IIT Goa CS 378 (Computer Networks LAB)
Instructor: Neha Karanjkar

# LAB 4: Reliable Transport Protocols

**Instructions:**

- This is an **individual** assignment. You can discuss concepts with other students. However sharing of code or reports with other students is not allowed. Any form of copying or collusion will lead to a double grade penalty for **all** students involved.
- This is a take-home assignment. The deadline for submission is **Friday 22 March 2pm**.

- **Format for final submission:**
  - Your final submission should consist of the following files:
    - A typeset report in pdf format. The file should be named **<ROLL NUM>.pdf**, for example "17000123.pdf". The quality of the report carries significant weightage. Remember the four 'C's of writing: your report should be complete, correct, clear and concise.
    - A Python file named **Protocol_rdt22.py** containing your implementation of the rdt2.2 protocol as asked in question 2. The code should be well-documented.
    - A Python file named **Protocol_rdt3.py** containing your implementation of the rdt3.0 protocol as asked in question 3.

  - Do **not** include any other files in your submission (such as the template files provided by the instructor). Your implementation of the protocols must adhere to the interface provided in the template. It should be possible for anyone to run the simulation with the existing template, by only adding your protocol files to the existing template folder and changing the name in the "import <protocol-file>" line in the testbench.

    Submissions that do not adhere to this format may not be graded.

---

**1.** The file Channel.py implements a model for an unreliable channel over which packets can be corrupted or lost. This model has the following parameters:
Pc: The probability of a packet being corrupted
Pl: The probability of a packet being lost
Delay: The time it takes for a packet to travel over the channel and reach the receiver.

**a)** The file Protocol_rdt1.py implements the trivial protocol rdt1.0 which works only if the channel is assumed to be ideal. Run the simulation first with Pc=0, and then Pc=0.5. Check that the protocol fails in the second case, and list the symptoms.

**b)** The file Protocol_rdt2.py implements the simple ACK/NAK based protocol rdt2.0 that can work when data packets can get corrupted. Check that this protocol indeed works when Pc>0 for the data-channel.

**c)** For the testbench using Protocol_rdt2.py, modify the code such that:

- The sending application generates a fixed total number of messages (say 1000), with a fixed time interval between each message (say 3 units of time)
- As soon as the protocol at the sending-side (rdt_Sender) receives positive acknowledgements for all of the 1000 messages, the simulation ends, and a quantity "T_avg" is printed as output, where T_avg is the time between sending a packet and receiving a positive acknowledgement for it, averaged across all packets.

  (Note that you need to run each simulation as long as necessary for all 1000 messages to be acknowledged.)

You would expect that T_avg should increase with Pc, but in what manner? (linearly? exponentially?). Obtain a plot of T_avg versus Pc for (0<=Pc<=0.9) and explain what trend you observe and why.

**d)** Protocol_rdt2.py will not work if ACK/NAK can also get corrupted. Check this by setting Pc>0 for the ack-channel and state the symptoms you observe.

**2.** Develop an alternating-bit protocol as described in K&R (rdt_2.2) which can work even when both the data and ack packets can get corrupted. Test that your protocol works by simulating for a large amount of time or a large number of packets sent, with Pc>0 for both the data and ack channels. You need to submit only the protocol as a file named "Protocol_rdt22.py".

**3.** The protocol rdt2.2 will not work if packets can be lost.

**a)** Check this by setting Pl>0 in the testbench with your implementation of Protocol_rdt22. What failure symptoms do you observe?

**b)** Implement an alternating-bit protocol with Timeouts (rdt3.0) that can work when data or ack packets can be corrupted or lost. Set the timeout value to 3*Delay. Test that your protocol indeed works by simulating for a large amount of time or a large number of packets sent, with Pc>0 and Pl>0 for both the data and ack channels. You need to submit the protocol as a file named "Protocol_rdt3.py".

**c)** You would expect that T_avg (defined the same way as in question 1) would increase with Pl. For your implementation of the rdt3.0 protocol, with channel Delay=2, Pc=0.2, timeout=3*Delay and total packets generated =1000, plot T_avg versus Pl.

**d)** Protocol rdt3.0 will not work if packets can be re-ordered over the channel. This can be easily modeled by setting the channel delay for each packet to be a randomly chosen number instead of having the same value for all packets. Thus, packet-2 sent after packet-1, might experience a lower delay and arrive ahead of a packet-1 at the receiver. Implement this feature in the model, and check if the rdt3.0 protocol indeed fails. In your report, show the code snipette that models this packet reordering.