Fundamentals of Computer Networking **Project 2: Simple Transport Protocol**
CS4700/CS5700 Spring 2011 **14 February 2011**

This project is due at 11:59pm on March 21st, 2011.

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, 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 a named file 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:

- Transfer the file name reliably.
- Transfer the file contents reliably.
- The receiver must write the contents it receives into the local directory with the appropriate file name.
- 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.

You may implement any reliability algorithm(s) you choose. However, more sophisticated algorithms 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.

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.

2.1 Undergraduate Requirements

The requirements for the undergraduate version of this project are different from the graduate version; this was left out in the Project 2 description handed out previously. Particularly, the undergraduate version of this project can make the following assumptions:

- The file we ask your program to send will not be larger than 1 MB.
- We will not require that your program deal with corrupted packets (e.g., you can assume that packets will arrive the same as they are sent, and do not need to test with the `--mangle` option to `netsim`).

If you choose to support unlimited file sizes and corrupted packets, we will reward extra credit to your submission. However, they are not required for the project.

2.2 Graduate Requirements

Graduate submissions are required to support files of any size (in particular, sizes that are larger than can be fit into memory), and are also required to tolerate potentially corrupted packets.

3 Your programs

For this project, you will submit two programs: a *sending* program that sends the file across the network, and a *receiving* program that receives the file and stores it to local disk. You may use any language that you wish. If you are using a non-standard language (e.g., other than C, C++, Java, perl, or python), please email the course staff ASAP to ensure that the necessary libraries are in place. You may *not* use any transport protocol libraries in your project (such as `TCP`). You must construct the packets and acknowledgements yourself, and interpret the incoming packets yourself.

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:

```
sendfile -r <recv_host>:<recv_port> -f <filename>
```

`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.

`filename` (Required) The name of the file to send.

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:
  `[send data] start (length)`
  where start is the beginning offset of the file sent in the packet, and length is the amount of the file sent in that packet.
- You may also print some messages of your own to indicate receiving acknowledgement, time- out, etc, depending on your design, but make it concise and readable.

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:

```
recvfile -p <recv_port>
```

`recv_port` (Required) The UDP port to listen on.

To aid in grading and debugging, your receiving program should print out messages to the console:

- When the receiver receives a valid data packet, it should print:
  `[recv data] start (length) status`
  where status is one of `ACCEPTED(in-order)`, `ACCEPTED(out-of-order)`, or `IGNORED`.
- If a corrupt packet arrives, it should print
  `[recv corrupt packet]`
- Similar to sendfile, you may add your own output messages.
- The receiver should store a file with name `x` as `x.recv`. (This will allow you to use the same directory for both the receiver and the sender).

Both the sender and the receiver should print out a message after completion of file transfer, and then exit:
`[completed]`

If you use C, your executables should be named `sendfile` and `recvfile`. If you use another language, you *must* write two brief Bash shell scripts, named `sendfile` and `recvfile`, that conform to the input syntax above and then launches your program with whichever incantations are necessary. For example, if you write your solution in Java, your `sendfile` Bash script might resemble

```
#!/usr/bin/perl -w
$args = join(' ', @ARGV);
print `java -jar sendfile.jar $args`;
```

You should develop your client program on the CCIS Linux machines, 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 the CCIS Linux machines. 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. To use this machine, please email the course staff (`amislove@ccs.neu.edu` and `aghayev@ccs.neu.edu`) and provide your CCIS Linux username. We will then create an account for you.

The machine is `cs4700dns.ccs.neu.edu`. Once you receive an account, you will be able to `ssh` to the machine and run your code on it. You only need to run one end (either the sending or the receiving program) on this machine; you can run the other end on any other machine you choose. You may configure the emulated network conditions by calling the following program:

```
/usr/bin/netsim [--delay <percent>] [--drop <percent>]
                [--reorder <percent>] [--mangle <percent>]
```

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

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

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

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

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

```
/usr/bin/netsim --delay 20 --drop 40
```

the simulator 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

```
/usr/bin/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.

5 Submitting your project

You should submit your project by emailing the course staff at `amislove@ccs.neu.edu` and `aghayev@ccs.neu.edu`. You should submit a `.tar.gz` file as an attachment to the email, which contains your code. Please have the subject line for the email be `[CS4700] [Project 2 Submission]` followed by your last name and the last name of your other group member(s).

This `.tar.gz` file should unpack to have the following structure

```
project2-{lastname1}-{lastname2}/
                                 README
                                 sender.{c, java, pl, ...}
                                 receiver.{c, java, pl, ...}
                                 ....
```

If using a compiled language (e.g., C, Java, C++), you should include a `Makefile` which will compile your code. The `README` should have a brief description and explanation of the packet formats, protocols, and algorithms you use, a list of properties/features of your design that you think is good, as well as examples of how to run your code.

6 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 `diff` and `md5sum` programs in Linux. Test your code with each type of manipulation separately and then in combination. Note that you will likely have to introduce multiple reliability mechanisms (checksums, timeouts, retransmits, etc) in order to handle all of the possible errors.
- Check the Blackboard 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 TA lab hours - these are held from 4:00pm - 6:00pm on Wednesdays in the lab at 212 West Village H. You are welcome to come to the lab and work, and ask the TA and instructor any questions you may have.