Lab 5

In this lab, we will be implementing RDT 3.0 over UDP using Python.

Part 1: UDP Server/Client in Python

In last week's lab, we used Python to implement a server/client pair using TCP. Implementing those things using UDP is even easier since UDP is connectionless. It is possible with only minor modifications to alter your code from last week to use UDP instead of TCP. If you're interested in a challenge, we encourage you to research and experiment with altering your code from last week to use UDP. If you can manage this on your own, you can skip to part 2 below.

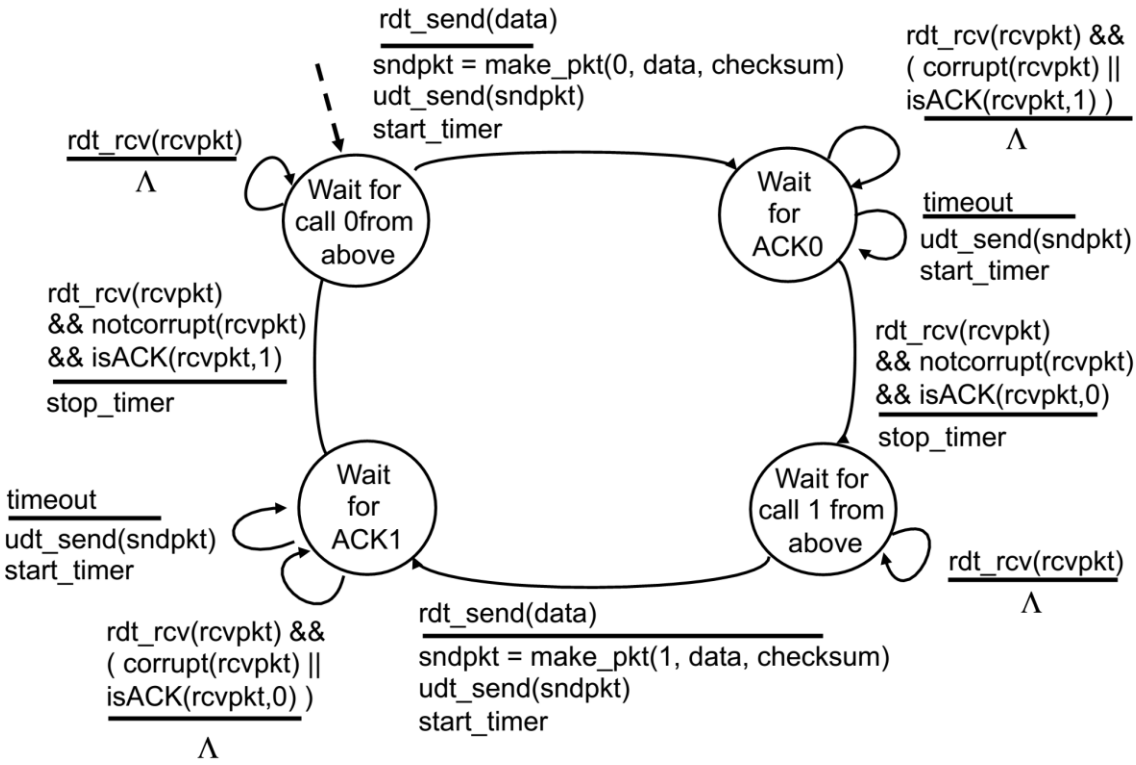
If you'd rather, though, here's a walkthrough that will take you through the process of creating a UDP client/server pair from scratch: http://www.binarytides.com/programming-udp-sockets-in-python/. There are two things to note: first, this walkthrough is best followed using your mininet VM. Second, wherever the walkthrough says to use `ncat` to test your server, you can use `netcat` instead since it is already installed on your mininet VM. Once you have successfully implemented your UDP client and server, you are ready for the next part of the lab.

Part 2: RDT 3.0

Recall that RDT 3.0 is a communications protocol meant to support sender/receiver interactions across unreliable network links. It is distinguished from the other versions of RDT primarily by its use of a timeout mechanism to compensate for the loss of entire packets. This provides for robust communications despite the fact that the underlying implementation is connectionless.

Your task for this part of the lab will be to alter your current UDP client/server code to implement RDT 3.0. If you'd like to review, the lecture material on RDT begins on page 21 of the slides for Transport Layer which can be found in the Course Material section of iLearn. The RDT 3.0 material begins on page 38. You will almost certainly find it helpful to reference the FSM's in the slides. The FSM for the sender is below.

rdt3.0 sender



Hints:

1) In order to handle timeouts, you can create a thread and set a timer value for it for every packet that you send (maybe look at '*threading'* in Python).

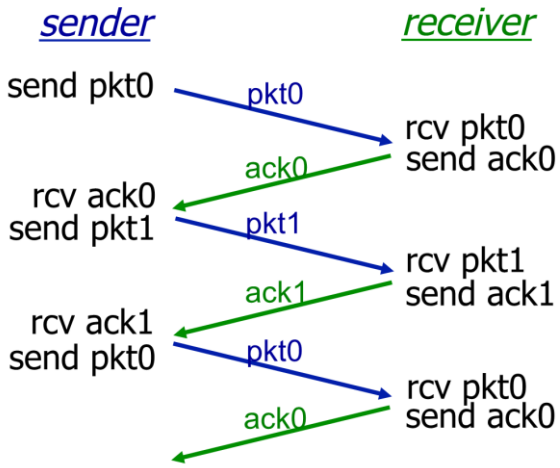
2) For calculating the checksum, you can use the code available in the check.py file (found on iLearn) or any other available libraries. Copy check.py to your files' directory and use it like so:

```
#add this line at the beginning
from check import ip_checksum
#use it in this way
d = ip_checksum( data)
```

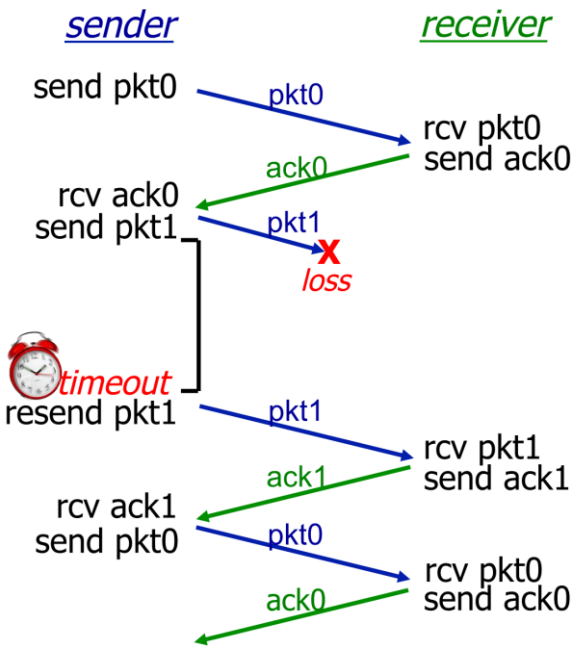
Demo:

You should show the packets transferred between sender and receiver to your TA to validate the implementation of your RDT protocol. You can print the packets' info in receiver and sender or you can use wireshark if you prefer. The three scenarios below are arranged in order from easiest to hardest so it's best to work on them one at a time in order. Move on to the next one only when you've completed the previous ones.

1) **Scenario 1:** Send three packets from sender to receiver and show the exchanged packets and acknowledgements.



2) **Scenario 2:** Send a packet with an invalid checksum from sender to receiver and show how this problem is handled by sender and receiver.



(Instead of complete loss, we have a corrupted packet)

3) **Scenario 3:** Send three packets from sender to receiver, but the receiver should delay more than the timeout period before sending the ack of the first packet.