CSc361 Fall 2021 Programming Assignment 3 (P3): SWS on RDP (SoR) Specification
Spec Out: by Monday, November 1, 2021
Code Due: by Friday, November 26, 2021, 5pm through bright.uvic.ca

Objective: So far, you have implemented a Simple Web Server (SWS) following the HTTP protocol and running on H2 in PicoNet for your P1, and a Reliable Datagram Protocol (RDP) transceiver running on H1 for your P2. Try to put them together? Yes, that's what P3 will achieve, i.e., your SWS is transported by your RDP!

Schedule: There are two tutorials (T9 and TA) and two lab sessions (L9 and LA) associated with this P3.

In T9 on November 5, 9:30am in ECS116, the tutorial instructor will go through the P3 specification and answer possible questions, so please read the P3 spec carefully in advance. **M2 at 1:30pm in DSB C103**.

In L9 on November 15, 16 or 17, the lab instructor will go through IP routing packet capture and analysis, and check student's SoR design and implementation. A list of SoR interactions is provided at the end of this spec.

In TA (the last tutorial session) on November 19, the tutorial instructor will provide feedback on student's SoR design and implementation, instruction on submission, and guideline on marking. A demo is planned too.

In LA (the last lab session) on November 22, 23 or 24, the lab instructor will go through ARP packet capture and analysis, and check student's SoR implementation and provide some last-week reminders and help.

Please follow our tutorial and lab schedule very closely for this assignment, which ensures its success and smoothness, as shown and/or experienced in P1 and P2. It is impossible to start or finish just in the last week.

Requirements: Unlike in P1 where your SWS runs on TCP socket, in P3, your SoR runs on your RDP, which runs on UDP socket from P2. You can leverage your P1 and P2, and you can make improvements over them.

SoR packet format

RDP-COMMAND(S)
RDP-Header: Value
…
RDP-Header: Value

HTTP-COMMAND
HTTP-Header: Value
…
HTTP-Header: Value

HTTP-PAYLOAD

where, RDP-* follows the RDP protocol defined in P2 and can contain multiple |'ed commands, and HTTP-* follows the HTTP protocol used in P1. Collectively, HTTP-* is known as RDP-PAYLOAD in the context of P3.

SoR protocol

In the following, the text between the quotation marks (i.e., “...”) is the SoR packet in RDP and HTTP format. *sws.py* and *numbers* are for illustration purposes, and actual values in your SoR packets shall reflect reality.

From H1 to H2 to establish the RDP connection and send the HTTP request: “SYN|DAT|ACK
Sequence: *0*
Length: *48*
Acknowledgment: *-1*
Window: *4096*

GET /*sws.py* HTTP/1.0
Connection: keep-alive
”

Please be aware that there is an empty line before “GET /sws.py HTTP/1.0” to indicate the end of the RDP headers, there is an ending “\r\n” (two bytes) for the HTTP command (so for each HTTP header), and there is an empty line of “\r\n” only after “Connection: keep-alive” to indicate the end of the HTTP headers, so the total RDP-PAYLOAD is 48 bytes in this example, where the SYN command also consumes sequence number 0 for reliable delivery. Please note that SoR is bidirectional with data flows between the SoR client on H1 and server on H2 for request-response applications, but unlike the TCP socket, SoR has the flexibility to establish the RDP connection and send the HTTP request in one packet. If the RDP-PAYLOAD is longer than what SoR can accommodate, **RST** is triggered to reset the connection, and the user can rerun the SoR client on H1 with a smaller RDP-PAYLOAD size. The SoR server on H2 shall not be affected and can serve many other clients.

From H2 to H1 to establish the RDP connection and send back the HTTP response: “ACK|SYN|DAT
Sequence: *0*
Length: *1024*
Acknowledgment: *49*
Windows: *4096*

HTTP/1.0 200 OK
Connection: keep-alive

the first 981 bytes of sws.py”

Same as in P2, RDP-PAYLOAD, which includes HTTP-COMMAND, HTTP-Headers and HTTP-PAYLOAD in P3, is at most 1024 bytes, so if sws.py is much longer than 1024 bytes, sws.py will be segmented into multiple HTTP-PAYLOAD segments encapsulated in many RDP packets, regulated by RDP flow and error control.

For example, from H2 to H1 to send back the next segment of the HTTP-PAYLOAD: “ACK|DAT
Sequence: *1025*
Length: *1024*
Acknowledgment: *49*
Windows: *4096*

the next 1024 bytes of sws.py” and so on.

Either SoR client or server can first close the RDP connection by sending an RDP packet containing the FIN command, and the connection is fully closed after the other party's FIN command is acknowledged, so in an ideal case for a small enough request and response, *SoR can finish the transaction in **1** RTT and **3** packets*.

How to emulate Internet delay and loss

Similar to P2 but for r-eth0 instead as the HTTP response data packets are going from H2 to H1, on R

tc qdisc add dev r-eth0 root netem delay 100ms loss 25%

will add 100 millisecond delay and set 25% packet loss at the output queue of r-eth0. You can run tcpdump on r-eth1 to observe the data packets from H2 to H1 before possible loss, and r-eth0 after the loss.

How to run SoR server

python3 sor-server.py server_ip_address server_udp_port_number server_buffer_size server_payload_length

on H2, where the SoR server binds to server_ip_address and server_udp_port for RDP, with server_buffer_size for each RDP connection to receive data, and server_payload_length to send data.

How to run SoR client

*python3 sor-client.py server_ip_address server_udp_port_number client_buffer_size client_payload_length read_file_name write_file_name [read_file_name write_file_name]**

on H1, where the SoR client connects to the SoR server at server_ip_address and server_udp_port, with client_buffer_size for each RDP connection to receive data, and client_payload_length to send data, and requests the file with read_file_name from the SoR server on H2 and writes to write_file_name on H1. "diff read_file_name write_file_name" can help you verify whether the file is transferred correctly. If there are multiple pairs of read_file_name and write_file_name in the command line, it indicates that the SoR client shall request these files from the SoR server in a *persistent* HTTP session over an RDP connection.

What SoR server outputs

SoR server outputs in the same way as SWS in P1 on H2.

What SoR client outputs

SoR client outputs in the same way as RDP in P2 on H1.

How to test and evaluate SoR

Run the SoR server on H2, run tcpdump on R to capture the interaction between H1 and H2, and run the SoR client on H1. Again, it is very important to correlate the packet exchange between H1 and H2 and the endpoint reaction at H1 and H2 for the SoR client and server through the RDP and SWS log, which can help you debug.

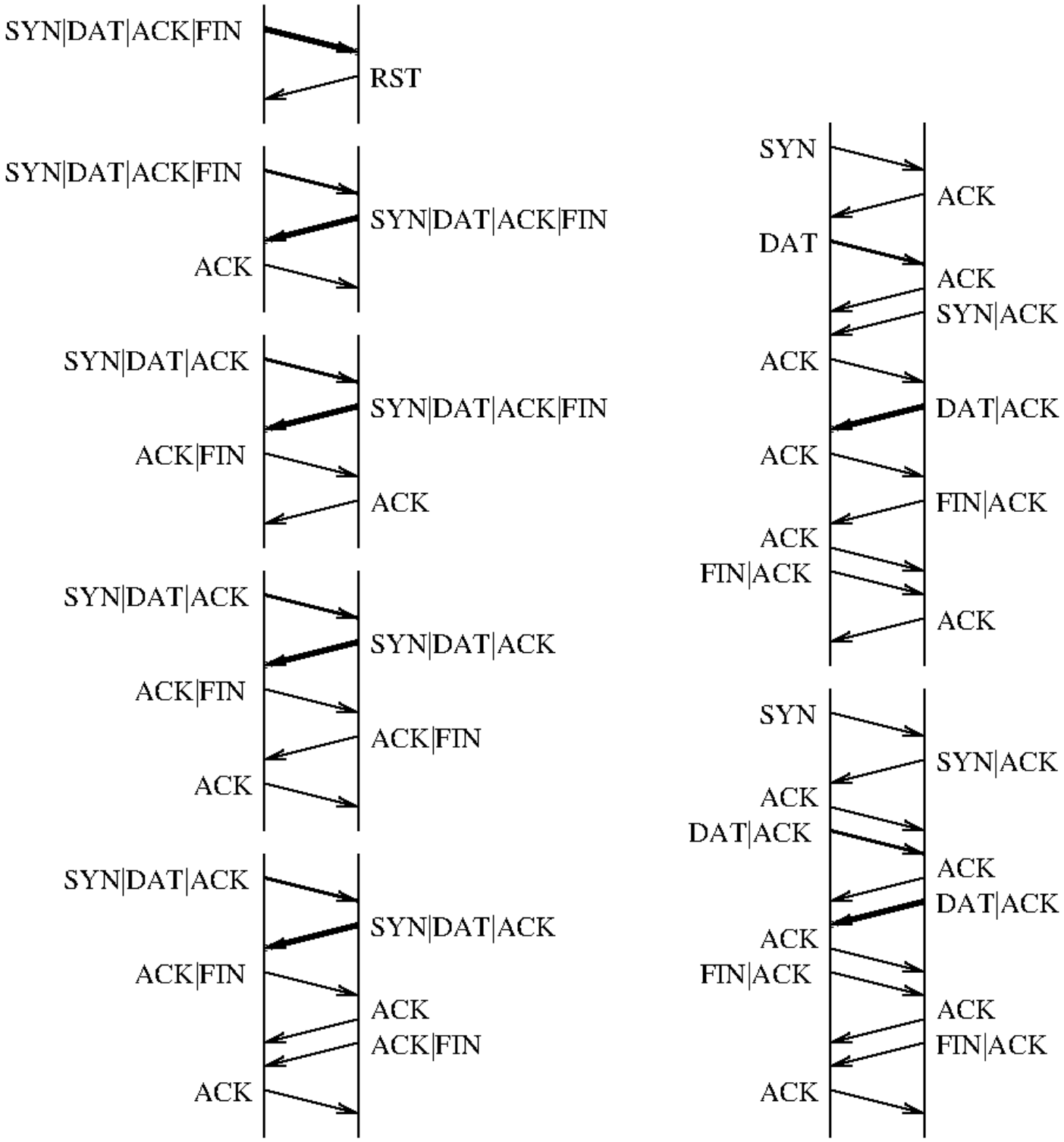
What to submit: sor-server.py and sor-client.py source file, and the tcpdump file on R, showing the interaction between sor-client.py on H1 and sor-server.py on H2 with connection management and flow and error control.

When: by Friday, November 26, 2021, 5pm through bright.uvic.ca => assignments => p3

Questions, answers and discussion (Please read P3 Spec carefully first): bright.uvic.ca => forums => p3

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

Appendix: The possible interactions of SoR client and server, why you need a protocol state machine (PSM)



and what is paraphrased as **Postel**'s law: "Be **liberal** in what you accept, and **conservative** in what you send."