Link Layer and ARQ Protocols

Objective:

- Understand how retransmission (ARQ) protocol works.
- Understand the effect of different parameters on the performance of the protocols, including channel capacity, propagation delay, size of data frame, size of ack frame, and error ratio.

1 Alternating Bit Protocol (ABP)

In ABP, the sender (sender here refers to the data link layer process) sends packets in data frames numbered 0 and 1 (this number is called a sequence number) alternatively and starts a timeout after each frame it sends. (The frame is called a data frame because it actually contains data from layer 3). The sender has a buffer able to contain 1 packet in which the packet currently being sent is kept until the sender is sure that it has been received correctly by the other side. The sender sends one packet at a time, i.e., it does not send packet i before being completely sure that packet $i-1$ has been correctly received. When the sender has completely finished with packet $i-1$, i.e., it has received a non-corrupted ACK for it, it empties its buffer and requests a new packet from the upper layer. The upper layer may not have a packet to send right away but is then aware of the availability of the layer 2 sending process.

The channel can corrupt or lose a data frame or an ACK frame. The receiver keeps a counter NEXT_EXPECTED_FRAME which represents the frame sequence number that it is expecting. A correct frame for the receiver is a frame that is received without error. If the receiver receives a correct frame whose sequence number is equal to NEXT_EXPECTED_FRAME, it interprets it as a new frame with a new packet. It sends the packet to Layer 3, increases its counter NEXT_EXPECTED_FRAME by 1 (modulo 2) and acknowledges the receipt of the frame by sending an acknowledgement (ACK) which is encapsulated in a frame from B to A. This frame can either be a data frame from B to A or a special frame that only contains the ACK. We will assume the latter in the following. This ACK frame carries an acknowledgement number RN equal to the updated value of NEXT_EXPECTED_FRAME. If the receiver receives a frame in error or a correct frame with an unexpected SN number (i.e., not equal to NEXT_EXPECTED_FRAME), it will not update the counter value NEXT_EXPECTED_FRAME and will send an ACK with the current value of NEXT_EXPECTED_FRAME as RN.

The sender maintains a counter NEXT_EXPECTED_ACK which is set to SN+1 (modulo 2) where SN is the sequence number of the latest frame sent by the sender. It will wait for the receivers ACK with the right ACK number (i.e., RN being equal to NEXT_EXPECTED_ACK) before emptying its buffer of the current packet, increasing SN and NEXT_EXPECTED_ACK by 1 (modulo 2) and informing the upper layer that it is ready to send a new packet. It will send a new packet in a new numbered frame as soon as the layer 3 of the sender has one to send. More precisely, when it receives a frame, if it is correct, it checks the value of RN. If RN = NEXT_EXPECTED_ACK, it does what was explained earlier in this paragraph, otherwise if the frame is correct but RN is not equal to NEXT_EXPECTED_ACK or if the frame contains errors, 2 options are possible. In the first option, the sender does not do anything, while in the second option, it resends the packet in the buffer in a new data frame with the same SN. If the sender does not receive an acknowledgement before the frames timeout, either because the ACK was not sent by the receiver or was lost or because the timeout was too short, the sender will retransmit the same packet in a new frame carrying the same sequence number SN as the original frame.

2 Simulation Framework

2.1 Discrete Event Simulation

Figure 1 shows the system that you are going to simulate:

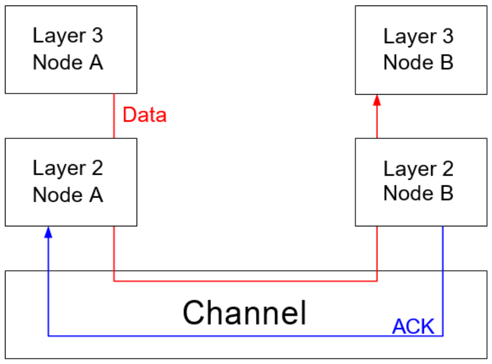


Figure 1: Model for simulation

We will assume that data always flows from node A (sender) to node B (receiver). You should bear in mind that in reality both nodes could be a sender of data and both could be a receiver of data. The channel is always bi-directional, even in this model, because data will go from A to B and ACK will go from B to A. Furthermore, we will assume that Layer 3 of node A ALWAYS has packets to send. Your simulator should be built based on all these assumptions.

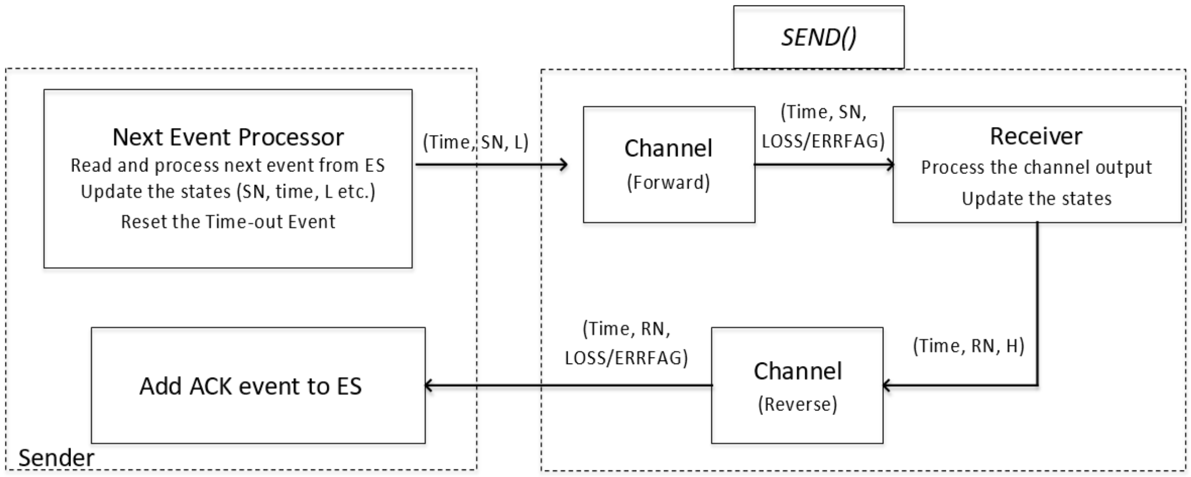


Figure 2: Functional diagram

We will use the DES (Discrete Event Simulation) framework to implement our simulator. DES is one of the basic paradigms for simulator design. It is used in problems where the system evolves over time, and the state of the system changes at discrete points in time when some events happen. These discrete points in time are not necessarily separated by equal duration. A queue called the Event Scheduler (ES) will be used to store the events in sequence and a function will be used to return the event at the head of the event scheduler (the next earliest happening event), which we call the next event. The structure of an event has to be defined, with fields (properties) which are associated with it. Among other things, an event always has a time field, telling us when it occurred. There can be multiple ways in which the DES simulator can be designed. As a guideline, we will mention one particular approach and describe the important functional elements of the design. We expect you to follow this guideline. Figure 2 shows the different functional components of this guideline design. Next, we will describe this design and the behaviours of the sender, receiver and the channel.

2.2 Details of the ABP Simulator

2.2.1 ABP Sender

The sender maintains two counters, namely SN and NEXT_EXPECTED_ACK. SN is initialized to 0 and NEXT_EXPECTED_ACK is set to 1. In the beginning, the sender also chooses a value of time-out δ which is kept constant throughout the simulation. The choice of δ affects the protocol performance greatly. The current time t_c is initialized to 0. The sender generates a packet of length $L = H + l$, where H is the fixed length of the frame header and l is the packet length. Normally, packets have variable packet lengths. However, in this simulation, we assume that all packets are of the same length. The sender stores this packet in its (only) buffer.

We assume (realistically) that the sender also knows the channel capacity C (in bits/sec). L/C is the amount of time required for the frame to be completely sent into the channel, i.e., the transmission delay. The sender sends the packet in a frame with sequence number SN, which is completely transferred to the channel by time $t_c + L/C$, and at this point it registers a TIME-OUT event in the ES (i.e., a time-out at time $t_c + L/C + \delta$). The sender then calls a function that implements together 1) the forward channel (from the sender to the receiver), 2) the receiver, and 3) the reverse channel (from the receiver to the sender). Let this function be called SEND(). It returns as a value an event that has to be registered in the ES (See Figure 2). Occasionally, a frame (on the forward channel) or an ACK (on the reverse channel) gets lost, in which case the function returns a NIL event which will not be registered in the ES. Otherwise, the return value of SEND() corresponds to an ACK event, which has a type: it can either come with or without error, and has a sequence-number RN. The sender reads the ES to get the next event. The reading process also updates the value of current time t_c, and removes the event from ES. If the next event is a TIME-OUT event (which means either the frame or the ACK was lost, or that the ACK arrived after the time-out), the same packet with the same SN is sent to the channel in a new frame. On the other hand, if the next event is an ACK without error and RN is equal to NEXT_EXPECTED_ACK, the sender knows its packet has been received correctly, so it increments SN and NEXT_EXPECTED_ACK by 1 (mod 2), generates a new packet, and sends a new frame. Note that there can be only one time-out in the ES, and hence as soon as the frame is completely passed on to the forward channel (i.e., at $t_c + L/C$), any outstanding time-outs in ES have to be purged and a new time-out at $t_c + L/C + \delta$ has to be registered. Event processing takes non-zero time (for example, sending a frame takes L/C time), and we will assume that once the sender starts processing an event, it does not get interrupted by another event. The events are processed sequentially without interruptions.

2.2.2 Channel

The channel is characterized by its propagation delay τ where τ is assumed to be a constant quantity. A data frame (for the forward channel) or an ACK (for the reverse channel) can be lost with a probability P_{LOSS} (which, in general, can be different for a frame and an ACK). A lost frame/ACK never appears on the other end of the channel. With a probability $1 - P_{\text{LOSS}}$, the frame/ACK arrives at the other end of the channel, with two possibilities: a) it arrives in error (frame ERROR), b) it arrives without any error (frame NOERROR). These three events (frame LOSS, frame ERROR and frame NOERROR) are mutually dependent and usually have intricate relationships. To simplify the relationships among these outcomes, we assume that the channel has a given bit error rate BER (the probability that a bit arrives in error, considered independently from one bit to another) and we arbitrarily consider a frame experiencing 5 or more bit errors to be a lost frame. If a frame experiences from 1 to 4 bits in error, it is considered to be a frame in ERROR. When all bits arrive correctly, we consider this a successful arrival of the frame (NOERROR). We can compute these probabilities for a frame of length L as follows.

$$P_{\text{NOERROR}} = (1 - \text{BER})^L$$

$$P_{\text{ERROR}} = \sum_{k=1}^{4} \binom{L}{k} \text{BER}^k (1 - \text{BER})^{(L-k)}$$

$$P_{\text{LOSS}} = 1 - P_{\text{NOERROR}} - P_{\text{ERROR}}$$

Note that we do not expect you to use the above stated formulas to generate the NOERROR, ERROR or LOSS event. We expect you to run L iterations. In each iteration, you generate 0 with probability BER, and 1 with probability $1 - \text{BER}$. You then count the number of zeroes. The number of zeroes represents the number of bits in error and thus corresponds to one of the three events.

Functionally speaking, the channel will take the time t_c, SN and the length of the frame/ACK as inputs and will either return nothing (NIL) if the frame/ACK is lost or return the time (after adding the delay τ), a flag representing whether the frame/ACK encountered an error or not, as well as the SN (which is unchanged).

2.2.3 ABP Receiver

The receiver maintains a NEXT_EXPECTED_FRAME counter, initialized to 0. The current time t_{cs} is also initialized to 0. The receiver acts on the return values of the forward channel. It updates its current time t_{cs} based on the channel return value. If the channel returns a frame with no errors, it checks the frame's SN and if it is equal to the NEXT_EXPECTED_FRAME, it counts it as a new and successfully delivered packet, which it sends to Layer 3 (in our case, it just increments a counter). This successful delivery is accompanied by increasing NEXT_EXPECTED_FRAME by 1 (mod 2). In all cases, it sends an ACK to the reverse channel of fixed length H, with the sequence number RN set to NEXT_EXPECTED_FRAME. Note that the return values from the reverse channel will be registered in the ES as described earlier. The forward and the reverse channel are functionally equivalent and hence should be implemented by one function definition. Also, it is important to make sure that correct tracking of time is carried out at both the sender and receiver implementation. As said already, the forward channel, receiver and the reverse channel should be wrapped into one functional block, to be used by the sender.

3 Experiment

A simulation experiment will comprise of the following parameters.

(1) Sender-side parameters: H, l, δ.

(2) Channel parameters: C, τ, BER.

(3) Experiment duration in terms of number of successfully delivered packets to be simulated.

These parameters will affect the performance of an ARQ protocol. In this experiment, you will need to investigate how a change in one or more of these parameters affects the throughput of an ARQ protocol. We will assume that the size of the frame header (which is also the size of an ACK) is fixed to $H = 54$ bytes and the packet length is fixed to $l = 1500$ bytes (i.e., a data frame has a constant size of $H + l$)

Implement the ABP sender, channel and receiver as described above using the first option (namely the sender does nothing when it receives an ACK in error or a correct ACK with an RN not equal to NEXT_EXPECTED_ACK). Call this simulator ABP. Also, bundle the forward channel, receiver and the reverse channel components into one procedure. Also, implement an event scheduler, with a function for registering an event and a function for dequeueing the earliest event. With the necessary components in place, integrate them to implement the simulator. Accompany your code with build scripts, and brief documentation explaining the key components of your implementation.

(i) Consider the scenario with BER = 0. Use your simulator to compute the throughput (bits/sec) as a function of δ for $C = 5$Mb/s, and two values of τ (5ms and 250ms). Note that while computing the throughput, you should not count the header bits. Simulate at least 10,000 successfully delivered packets. Take the values of δ from the set $\{2.5\tau, 5\tau, 7.5\tau, 10\tau, 12.5\tau\}$.

(ii) Now, repeat the set of experiments in (i) with BER = 10^{-5} and BER = 10^{-4}.

4 Submission

Choose one programming language to do the simulation from C/C++, Python, and Java. Submit a zip file containing the followings files.

- Complete source and simulation codes.
- Simulator Run Script (run_ABP). Your code has to compile and run successfully in a Linux environment.
- A report file (pdf) containing at least
  - A brief explanation of your simulator ABP.
  - Relevant plots (as shown in the next section).

Submit your zip file on Moodle.

5 Submission Guideline

Read the following submission guidelines carefully.

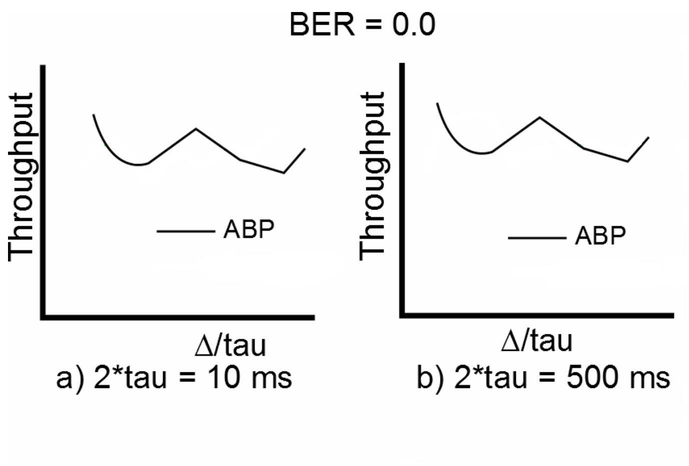
δ/τ	**2τ = 10ms**			**2τ = 500ms**		
	BER=0.0	BER=1e-5	BER=1e-4	BER=0.0	BER=1e-5	BER=1e-4
2.5	a1	a2	a3	a4	a5	a6
5	b1	b2	b3	b4	b5	b6
7.5	c1	c2	c3	c4	c5	c6
10	d1	d2	d3	d4	d5	d6
12.5	e1	e2	e3	e4	e5	e6

5.1 Summary of result data

The above table summarizes the throughput results that you need to generate from the simulations. Hence your report should include the above table for the simulations.

5.2 Summary of required plots

Include plots (as shown) and discuss the findings in your report. Include similar plots for BER = 10^{-5} and 10^{-4}.


BER = 0.0
Throughput
ABP
Δ/tau
a) 2*tau = 10 ms
Throughput
ABP
Δ/tau
b) 2*tau = 500 ms