Implement a Reliable Transport Protocol

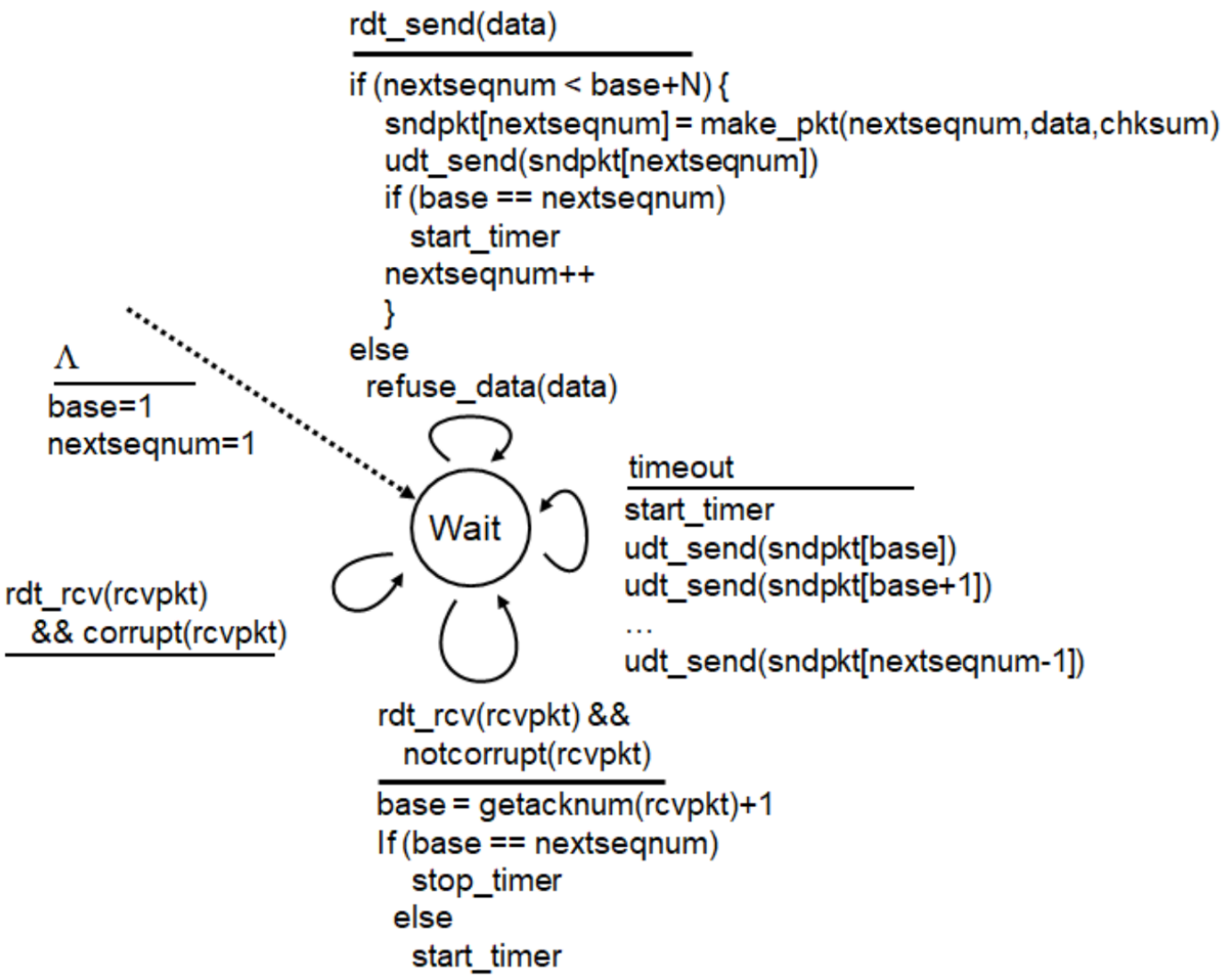
Submit Assignment

Due Saturday by 11:59pm **Points** 75 **Submitting** a file upload

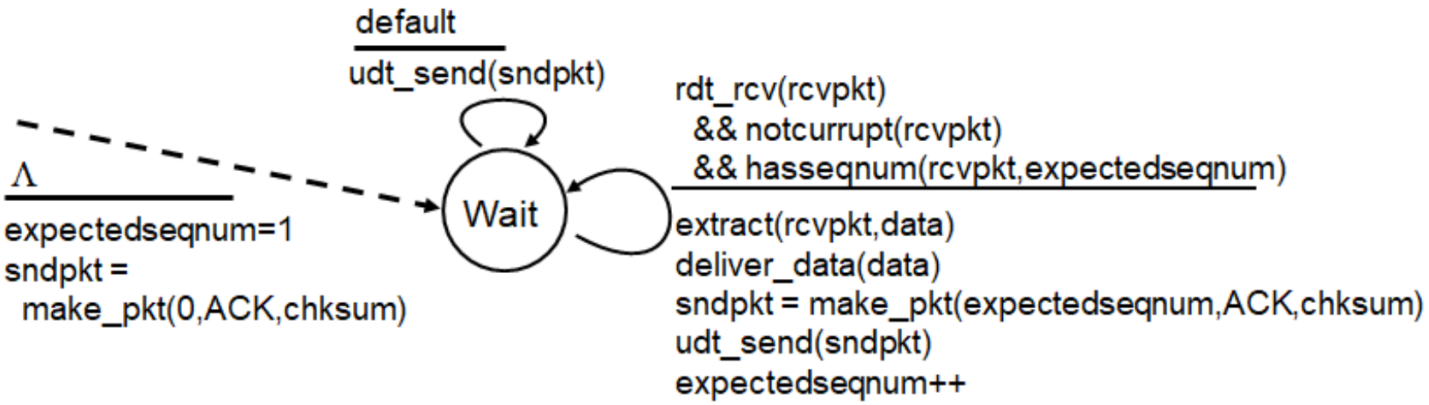
In this project, you will implement the Go-Back-N reliable transfer protocol, as discussed in class. Specifically you need to implement the sender and receiver FSM shown below. As a reminder:

- rdt_send(data) corresponds to an application calling socket.send() using TCP
- rdt_rcv(rcvpkt) corresponds to the network layer providing TCP (which runs at the transport layer) with data it has received from over the network
- udt_send(pkt) corresponds to TCP passing a packet to the network layer for transmission

GBN Sender



GBN Receiver



Rather than running over a real network, we'll run your code over a simulator. This will allow us to precisely control network latency, packets loss rates, and packet corruption rates. You will implement a full-duplex GBN system, capable of both sending and receiving packets.

I've provided you with three python files, and a folder with 12 test cases. You can find these files in Files\Programming Assignments\GBN Protocol [note:they have not yet been uploaded]

- **GBNHost.py**
  - You are responsible for completing three functions in this file:
    - **receive_from_application_layer(payload):** this function is responsible for handling rdt_send() events for the GBN Sender FSM.
      - There is a key difference in how this function needs to be implemented, as compared to the FSM: the FSM assumes that data from the application layer is refused if it cannot be immediately sent. In your implementation, I want you to buffer received data and then send it when GBN is able to do so.
    - **receive_from_network_layer(byte_data):** this function is responsible handling rdt_rcv() events for BOTH the GBN Sender and Receiver FSM. This will require implementing portions of both FSM, as they each contain rdt_rcv events.
    - **timer_interrupt():** this function is responsible for handling timeout events for the GBN Sender FSM.
- **Simulator.py**
  - You should not change any of this code (you can add debugging statements if you think that would be helpful). We will test your program using a clean copy of this file, so no changes you make will persist.
  - Simulator.py implements four functions you will need to call from your code to access the simulated hardware. You can access these methods from your SlidingWindowHost class using the syntax: self.simulator.<method_name>(<parameters>)
    - **start_timer(entity,duration)** starts a simulated hardware timer that will trigger a timeout event when it expires
      - Entity should be either EventEntity.A or EventEntity.B, depending on who is starting the timer. The RDTTester creates two instances of SlidingWindowHost for each test, where one instance

is assigned to EventEntity.A and the other to EventEntity.B. You can access which entity a specific instance of SlidingWindowHost is assigned to by calling *self.entity*
    - Duration is a float value indicating the amount of time that will pass before the timer interrupts. The duration of the timer is different for each test case, and is stored in *self.timer_interval* in SlidingWindowHost.py
  - **stop_timer(entity)** terminates a running timer without triggering a timeout event
    - As with start_timer, entity is either EventEntity.A or EventEntity.B, and the specific value for a given instance of SlidingWindowHost can be found in *self.entity*.
  - **to_layer3(entity, packet, is_ACK)** passes the packet to the simulated network layer for forwarding to the destination entity. This corresponds to the udt_send() function in the GBN sender and receiver FSMs.
    - Entity is the same here as specified with start_timer
    - Packet should contain a packed byte array, as specified below
    - is_ACK should be True when sending an ACK. This exists to help the test cases and would not be needed in a production system
  - **to_layer5(entity, datasent)** allows GBN to pass processed data to a simulated application. This corresponds to the deliver_data function in the GBN receiver FSM
    - Entity is the same here as specified with start_timer
    - datasent is the data received over the network that is being delivered to the application
- **RDTTester.py**
  - RDTTester.py contains the testing framework for this application, and functions similarly to IRCNetworkLauncher.py from the last assignment. You do not need to edit this file, and we will test your program with a fresh copy of it. You can turn tests off and on by commenting them out of the tests dictionary defined in the main function at the end of the RDTTester.py file.

Additional Requirements

- **Packet format:** We are using a stripped down version of the TCP header for this assignment. You should use the following packet format:
  - sequence_number (a 32-bit int)
  - ack_number (an 32-bit int)
  - checksum (a 16-bit unsigned short)
  - ACK flag (a 1-bit boolean)
  - payload length (a 32-bit int)
  - payload data (a variable length char array)
    - Payload data should be left empty, with payload length set to 0, if no data is present (e.g. an ACK packet)
- **Checksumming:** You will need to implement the Internet checksum, as discussed in the UDP lecture and the lecture explaining this project. See the lecture associated with this project for more information on how to do this in Python.

What to Submit

- Submit your SlidingWindowHost.py file