CSc361 Fall 2020 Programming Assignment 2 (P2): Reliable Datagram Protocol (RDP) Specification
Spec Out: by Wednesday, October 7, 2020
Code Due: by Friday, October 30, 2020, 5pm through connex.csc.uvic.ca

Objective: In this programming assignment, you will use the DATAGRAM socket (i.e., supported by UDP) in Python to create a Reliable Datagram Protocol (RDP) transceiver (rdp.py, a combined sender and receiver) running on H1 in PicoNet, to interact with a given echo server running on H2, over an unreliable link at R.

Schedule: There are four tutorials (T5, T6, T7 and T8) and three lab sessions (L5, L6 and L7) associated with this assignment.

In T5 on October 9, the tutorial instructor will go through the P2 specification and answer possible questions, so please read P2 spec carefully beforehand.

In L5 on October 13 or 14, the lab instructor will go through TCP connection management and flow control packet capture and analysis, and help students form their design for P2.

In T6 on October 16, the tutorial instructor will check students P2 design on RDP connection management and flow control, and provide feedback and instruction on how to test P2 in PicoNet with network impairments.

In L6 on October 20 or 21, the lab instructor will go through TCP error control, check students P2 implementation on connection management and flow control, capture and analyze RDP packets, and provide help if needed.

In T7 on October 23, the tutorial instructor will check students P2 design on RDP error control, and provide feedback and instruction on submission.

In L7 on October 27 or 28, the lab instructor will go through TCP congestion control, and check students P2 implementation on RDP error control, capture and analyze RDP packets, and provide help if needed.

In T8 on October 30, the tutorial instructor will provide some last-minute reminders and help.

Please follow our tutorial and lab schedule very closely for this assignment, which ensures its success and smoothness.

Requirements: A basic RDP design is provided, and you can extend the design as you see fit. RDP follows HTTP design, using full text, line-by-line control headers to establish (SYN) and release (FIN) connection, and Sequence number in data (DAT) packet of PAYLOAD Length and Acknowledgment number in acknowledgment (ACK) packet to reorder out-of-order packets, acknowledge received packets and identify lost packets for retransmission. To be efficient, RDP cannot use stop-and-wait (i.e., transmit a packet, wait for acknowledgment, and retransmit until it is received), and has to support flow control using Window size (i.e., multiple packets can be transmitted back-to-back within the Window, and lost packets will be recovered through retransmission by timeout or three duplicate acknowledgments). You can model your RDP design after TCP, but you do not need to implement the entire TCP. Congestion control is not needed for RDP in P2.

RDP packet format

COMMAND
Header: Value
…
Header: Value

PAYLOAD

where, COMMAND := SYN|FIN|DAT|ACK|RST, and SYN indicates to establish connection, FIN to close connection, RST to reset connection, DATA to send data, and ACK to acknowledge received data.

Header := Sequence|Acknowledgment|Window|Length, and Sequence provides the sequence number associated with the packet (the first byte of the PAYLOAD if any), Acknowledgment the sequence number expected by the receiver (the next byte of the PAYLOAD if any), Window the size of the receiver's window in bytes, and Length the length of the PAYLOAD in bytes. Note that an empty line indicates the end of headers.

PAYLOAD, if existing, is the actual data sent from the sender to the receiver. PAYLOAD is at most 1024 bytes.

The Echo Server

On H2

mkfifo fifo

which makes a FIFO (named pipe) called fifo. you only need to run this command once at /home/jovyan

Then

cat fifo | nc -u -l 8888 > fifo

which runs nc (netcat) in UDP server mode at port 8888, redirects output to fifo, and pipes through fifo to nc.

To test the echo server, on H1

nc -u h2 8888

and type any input---it will be echoed back by H2 and shown on H1. Use tcpdump on R

tcpdump -n -l -i r-eth1 udp port 8888

to verify it.

Connection Management

The RDP sender sends SYN first to establish the connection with the RDP receiver, and the RDP receiver sends ACK to accept the connection. Note that RDP is unidirectional from the sender to receiver, so the receiver does not need to send SYN. After the data transfer finished, the RDP sender sends FIN to close the connection, which is acknowledged by the RDP receiver with ACK. Please note that COMMAND, Headers and

PAYLOAD are all case sensitive. For unrecognized or incompatible COMMAND and Headers (e.g., DAT packets missing Sequence number and Length, ACK packets missing Acknowledgment number and Window, and so on), RDP will reset the connection with RST, and there is no ACK for RST.

Data Transfer

Once connected, the RDP sender can send DAT (data) packets to the receiver, each with a Sequence number header indicating the sequence number of the first bytes of the PAYLOAD, and Length indicating the size of the PAYLOAD in bytes. The RDP receiver will acknowledge the received DAT packets cumulatively, using the Acknowledgment header indicating the first byte expected in the next PAYLOAD.

How to emulate Internet delay

On R,

tc qdisc add dev r-eth1 root netem delay 100ms

will add 100 millisecond delay at the output queue of r-eth1.

Flow Control

The RDP receiver will advertise its receiver window in ACK packets with the Window header. The RDP sender shall respect the Window size and shall not send any data with sequence number equal to or above Acknowledgment+Windows.

How to emulate Internet delay and loss

On R

tc qdisc change dev r-eth1 root netem delay 100ms loss 25%

after the above command will add 100 millisecond delay and set 25% packet loss at the output queue of r-eth1.

Error Control

Once packet loss occurs, the RDP sender and receiver may miss SYN, FIN, DAT or ACK packets, so error control is needed. The RDP receiver shall acknowledge received packet cumulatively, and the sender will retransmit unacknowledged packets when timeout or after receiving enough duplicate acknowledgments.

How to run RDP

python3 rdp.py ip_address port_number read_file_name write_file_name

on H1 will bind rdp to ip_address and port_number to send or receive UDP packets, and to transfer a file with read_file_name from the RDP sender to receiver saved with write_file_name. After the file transfer is finished,

diff read_file_name write_file_name

on H1 can tell you whether the received file is different from the sent file of length greater than 10,240 bytes.

What RDP outputs

In addition to saving the received file for diff, RDP shall also output a log to the screen in the following format

DATE: EVENT; COMMAND; Sequence|Acknowledgment: Value; Length|Window: Value

where DATE is timestamp and EVENT := Send|Receive. For example,

Fri Oct 2 16:54:09 PDT 2020: Send; SYN; Sequence: 0; Length: 0
Fri Oct 2 16:54:09 PDT 2020: Receive; SYN; Sequence: 0; Length: 0
Fri Oct 2 16:54:09 PDT 2020: Send; ACK; Acknowledgment: 1; Window: 2048
Fri Oct 2 16:54:09 PDT 2020: Receive; ACK; Acknowledgment: 1; Window: 2048
Fri Oct 2 16:54:09 PDT 2020: Send; DAT; Sequence: 1; Length: 1024
Fri Oct 2 16:54:10 PDT 2020: Receive; DAT; Sequence: 1; Length: 1024
Fri Oct 2 16:54:10 PDT 2020: Send; ACK; Acknowledgment: 1025; Window: 1024
Fri Oct 2 16:54:10 PDT 2020: Receive; ACK; Acknowledgment: 1025; Window: 1024
Fri Oct 2 16:54:10 PDT 2020: Send; FIN; Sequence: 1025; Length: 0
Fri Oct 2 16:54:10 PDT 2020: Receive; FIN; Sequence: 1025; Length: 0
Fri Oct 2 16:54:10 PDT 2020: Send; ACK; Acknowledgment: 1026; Window: 2048
Fri Oct 2 16:54:10 PDT 2020: Receive; ACK; Acknowledgment: 1026; Window: 2048

where the RDP sender sends a SYN packet with Sequence number 0, which is received by the RDP receiver, and the RDP receiver sends ACK to accept with Acknowledgment number 1 and Window size 2048 bytes. Next, the RDP sender sends a DAT packet with Sequence number 1 and payload Length 1024 bytes, which is received by the RDP receiver and acknowledged with an ACK packet with Acknowledgment number 1025 and Windows size 1024 bytes. The sender then closes the connection by sending a FIN packet with Sequence number 1025, which is acknowledged by the receiver with Acknowledgment number 1026, so the connection is closed. Please note that this example just illustrates a very simple case of transferring 1024 bytes reliably.

How to test and evaluate RDP

Run the echo server on H2, run tcpdump on R to capture the interaction between H1 and H2, and run rdp.py on H1. It is very important to correlate the packet exchange between H1 and H2 and the endpoint reaction at H1 for both the RDP sender and receiver through the RDP log, which can help you debug your code as well. Although RDP sender and receiver are in the same rdp.py on H1, RDP protocol and logic cannot be bypassed.

What to submit: rdp.py source file, and the tcpdump files on R, showing the interaction between H1 and H2. Please also copy&paste the content of rdp.py and tcpdump -r the capture file to the text input box on connex.

When: by Friday, October 30, 2020, 5pm through connex -> assignments -> p2

Questions and answers (Please read P2 Spec carefully first): connex -> forums -> p2

Academic integrity: This is an individual assignment so your submitted work shall be done by yourself alone.