

Lab 3 - Task 1

In this task, we will get an overview about the transport layer simulation and measure the throughput of the wait and stop protocol.

INSTRUCTIONS

The MATLAB code in the window below simulates the stop and wait protocol. Note that the simulation in the network layer is different from that in the physical layer where we are only concerned with the one-way transmission from the transmitter to the receiver. In the network layer, the receiver also sends acknowledgement (ACK) packets back to the transmitter, which when received by the transmitter will affect its next action. As a result, we need to take the "time dimension" into consideration when simulating the behavior in the transport layer.

In our simulation, we assume a discrete-time model where we divide time up into different slots, and "events", such as packet transmissions, happen within one slot. Slots are indexed by integer time indices. For example, if the transmitter transmits a packet at time index **1** and the transmission delay is 4, then the packet will arrive at the receiver at time index **5**. Similarly, the parameter "timeout" is also set to be an integer number. Conceptually, this is similar to what we did with slotted Aloha in the network layer.

Let's now look at the code and see how the simulation works.

The code first initializes a text message (variable **tx_msg**), which we can think of as a message at the application layer that is passed to the transport layer. It then defines several simulation parameters:

1. **p_loss** indicates the probability that a packet is lost on its way between sender and receiver. In practice, this is determined by what happens at the network, link and physical layers in the nodes along the path from sender to receiver.
2. We assume the transmission delay between the transmitter and receiver follows a discrete uniform distribution between two values **d_min** and **d_max**. In practice, this is also determined by what happens at lower layers, say, for example, routing by the network layer.
3. **time_out** represents the waiting time before the sender retransmits the same packet if it does not receive the acknowledgement for that packet.

Then, the code converts the text message to a bit sequence (variable **tx_bs**) and divides the bit sequence into a list of packets (**sender_packet_list**) using the function **packetize**. In this lab, each packet contains one character. As a result, the message 'Hello World!', which has 12 characters, will be converted to 12 packets. Each packet consists three parts, namely, the sequence index of the packet (starting from 1), the total number of packets in the list and the data (the 8 bit ASCII code for the character). Since each of these three parts is 8-bits long, the packet is 24-bits long in total. After that, the code initializes the list of received packets (**receiver_packet_list**). This list will eventually contain every received packet in the order it was received, since the current implementation of the receiver does not handle duplicated and/or out of order packets.

Then, the code initializes the transport layer using the function **start_transport** and initializes the sender of the stop-and-wait protocol using the function **sender_stopwait**. The main loop of the simulation follows. Each iteration of the while loop executes the operations of the sender, receiver and the underlying network for one time slot. These are implemented by the functions **sender_stopwait()**, **receiver_stopwait()**, and **step_transport()**, respectively.

1. The sender function, **sender_stopwait()**, checks whether the timeout has been reached and/or an acknowledgement has been received. If so, it decides whether and which packet to send to the receiver. Otherwise, it does nothing. The sender function returns two values. The first one is a Boolean variable that is true when all packets of the list have been successfully transmitted (**done**). The second value tells us the number of packets that have been successfully transmitted and acknowledged so far (**noSentPackets**).

2. The receiver function, **receiver_stopwait()**, sends an acknowledgement (ACK) to the sender when it receives a packet and adds the packet to the matrix **receiver_packet_list**. The content of the ACK packet is simply the index of the received message. Note that this function does not handle duplicated or out of order packets. You will handle that in Task 3.

3. The function **step_transport** simulates the transmission of packets through the network for one time slot, and increments the simulation time.

At the end of the loop, we convert the list of packets in the matrix **receiver_packet_list** into a text message (**rx_msg**). If you run the code, you will observe that some characters are repeated because this function does not deal with duplicated packets that may occur in the stop and wait protocol. We will handle this in Task 3.

Finally, we plot a figure to show the traffic in the network at the sender and receiver for the first 200 time slots. Green lines in the figure represent packets sent by the sender. Red lines represent the acknowledgment sent by the receiver. We can observe that some characters have to be sent several times due to the packet loss and timeouts.

Your task is to revise the code to compute the throughput of the stop-and-wait protocol and save the result inside the variable **throughput**. Note that the throughput is defined as the ratio between the number of successful transmission/acknowledgements and the number of time slots needed to complete those transmissions. It has units of packets per time slot. We have provided a function **getSimTime**, which returns the total simulation time. Revise the code between the lines

```
% % % % Revise the following code % % % %
```

and

```
% % % % Do not change the code below % % % %
```

Please, do not change other parts of the code.

Your Solution



Save



Reset



MATLAB Documentation (<https://www.mathworks.com/help/>)

```
50
51 % % % % Revise the following code % % % %
52 RTT = getSimTime() / length(receiver_packet_list);
53 disp(RTT);
54
55 throughput = RTT + ((p_loss/(1-p_loss) * time_out));
56
57 % % % % Do not change the code below % % % %
```

Assessment Tests: Failed

✓ Is the problem unchanged?

✗ Is the throughput correct?

The value stored in the variable throughput is incorrect.

The throughput is incorrect!

Output

Simulation Time: 389

Number of packets sent correctly: 12

The received message is: HHHHeello Woorlld!
16.2083

Throughput: 26.2083 packets per slot

