CSc 361: Computer Networks (Fall 2014)

Programming Assignment 2 (P2): Reliable Datagram Protocol (RDP)

Spec Out: October 21, 2014
Code Due: November 21, 2014

1 Introduction

So far, you have implemented a Simple Web Server following the Hyper Text Transfer Protocol (HTTP) in the first programming assignment (P1). Good job! But very soon you have found that the simple web server is not always working reliably, due to the fact that it just relies on the unreliable User Datagram Protocol (UDP). For example, if the HTTP request or response messages are larger than what a normal UDP packet can accommodate, multiple UDP packets have to be sent over the network. These packets may get lost, duplicated, reordered or corrupted along the way, and neither the sender nor the receiver has the capability to recover them. In addition, a fast sender can send packets much faster than how much the receiver can handle and may potentially overflow the receiver's buffer and cause packet loss, even if the packets have arrived at the receiver. Further, the sender should be able to indicate to the receiver the end of the request or response messages, so the receiver does not need to wait any further for more packets.

Therefore, you need to add some flow control and error control capabilities, as well as the start and finish indicators, to UDP, leading to a "new" protocol that we call CSc361 "Reliable Datagram Protocol" or RDP. In this programming assignment, you will complete the design of the CSc361 RDP protocol and implement it using the datagram (DGRAM) socket Application Programming Interface (API) with UDP. The goal is to transfer a text file from a sender to a receiver through the Linksys router in ECS360 that emulates various network impairments such as packet loss, duplication, reordering and corruption in a real network.

In this assignment, only the assignment requirements and basic design are provided, and you have the freedom to create your own detailed design and implement it. Be creative, but you should be able to justify your own design choices.

Date	Lab Lecture and Tutorial Tasks	Milestones
Oct 22/23	P2 Spec, RDP header, Connection Mgmt (CM) refreshed	RDP header done
Oct 29/30	Flow Control (FC) and Error Control (EC)	CM done, FC half-way
Nov 5/6	Congestion Control, network impairments and NAT	FC done, EC half-way
Nov 12/13	reading break; no labs	CM, FC and EC all done
Nov 19/20	last-minute help (if time permits)	all done and tested

2 Schedule

In order to help you finish the assignment on time successfully, the schedule of this assignment has been synchronized with both the lectures and the lab and tutorial modules. There are at least four tutorial modules arranged in the weekly lab sessions during the course of this assignment (five weeks), and it is very important for you to follow the schedule closely for your assignment.

3 Requirements

3.1 Reliable Datagram Protocol (RDP)

1. RDP should be able to transfer a file of any size from a sender to a receiver, through a router.
2. The received file should be identical to the sent file in content.
   - Hint: how do you know two files are identical in content? 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.
3. Maximal RDP packet size (including RDP packet header and data payload): 1,024 bytes.
4. RDP packet header has a similar design as HTTP request or response header fields in ASCII string coding. The minimal set of the supported RDP packet header fields are:

RDP header field	Possible value	Meaning
Magic: _magic_	UVicCSc361	CSc361 RDP protocol
Type: _type_	DAT	Data packet
	ACK	Acknowledgment packet
	SYN	Synchronization packet
	FIN	Finish packet
	RST	Reset packet
Sequence: _seqno_	e.g., 0	byte sequence number
Acknowledgment: _ackno_	e.g., 900	byte acknowledgment number
Payload: _length_	e.g., 900	RDP payload length in bytes
Window: _size_	e.g., 10240	RDP window size in bytes
(an empty line)		the end of the RDP header

- `Magic: UVicCSc361`: indicating the RDP protocol used in the CSc 361 Lab at UVic.
- `Type: _type_`: identifying the type of the packet, i.e.,
  - `DAT`: Data packet, sent by the sender and carries data payload (i.e., `Payload` is greater than zero for the same packet).
  - `ACK`: Acknowledgment packet, sent by the receiver to acknowledge the reception of DAT, SYN or FIN packets.
  - `SYN`: Synchronization packet, sent by the sender to initiate the data transfer.
  - `FIN`: Finish packet, sent by the sender to finish the data transfer.

    - `RST`: Reset packet, sent by either the sender or the receiver to terminate the current data transfer due to unrecoverable errors.
    - You may be able to define more types of packets in your extended design.
- `Sequence` or `Acknowledgment`: integer, byte sequence or acknowledgment number (i.e., `x+1` represents the byte immediately after the byte represented by `x` in the data stream). Initial sequence number is chosen by the sender randomly.
    - For packets such as DAT, SYN and FIN sent by the sender, it represents the sequence number of the first byte of the data payload, the initial sequence number, and the finish sequence number, respectively.
    - For packets such as ACK sent by the receiver, it represents the sequence number of the next byte expected at the receiver, i.e., acknowledgment number.
- `Payload` or `Window`: integer, byte payload length or window size (i.e., `x` represents x data bytes).
    - For packets such as DAT, it indicates the length of the data payload in the packet.
    - For packets such as ACK, it represents the window size (i.e., the extra amount of data can be accommodated by the receiver).
- You may be able to define more packet header fields in your extended design.

5. Packets may be dropped, duplicated, reordered and corrupted by the network and the router.
    - Hint: you will need error control procedures, including error detection, error notification and error recovery, in your design.
6. For performance concerns, RDP cannot use the stop-and-wait strategy.
    - Hint: you will need some flow control procedures in your design.
    - Hint: you do not need to implement a full set of TCP protocol mechanisms over UDP, but you can use TCP to help formulate your design. Note that unlike TCP with bidirectional data flows, there is only one single data flow from the sender to receiver.

For self-testing purposes, your RDP implementation should correctly transfer a file of minimal size 1 MB through the Linksys router in the lab, and the packet error rate can go as high as 10% (BTW, normally TCP suffers greatly or even fails with a 10% packet error rate).

3.2 RDP Sender

Your RDP sender should have the following command line syntax:

```
rdps sender_ip sender_port receiver_ip receiver_port sender_file_name
```

`sender_ip` and `sender_port` specify the IP address and UDP port number the sender will use; `receiver_ip` and `receiver_port` specify the location of the receiver; `sender_file_name` specifies the location of the file at the sender to be sent to the receiver through the router.

While transferring the file, your RDP sender should print to screen the log message on sending/receiving each packet, i.e., not just data packets, with the following format:

```
HH:MM:SS.us event_type sip:spt dip:dpt packet_type seqno/ackno length/window
```

- `HH:MM:SS.us`: represents the time instance in `Hour:Minute:Second.microsecond` format sending or receiving a packet, e.g., `23:59:59.999999` means 1 microsecond before midnight.
- `event_type`: represents the type of events, i.e.,
  - `s`: send a packet for the first time
  - `S`: resend the packet
  - `r`: receive a packet
  - `R`: receive the same packet again
- `sip:spt dip:dpt`: are source/destination IP address and UDP port number of the packet.
- `packet_type`: represents the type of the packet, i.e, DAT, ACK, SYN, FIN, or RST.
- `seqno/ackno`: sequence number or acknowledgment number.
- `length/window`: payload length or window size.

After the file transfer is finished, your RDP sender should print to screen the summary message following the example below

```
total data bytes sent: 1165152
unique data bytes sent: 1048576
total data packets sent: 1166
unique data packets sent: 1049
SYN packets sent: 1
FIN packets sent: 1
RST packets sent: 0
ACK packets received: 1051
RST packets received: 0
total time duration (second): 0.093
```

In this example, a file of 1 MB (1,048,576 bytes) is successfully transferred, which corresponds to 1,049 unique data packets (1,048 packets of 1,000 bytes each in payload, and one packet of 576 bytes). Due to packet errors, in total 1,165,152 data bytes are sent (1,164 packets of 1,000 bytes each in payload, and two packets of 576 bytes each), for a total time duration of 0.093 second from the ACK packet on the first SYN packet to the ACK packet on the last FIN packet.

3.3 RDP Receiver

Your RDP receiver should have the following command line syntax:

```
rdpr receiver_ip receiver_port receiver_file_name
```

`receiver_file_name` specifies the location at the receiver to store the received file.

While receiving the file, your RDP receiver should print to screen the log message on <u>receiving/sending each packet, i.e., not only data packets,</u> with the following format:

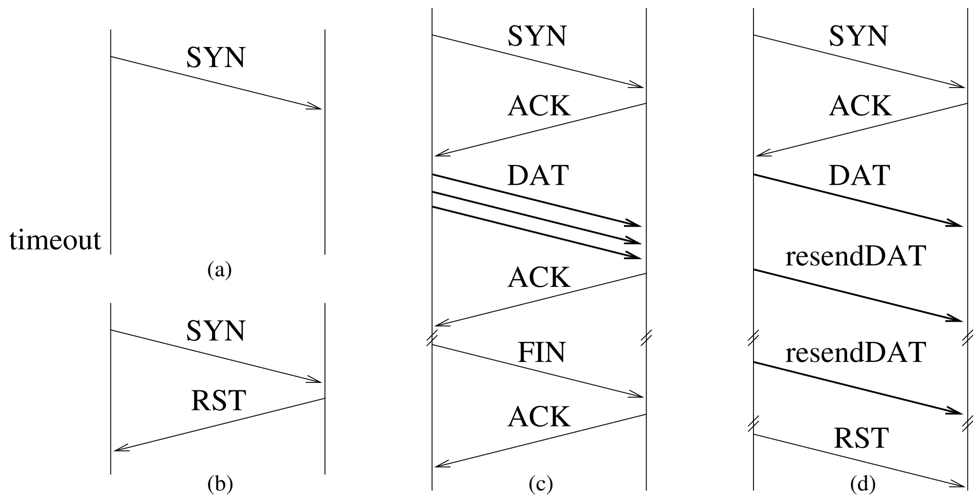


Figure 1: RDP protocol interaction examples.

```
HH:MM:SS.us event_type sip:spt dip:dpt packet_type seqno/ackno length/window
```

After the file transfer is finished, your RDP receiver should print to screen the summary message with the following format

```
total data bytes received:
unique data bytes received:
total data packets received:
unique data packets received:
SYN packets received:
FIN packets received:
RST packets received:
ACK packets sent:
RST packets sent:
total time duration (second):
```

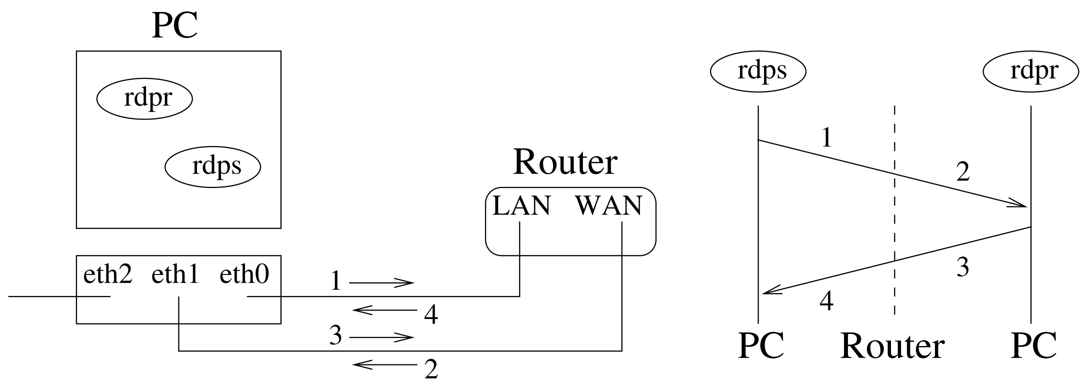
Figure 1 shows some RDP protocol interaction examples. They are not exhaustive.

For self-testing purposes, you first run your RDP receiver on the "WAN" interface (10.10.1.100) and port 8080 of your lab computer,

```
./rdpr 10.10.1.100 8080 received.dat
```

and then run your RDP sender on the "LAN" interface (192.168.1.100) and port 8080 of your lab computer to send the `sent.dat` file.

```
./rdps 192.168.1.100 8080 10.10.1.100 8080 sent.dat
```



1. 192.168.1.100:8080 –> 10.10.1.100:8080
2. 10.10.1.1:9876 –> 10.10.1.100:8080
3. 10.10.1.100:8080 –> 10.10.1.1:9876
4. 10.10.1.100:8080 –> 192.168.1.100:8080

Figure 2: Network configuration in the lab.

4 Network Emulation

Figure 2 shows the network configuration in the lab for this assignment, where your `rdps` and `rdpr` are running on the PC and using `eth0` and `eth1`, respectively. `eth0` and `eth1` are connected to the LAN and WAN port of the Linksys WRT54GL router, respectively.

Figure 2 also shows the sample packets appearing on the `eth0` and `eth1` interface and sent/received by `rdps` and `rdpr`, respectively, for the SYN-ACK interaction. You may notice that the source IP address and port number have been changed from packet 1 to packet 2 when it travels through the router. This is due to the Network Address Translation (NAT). However, the NAT process should be transparent to your RDP sender and receiver, and your `rdpr` will extract the translated IP address and port number for your `rdps` from the incoming packets and respond accordingly.

The Linksys WRT54GL router in the lab has been specially instrumented to emulate a wide-area network with packet delay, loss, duplication, corruption and reordering. You will be given detailed instruction to set these network impairments in lab. For reference, see [1].

In order for your packets to go through the router, the computer in the lab has been specially instrumented to forward packets with the "LAN" interface source IP address (192.168.1.100) and the "WAN" interface destination IP address (10.10.1.100) through the router, when you bind the socket of your `rdps` to source IP address (192.168.1.100).

5 Submission

The entire programming assignment, including the code and documentation, should be submitted electrically through `http://connex.csc.uvic.ca` on or before the due date. The site will start to accept submissions one week before the due date.

Only the source code (including header files and Makefile) and documentation (including Readme) 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 directory
```

```
tar -zcvf p2.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 binary files when you type in

```
make clean
```

All assignments will be tested on `n-`*greek*`.csc.uvic.ca`

In `directory`, you also need to include a `Readme` plain text file, which contains your student number, registered lab section and a brief description of your design in RDP flow and error control and code structure, as well as the approved bonus features, if any.

The code itself should be sufficiently self-documented, which will not only help you build and maintain your code well but also assist the code demo and evaluation.

IMPORTANT: All submitted work should be yours. If you have used anything out there, even a small component in your 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 25% in the final grade of this course. Demo will be arranged after the submission, where the marker will ask questions on the code design and implementation. The lab instructor will also check the progress of your assignment during the weekly lab sessions.

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

References

[1] `http://www.linuxfoundation.org/collaborate/workgroups/networking/netem`

[2] `http://courses.seng.engr.uvic.ca/courses/2014/fall/csc/361`