CSc 450/550: Computer Communications and Networks (Summer 2007)

Lab Project 2: Reliable Datagram Protocol

Spec Out: June 1, 2007
Design Due: June 20, 2007
Demo Due: July 4, 2007
Code Due: July 6, 2007

1 Introduction

In this project, students will design and implement a reliable datagram protocol (RDP) over UDP to transfer a large binary file from a sender to a receiver, through a shared relay. The project allows students to better understand the common transport-layer protocol mechanisms, such as flow, error and congestion control. As a **design** project, only project requirements are provided, and students have the freedom to create their own design.

- Hint: be creative, but you should be able to justify your own design.

2 Requirements

2.1 Reliable datagram protocol

1. RDP should transfer a file of any size from a sender to a receiver, through a shared relay.
2. The received file should be identical to the sent file in content.
    - Hint: how do you know two files are identical? You can compare them bit-to-bit **for sure**. But for a quick check, you can use `md5sum` to know that two files of the same size are actually different if they do not have the same MD5 checksum.
    - Hint: in order to assist file transfer, you may need to have a simple "application-layer" protocol, or being embedded in RDP, to convey the meta information of the file (e.g., file name and size) and indicate the beginning and end of the file transfer.
3. Maximal RDP packet size (including RDP packet header and data payload): 1024 bytes.
4. Minimal RDP packet header size: 8 bytes (see Section 2.4).
    - Hint: since the maximal RDP packet size is limited, you may want to minimize the size of your RDP packet header, while still achieving the functionality of your design, in order to maximize the size of the data payload in each RDP packet.

5. Packets may be dropped, duplicated, reordered and corrupted by the network and the relay.

   - Hint: you will need to include some error control procedures, including error detection, error notification and error recovery, in your design.

6. Due to performance concerns, RDP cannot use the stop-and-wait strategy.

   - Hint: you will need to include some flow control procedures in your design.

For testing purposes, your RDP should be able to correctly transfer a binary file of minimal size 1 MB through a relay with packet error probability 0.1 in a reasonable amount of time.

- Hint: you do not need to implement a full set of TCP protocol mechanisms over UDP, but you can use TCP to help you formulate your design. Also, you may use HTTP or FTP to help you formulate your "application-layer" protocol.

2.2 RDP Sender

Your RDP sender should have the following command line syntax:

```
./rdps <file_name> <receiver_ip> <receiver_port> <relay_ip> <relay_port>
```

`<file_name>` specifies the location of the file at the sender to be sent to the receiver through the relay; `<receiver_ip>` and `<receiver_port>` specify the location of the receiver; `<relay_ip>` and `<relay_port>` specify the location of the relay.

If a wrong syntax is used when invoking the program, the program should print out error messages showing the proper usage and exit gracefully. If the file cannot be read (e.g., nonexistence, bad file permission, etc), an error message is printed out and the program exits gracefully:

```
rdps: read error with <file_name>, exiting...
```

Otherwise, the following messages are printed out to show the progress of the file transfer:

```
rdps: sending <file_name> of <file_size> bytes to <receiver_ip>:<receiver_port>
      through <relay_ip>:<relay_port>...
rdps: <TO> <HBOA> <FBOP> <LBOP> <HBOS>
rdps: <TO> <HBOA> <FBOP> <LBOP> <HBOS>
...
rdps: <TO> <HBOA> <FBOP> <LBOP> <HBOS>
rdps: sent <HBOA+1> bytes in <TO> seconds at <throughput> Bps
```

`<TO>`: time offset in `second.microsecond` since sending the first data packet; `<HBOA>`: the highest byte offset acknowledged so far (i.e., the receiver has correctly received up to this byte inclusive); `<FBOP>`: the first byte offset in the packet currently being sent; `<LBOP>`: the last byte offset in the packet currently being sent; `<HBOS>`: the highest byte offset sent in all packets so far. Byte offset starts at 0.

2.3 RDP Receiver

Your RDP receiver should have the following command line syntax:

```
./rdpr <receiver_port>
```

The received file will be stored in the current directory, with the same `<file_name>` as that at the sender. If a wrong syntax is used when invoking the program, the program should print out error messages showing the proper usage and exit gracefully. If the file cannot be created, an error message is printed out and the program exits gracefully:

```
rdpr: write error with <file_name>, exiting...
```

Otherwise, the following messages are printed out to show the progress of the file transfer:

```
rdpr: receiving <file_name> of <file_size> bytes from <sender_ip>:<sender_port>
      through <relay_ip>:<relay_port>...
rdpr: <TO> <HBOA> <FBOP> <LBOP> <HBOR>
rdpr: <TO> <HBOA> <FBOP> <LBOP> <HBOR>
...
rdpr: <TO> <HBOA> <FBOP> <LBOP> <HBOR>
rdpr: received <HBOA+1> bytes in <TO> seconds at <throughput> Bps
```

`<TO>`: time offset in `second.microsecond` since receiving the first data packet; `<HBOA>`: the highest byte offset acknowledged so far (i.e., the receiver has correctly received up to this byte inclusive); `<FBOP>`: the first byte offset in the packet currently being received; `<LBOP>`: the last byte offset in the packet currently being received; `<HBOR>`: the highest byte offset received in all packets so far. Byte offset starts at 0.

For testing purposes, you first run your RDP receiver at a given port on one machine and then run your RDP sender accordingly on another machine, or at a different port on the same machine, with the name of the file to be sent. Both your RDP sender and receiver should exit properly after the file is correctly transferred and print out: e.g.,

```
rdps: file transfer successful, exiting...
```

or a failure is declared by your programs with, e.g.,

```
rdps: RDP protocol error, exiting...
```

2.4 RDP Relay

The RDP relay will be provided to you.

1. The RDP relay command line syntax

   ```
   ./relay <relay_port> <packet_error_prob>
   ```

`<packet_error_prob>`, in the range of $[0, 1)$, specifies the probability of a packet being intentionally corrupted by the relay. Be aware that even when `<packet_error_prob>` is 0 at the relay, packets may still be dropped, duplicated, reordered and corrupted by the network.

2. To allow the relay to relay packets from their source to destination, the RDP header should begin with the following three fields in the specified order: RDP magic number (16-bit, `0xabcd`), destination port number (16-bit), destination IP address (32-bit), all in network byte order (i.e., Big Endian). Additional RDP header fields may be defined by you in your design to meet the protocol requirements.

   When the relay receives a packet, it will first check whether the packet has the expected magic number (i.e., `0xabcd`); if not, the packet will be dropped immediately; otherwise, the relay will learn the destination IP address and port number in the RDP header and then send the packet to the destination with error probability `packet_error_prob`.

   Also, since the relay is shared by multiple file transfers, you may have packets from other file transfers delivered to you due to network errors or the errors introduced by the relay.

3. No direct communication between sender and receiver is allowed.

3 Design

The design of your RDP protocol and RDP sender and receiver should be submitted on paper in the lab section for which you have registered on the design due date.

In the initial design, you need to tell lab instructors the RDP packet format and flow and error control procedures that you use to meet the protocol requirements. Lab instructors may discuss with you whether your design could be improved, so you want to put enough detail there. You still can change your design after the design due, and you need to include the final design in the code submission and justify your design and the changes you have made.

If you do not submit your initial design, your final design will not be marked.

4 Demonstration

Your RDP sender and receiver should be demonstrated in the lab section for which you have registered on the demo due date. **Lab projects not demonstrated will have their code submission not marked. It is expected that your lab project be working at the time of demonstration.** During the project demo, lab instructors will go through a demo checklist together with the student, and then provide the checklist to the student. Students will have the chance to improve their design and implementation until the code due date.

The demo is intended to allow students to demonstrate their projects and to help them improve their design and implementation, not to be coding or debugging assistance, and only focuses on required features. There is no guarantee on the correctness and grade of the project, which can only be determined after the code inspection.

5 Submission

The entire lab project, including the code and documentation, should be submitted electrically through `csc450l` (`l` is for letter L) at `http://www.csc.uvic.ca/~submit/index.cgi` on or before

the code due date.

Only the source code (including header files and `Makefile`) and documentation (including `readme.txt` and `design.pdf`) should be included in a single `tar.gz` file to be submitted. No object or binary files are included in the submission. If `directory` is your project directory, to create such a gzipped tarball, you can

```
cd direcoty
tar -zcvf p1.tar.gz .
```

This packing and naming convention should be strictly followed to allow your submission to be properly located for grading.

In `directory`, you need to include a `Makefile`, which compiles and builds the final binary executable (`rdps` and `rdpr`) automatically by typing

```
make
```

The same `Makefile` also removes all object and executable files when you type in

```
make clean
```

All projects will be tested on `linux.csc.uvic.ca`

In `directory`, you need to include `readme.txt` in plain text format, which contains your student number, registered lab section and a brief description of your code structure.

You also need to include `design.pdf` in Portable Document Format, which describes and justifies your final design and the changes you have made since the initial design, if any.

The code itself should be sufficiently self-documented. For more information on acceptable coding style, please see [1].

IMPORTANT: All submitted work should be yours. If you have used anything out there, even a small component in your design and implementation, you should credit and reference properly, and your contribution can be determined accordingly. For academic integrity policies, please see [2].

6 Marking

This lab project is worth 15% in the final grade of this course for CSc 450 students, and 10% for CSc 550 students.

For mark posting and appeal policies, please see the official course outline at [2].

References

[1] `http://www.csc.uvic.ca/~csc450l/references/Code-Style.html`

[2] `http://courses.seng.engr.uvic.ca/courses/2007/summer/csc/450`