

Reliable Data Transfer

Please read all sections of this document before you begin coding.

Objectives

In this MP, you will implement reliable data transfer. You will need to use the knowledge of the reliable data transfer principles discussed in class to design and implement a sender and receiver that transfer a file across an unreliable link.

Guidelines

For this MP, you are encouraged to work with another student in the class, although you may work individually (however, you will be graded on the same scale and responsible for the same amount of work). Please find a partner as soon as possible in order to get a head start on the MP. If you cannot find a partner, post a message on the newsgroup. If this does not work, email the instructor.

You may discuss interpretations of the MP with other people from the class, but any form of group work with people other than your partner is forbidden. This includes code-sharing, which will be checked electronically.

You are required to make sure that your programs compile and run on the EWS Linux machines. Your programs will be tested on EWS machines only. Programs suffering compilation errors when tested by the course staff will earn no credit.

You may find it useful to refer to the Unix man pages as necessary. You may also want to refer to the links posted on the course webpage. If you use code from another source, credit it in your README. Stevens book on UNIX Programming may also prove helpful.

If you are borrowing heavily from a single source, check with the TAs or professors to confirm that you are still in line with the class policies. When in doubt, get permission to use a source well before the due date.

Please indent and document your code. Use meaningful names for variables and follow other style guidelines that enhance the clarity of your code.

Follow the guidelines in the “Hand In” section below when turning in this assignment.

Specification

This MP will require you to create a sender and receiver that implement reliable transfer over the UDP protocol. You will be required to deal with dropped, duplicated, and reordered packets in your data transfer. You will also need to implement a sliding window protocol to support efficient data transfer over high-RTT links.

Sender

The sender should be run as follows:

```
sender filename listenport trollhost trollport
```

The sender will read the file specified by the filename and transfer it using a reliable transfer mechanism to the receiver sitting behind a troll. After completing the transfer, the sender should terminate and exit.

The sender will send data to a troll, sitting at `trollhost:trollport`, which will then forward data to the receiver. The sender should bind to `listenport` to receive acknowledgments and other signaling from the receiver. You should only use a single UDP socket for both sending and receiving data.

Receiver

The receiver should be run as follows:

```
receiver filename listenport trollhost2 trollport2
```

The receiver will need to bind to the UDP port specified on the command line and receive a file from the sender sent over the port. The file should be saved to `filename`. Note that you should make sure in your testing that the filename used by the receiver is not the same as the one used by the sender, since otherwise the receiver will overwrite the file and interfere with the sender.

The receiver will use the troll at `trollhost2:trollport2` to send acknowledgments and any other control signals to the sender. The receiver should exit once the transfer is complete.

Troll

To test your algorithm, use a “network troll,” a program that selectively drops, delays, and reorders packets. The troll program is located in `ece438/MP3/troll.py`, and its usage is as follows:

```
Usage: troll.py -i port -h desthost [options] listenport
-s delay: delay in ms
-m perc: duplicate percentage of packets
-g perc: garble percentage of packets
-r perc: reorder percentage of packets
-x perc: destroy (drop) percentage of packets
```

The troll will listen for UDP packets on `listenport` and forward them to `desthost:port`, after potentially dropping, delaying, reordering, or duplicating the packets. To set up a test, you will need to use two trolls, one for sending packets from sender to the receiver and once for sending packets back from the receiver to the sender. For example, you can run the programs as follows:

```
> receiver outfile 5000 localhost 5003
> troll -h localhost -i 5000 -s 500 -x 10 5002
> troll -h localhost -i 5001 -s 500 -x 10 5003
> sender infile 5001 localhost 5002
```

In this case, the sender will send files to `localhost:5002`, which will be received by troll #1 and sent to the receiver, who is listening on `localhost:5000`. The receiver will send acknowledgments to the second troll on port `localhost:5003`, who will forward them to the sender on port 5001. The communication pattern is shown in Figure 1.

Note that although UDP will allow you to send large packets using IP fragmentation, we will configure the troll to discard any packets larger than 1472 bytes (in payload), so you should make sure that you restrict your packets to that size.

The troll can also garble packets (`-g` switch). For extra credit, you may choose to implement error detection in your code. Please note this in your design document so that we know to test it.

Design Document

To ensure that you are making sufficient progress on the project, you will be required to submit a design document as part of your project. The document must be handed in **in class on November 28th**; feedback will be provided to you by the following lecture about any potential problems in your project. You will also submit a revised version of this document with your project.

The design document should include the following components:

1. *Algorithm Overview*

Describe the details of your reliable transfer algorithm. You are welcome to use any of the algorithms described in class verbatim, or to combine ideas from the different algorithms we discussed. Draw the sender and the receiver state machines.

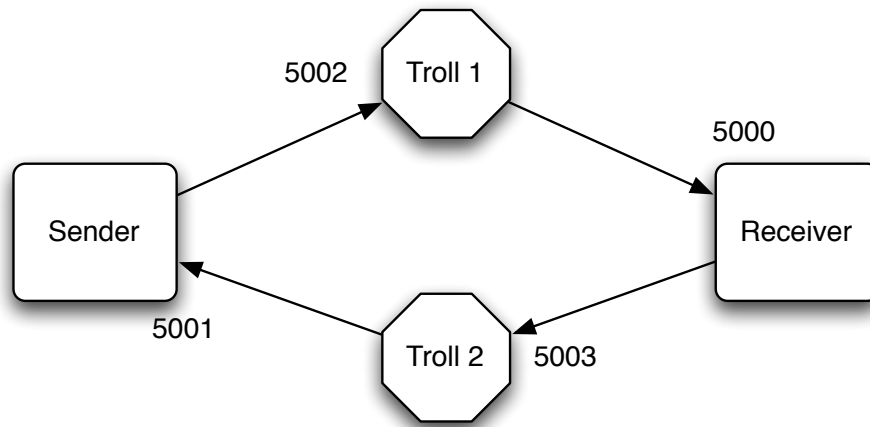


Figure 1: Sender and receiver with two trolls

2. Message Formats

Describe the message formats you will be using for sending the file segments and any other signaling between the sender and the receiver. Describe any message types you will be using and the fields in them.

3. Data Structures

Describe the data structures you will use to buffer segments on the sending and receiving side. Describe some key variables that are keeping track of state (in particular, explain any variables you reference in the state machine). Also list any timers you will be using and what their purpose is.

4. Work Breakdown

Describe who will be responsible for which component of the project. Include items such as design, implementation, integration, debugging, testing, and documentation. Use the initial design document to plan the division of work and the final submitted one to document what was actually the case. (If you are working alone, you should omit this section.)

Evaluation and Strategies

Your project will be evaluated along three criteria. First, we will evaluate your design documentation and code for quality. We will then test your project against a number of “adversaries” who will reorder, delay, drop, duplicate, and corrupt packets in such a way as to make the reliable transfer task difficult. You will receive *correctness* points for transferring a file reliably against each adversary; we will use `cmp` to compare the received file with the original. You will get partial credit based on how many adversaries you can “defeat” by transferring the file reliably.

If you correctly handle *all* adversaries, we will evaluate your *performance*, in terms of transfer time, against a benchmark and assign you a grade. Note that we will perform the test on an incompressible file, so if you want to improve performance, use protocol design rather than compression. There are three things to keep in mind regarding performance:

- If you do not get a perfect correctness grade, you will get *no* credit for performance.
- Performance is the smallest component to your grade.
- It is possible, however, to earn bonus points on performance and this can result in a grade of over 100 on the project.

In particular, you can get up to 80% of the grade by implementing a reliable but inefficient solution, and we recommend that you get that component working *first*, and then work on improving performance as there is time available.

To get full marks (before bonus) on the performance component, you will need to implement at least a sliding window algorithm and a timeout estimation algorithm.

Hand In

Place your source code in a directory called MP3 along with a Makefile that, on execution of the command `make`, causes the executable code for the programs `sender` and `receiver` to be generated by the compiler. Do not include any executable or test files with your submission; we will take off marks if you do.

Note that we have been lenient in the past for submissions with incorrect Makefiles, wrong executable names, or compilation errors. Due to the time constraints for grading MP3, we will be performing the tests using a script, therefore, if your project fails to generate the correct executables, you will get **no credit** for correctness or performance.

You should also include in your submission a file called DESIGN which includes the revised version of your design, along with notes of what has been changed since your original in-class submission and why, and a README file describing your implementation log and citing any sources that you used in solving the assignment.

Grading Guidelines

(20 pts) Initial Design

- This document will be evaluated for completeness, rather than correctness

(40 pts) Correctness

- Correct and complete files sent to the receiver under different adversaries.
- There will be several tests against different adversaries, and we will evaluate each one of them by using “`diff`” to check whether a file was transferred correctly.
- Each individual test will be graded on an all-or-nothing basis, but it *is* possible to get partial credit on this component (e.g. by passing half the tests.)

(20 pts+up to 10 pts bonus) Performance

- You will get **NO CREDIT** on this component if you do not pass all the correctness tests
- Grade will be assigned based on transfer time passing each test. We will perform a weighted average of the test times.
- A sliding window protocol with appropriate RTT estimation will get full credit (20 pts) on this component. You may get up to 10 bonus points for implementing faster transfer

(10 pts) Documentation

- README
- Revised design

(10 pts) Code quality

- Well commented
- Proper indentation
- Sensible variable names

(5 pts bonus) Error correction