on it. You will need to use the loopback interface in order to leverage the emulated network. In other words, you might run something like `./3700recv` in one terminal, record the port it local binds to (say, 3992), and then run `./3700send 127.0.0.1:3992` in another terminal.

You may configure the emulated network conditions by calling the following program:

```
/course/cs3700sp15/bin/project2/netsim [--bandwidth <bw-in-mbps>]
       [--latency <latency-in-ms>] [--delay <percent>]
       [--drop <percent>] [--reorder <percent>]
       [--corrupt <percent>] [--duplicate <percent>]
```

`bandwidth` This sets the bandwidth of the link in Mbit per second. If not specified, this is 1 Mb/s.

`latency` This sets the latency of the link in ms. If not specified, this value is 10 ms.

`delay` This sets the percent of packets the emulator should delay. If not specified, this is 0.

`drop` This sets the percent of packets the emulator should drop. If not specified, this is 0.

`reorder` This sets the percent of packets the emulator should reorder. If not specified, this is 0.

`corrupt` This sets the percent of packets the emulator should introduce errors into. If not specified, this is 0.

`duplicate` This sets the percent of packets the emulator should duplicate. If not specified, this is 0.

Once you call this program, it will configure the emulator to delay/drop/reorder/mangle/duplicate *all* UDP and ICMP packets sent by or to you at the specified rate. For example, if you called

```
/course/cs3700sp15/bin/project2/netsim --bandwidth 0.5 --latency 100 --delay 20 --drop 40
```

the simulator will configure a network with 500 Kb/s bandwidth and a latency of 100 ms, and will randomly delay 20% of your packets and drop 40%. In order to reset it so that none of your packets are disturbed, you can simply call

```
/course/cs3700sp15/bin/project2/netsim
```

with no arguments. Note that the configuration is done on a per-user-account, rather than per-group, basis. The simulator is also stateful, meaning your settings will persist across multiple sessions.

4.2 Helper script

In order to make testing your code easier, we have also included a perl script that will launch your receiver, read the port number, launch your sender, feed the sender input, read the output from the receiver, compare the two, and print out statistics about the transfer. This script is included in the starter code, and you can run it by executing

```
./run
```

This script also takes a couple of arguments to determine what it should do:

```
./run [--size (small|medium|large|huge)] [--printlog] [--timeout <seconds>]
```

`size` The size of the data to send, including 1 KB (`small`), 10 KB (`medium`), 100 KB (`large`), MB (`huge`). Default is `small`.

`printlog` Instructs the script to print a (sorted) log of the debug output of `3700send` and `3700recv`. This may add significant processing time, depending on the amount of output.

`timeout` The maximum number of seconds to run the sender and receiver before killing them. Defaults to 30 seconds.

The output of this script include some statistics about the transfer:

```
bash$ ./run --size large
Time elapsed: 1734.921 ms
Packets sent: 140
Bytes sent: 107000
Effective goodput: 461.116 Kb/s
Data match: Yes
```

where `Data match` is whether the data was transferred correctly.

Note: The `run` script will *only* work on the `cs3600tcp.ccs.neu.edu` machine, as it assumes certain libraries exist. You should not run the script on other machines.

4.3 Testing script

Additionally, we have included a basic test script that runs your code under a variety of network conditions and also check your code's compatibility with the grading script. If your code fails in the test script we provide, you can be assured that it will fare poorly when run under the grading script. To run the test script, simply type

```
bash$ make test
```

This will compile your code and then test your DNS client on a number of inputs, comparing the results against the reference solution. If any errors are detected, the test will print out the expected and actual output.

Note: The testing script will *only* work on the `cs3600tcp.ccs.neu.edu` machine, as it changes the network emulation. You should not run the script on other machines.

4.4 Performance testing

As mentioned in class, 20% of your grade on this project will come from performance. Your project will be graded against the submissions of your peers. To help you know how you're doing, the testing script will also run a series of performance tests at the end; for each test that you successfully complete, it will report your time elapsed and bytes sent to a common data base. For example, you might see

```
Performance tests
  Huge 5Mb/s, 10 ms                                              [PASS]
    13.889 sec elapsed, 1.1MB sent
```

This indicates that you passed the test in 13.889 seconds and sent a total of 1.1MB of data (including retransmissions, overhead, etc). This score will be reported to the common database.

In order to see how your project ranks, you can run

```
amislove@cs3600tcp:~$ /course/cs3700sp15/bin/project2/printstats
----- TEST: Huge 5Mb/s, 10 ms -----
Quickest:
1:amislove                                          9.322 sec
2:othergroup                                       13.889 sec

Most Efficient:
1:othergroup                                        1.1MB sent
2:amislove                                          2.8GB sent
```

which will print out the rank of each group for each performance test, divided into time spend and bytes sent. In this particular example, `amislove`'s project is quicker but has higher overhead. Obviously, you would ideally have both lower time and fewer bytes sent.

5 Grading

The grading in this project will consist of

60% Program correctness

15% Performance

15% Style and documentation

10% Milestone functionality

By definition, you going to be graded on how gracefully you handle errors; your code should **never** print out incorrect data. Your code will definitely see corrupted packets, delays, duplicated packets, and so forth. You should always assume that everyone is trying to break your program. To paraphrase John F. Woods, "Always code as if the [the remote machine you're communicating with] will be a violent psychopath who knows where you live."

6 Submitting your project

6.1 Registering your team

You and your partner should first register as a team by running the `/course/cs3700sp15/bin/register` script. You should pick out a team name (no spaces or non-alphanumeric characters, please) and run

```
/course/cs3700sp15/bin/register project2 <teamname>
```

This will either report back success or will give you an error message. If you have trouble registering, please contact the course staff.

You must register your team by 11:59:59pm on February 15, 2015.

6.2 Milestone

In order to ensure that you are making sufficient progress, you will have an interim milestone deadline. For the milestone, your code must pass the Basic (friendly network) tests in the testing script. In other words, your code must correctly transmit data when the network reliably delivers your packets without any duplication, drops, etc.

You should submit your milestone by running the `/course/cs3700sp15/bin/turnin` script. Specifically, you should create a `project3` directory, and place all of your code in it. Then, run

```
/course/cs3700sp15/bin/turnin project2-milestone <dir>
```

Where `<dir>` is the name of the directory with your submission. The script will print out every file that you are submitting, make sure that it prints out all of the files you wish to submit! You should receive an email confirmation of your submission.

You must submit your milestone by 11:59:59pm on February 23, 2015. No slip days can be used on the milestone.

6.3 Final submission

For the final submission, you should submit your (thoroughly documented) code along with a plain-text (no Word or PDF) README file. In this file, you should describe your high-level approach, the challenges you faced, a list of properties/features of your design that you think is good, and an overview of how you tested your code. You should also describe the basic transport protocol that you implemented, and why you made that choice.

You should submit your project by running the `/course/cs3700sp15/bin/turnin` script. Specifically, you should create a `project2` directory, and place all of your code and README files in it. Then, run

```
/course/cs3700sp15/bin/turnin project2 <dir>
```

Where `<dir>` is the name of the directory with your submission. Again, the script will print out every file that you are submitting, make sure that it prints out all of the files you wish to submit! You should receive an email confirmation of your submission.

You must submit your project by 11:59:59pm on March 5, 2014.

7 Advice

A few pointers that you may find useful while working on this project:

- Start by getting your code working without any packet manipulation. You can check whether your code successfully transmits the file by using the included `run` script, or manually using `diff` and `md5sum` programs in Linux. Test your code with each type of manipulation separately and then in combination. Note that you have to introduce multiple reliability mechanisms (checksums, timeouts, retransmits, etc) in order to handle all of the possible errors.
- Check the Piazza forum for question and clarifications. You should post project-specific questions there first, before emailing the course staff.
- Finally, get started early and come to the instructor's office hours and TA lab hours. You are welcome to come to the lab and work, and ask the TA any questions you may have.