Assignment2 Code Walkthrough

Release 1.0.0

CSCI-UA.0480-062

Sep 29, 2021

CONTENTS:

1 Indices and tables 4

Python Module Index 5

Index 6

The simulator first creates a Python object for the link (link), a Python object for modeling the minimum RTT (pdbox), and a Python object to capture the end hosts, i.e., the sender and the receiver (host).

There is no separate Python object for the sender and for the receiver. Both the sender and the receiver are implemented within the same Python object called host. host.send() is called when sending packets and host.recv() is called when receiving ACKs for the packet an RTT (i.e., minimum RTT + queuing delay) later.

Similarly, there is no separate packet header format for packets and ACKs. They are one and the same. The acknowledgement process works by calling host.recv() on the same packet that was sent out as part of a host.send()in the past.

The code connects host to link, link to pdbox, and pdbox to host. Hence, packets "flow" from host (where the send() method is called to send packets) to link (where queues build up) to pdbox (where the packets incur a min.RTT worth of delay) back to host (where the recv() method is called to process ACKs for packets that were sent by host.send()).

The current value of time is represented by the variable tick, which is set in the for loop and then passed to each of the objects within the simulator. If you need to access the current value of time when implementing a protocol (e.g., for computing RTT samples, or for setting the timestamp at which a packet was sent), tick is the variable you should be using. You don't need to create your own version of time.

class aimd_host.**AimdHost**(*verbose=True*, *min_timeout=100*, *max_timeout=10000*)

This class implements a host that follows the AIMD protocol. Data members of this class are

unacked: List of unacked packets

window: Size of the window at any given moment

max_seq: Maximum sequence number sent so far

in_order_rx_seq: Maximum sequence number received so far

slow_start: Boolean to indicate whether algorithm is in slow start or not

next_decrease: Time (in ticks) at which the window size should be decreased

timeout_calculator: An object of class TimeoutCalculator (Refer to TimeoutCalculator class for more information)

There are two member functions - send and recv that perform the task of sending and receiving packets respectively. All send and receive logic should be written within one of these two functions.

__init__(*verbose=True*, *min_timeout=100*, *max_timeout=10000*)

recv(*pkt*, *tick*)

Function to get a packet from the network.

Args:

pkt: Packet received from the network

tick: Simulated time

send(*tick*)

Function to send packet on to the network. Host should first retransmit any unacked packets that have timed out. Host should also decrease the window size if it is time for the next

decrease. After attempting retransmissions, if the window is not full, fill up the window with new packets.

Args:

tick: Simulated time

Returns:

A list of packets that the host wants to transmit on to the network

class network.**DelayBox**(*prop_delay*)

A class to delay packets by the propagation delay In our case, we'll use it to delay packets by the two-way propagation delay, i.e., RTT_min

__init__(*prop_delay*)

class network.**Link**(*loss_ratio*, *queue_limit*, *verbose=True*)

A class to represent a link with a finite capacity of 1 packet per tick We can generalize this to other capacities, but we're keeping the assignment simple

__init__(*loss_ratio*, *queue_limit*, *verbose=True*)

recv(*pkt*)

Function to receive a packet from a device connected at either ends of the link. Device here can represent an end host or any other network device.

The device connected to the link needs to call the recv function to put packet on to the link. If link's queue is full, it starts dropping packets and does not receive any more packets.

tick(*tick*, *pdbox*)

This function simulates what a link would do at each time instant (tick). It dequeue packets and sends it to the propagation delay box

class packet.**Packet**(*sent_ts*, *seq_num*)

Class to represent a simulated packet. It has the following data members

sent_ts: Time at which the packet was sent

seq_num: Sequence number of the packet

pdbox_time: Arrival time at the propogation delay box

retx: To identify if the packet is a retransmission

__init__(*sent_ts*, *seq_num*)

class sliding_window_host.**SlidingWindowHost**(*window_size*, *verbose=True*, *min_timeout=100*, *max_timeout=10000*)

This host follows the SlidingWindow protocol. It maintains a window size and the list of unacked packets. The algorithm itself is documented with the send method

__init__(*window_size*, *verbose=True*, *min_timeout=100*, *max_timeout=10000*)

recv(*pkt*, *tick*)

Function to get a packet from the network.

Args:

pkt: Packet received from the network

tick: Simulated time

send(*tick*)

Method to send packets on to the network. Host must first check if there are any unacked packets, if yes, it should retransmit those first. If the window is still empty, the host can send more new packets on to the network.

Args:

tick: Current simulated time

Returns: A list of packets that need to be transmitted. Even in case of a single packet, it should be returned as part of a list (i.e. [packet])

class stop_and_wait_host.**StopAndWaitHost**(*verbose=True*, *min_timeout=100*, *max_timeout=10000*)

This host implements the stop and wait protocol. Here the host only sends one packet in return of an acknowledgement.

__init__(*verbose=True*, *min_timeout=100*, *max_timeout=10000*)

recv(*pkt*, *tick*)

Function to get a packet from the network.

Args:

pkt: Packet received from the network

tick: Simulated time

send(*tick*)

Function to send a packet with the next sequence number on to the network.

CHAPTER

ONE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

a

aimd_host, 1

e

ewma, 2

n

network, 2

p

packet, 2

s

simulator, 1
sliding_window_host, 2
stop_and_wait_host, 3

t

timeout_calculator, 3

INDEX

Symbols

__init__() (*aimd_host.AimdHost method*), 1
__init__() (*network.DelayBox method*), 2
__init__() (*network.Link method*), 2
__init__() (*packet.Packet method*), 2
__init__() (*sliding_window_host.SlidingWindowHost method*), 2
__init__() (*stop_and_wait_host.StopAndWaitHost method*), 3

A

aimd_host
    module, 1
AimdHost (*class in aimd_host*), 1

D

DelayBox (*class in network*), 2

E

ewma
    module, 2

L

Link (*class in network*), 2

M

module
    aimd_host, 1
    ewma, 2
    network, 2
    packet, 2
    simulator, 1
    sliding_window_host, 2
    stop_and_wait_host, 3
    timeout_calculator, 3

N

network
    module, 2

P

packet
    module, 2
Packet (*class in packet*), 2

R

recv() (*aimd_host.AimdHost method*), 1
recv() (*network.Link method*), 2
recv() (*sliding_window_host.SlidingWindowHost method*), 2
recv() (*stop_and_wait_host.StopAndWaitHost method*), 3

S

send() (*aimd_host.AimdHost method*), 1
send() (*sliding_window_host.SlidingWindowHost method*), 3
send() (*stop_and_wait_host.StopAndWaitHost method*), 3
simulator
    module, 1
sliding_window_host
    module, 2
SlidingWindowHost (*class in sliding_window_host*), 2
stop_and_wait_host
    module, 3
StopAndWaitHost (*class in stop_and_wait_host*), 3

T

tick() (*network.Link method*), 2
timeout_calculator
    module, 3