Data Networks
HW 2: Data Link Layer

Dr. Mohammad reza Pakravan

Theoretical Problems

This section is comprised of five theoretical problems that aim at helping you build an intuition about some concepts within data link layer technologies and protocols.

Burst Errors and Interleaving

Suppose that the sequence of bits $b_0b_1...b_{n-1}$ is the result of encoding a w-bits binary message using a repetition code for which the repetition order is $r = 2k+1$ for some integer $k \geq 0$. Consequently, we have

$$b_{mr} = b_{mr+1} = \ldots = b_{mr+r-1} \quad \forall m \in \{0, 1, \ldots, w-1\}.$$

Before transmission, the bits are interleaved to form the following sequence which is then sent over a bursty channel:

$$b_0b_r...b_{(w-1)r}b_1b_{r+1}...b_{(w-1)r+1}...b_{r-1}b_{2r-1}...b_{(w-1)r+r-1}.$$

When this sequence is transmitted, the channel alters a single burst of bits with length B. The receiver performs de-interleaving and decoding in order to extract the w message bits.

1. What is the maximum size of B for which the transmission system can work with no errors? (Hint: Recall that the repetition decoder uses majority vote on the received encoded bits to decide on the transmitted bit.)
2. Suppose we use the interleaving system in previous question, and assume that the channel keeps causing a single error burst per transmission. As we observed, for a fixed value of B, there is a minimum safe (in terms of bit error) value for w. What is the drawback of using a w, larger than this minimum?

Link Utilization

Frames of 32 KB are sent over a 100 Mbps link with a propagation delay of 3 ms. Four bits are allocated for frame sequence numbering. The frame error probability is 8×10^{-2}. Compute link utilization in the case of using:

- Stop and Wait
- Go-Back-N
- Selective Repeat

Selective Repeat

Selective Repeat attempts to retransmit only those packets that are actually lost due to errors. Consider that the window size is w and NAK is not employed; instead, we trust the timeout mechanism of the sender. So the sender retransmits un-ACKed packets after a timeout. The cumulative ACK method is utilized, meaning that whenever the receiver gets a packet, whether in order or not, it sends an ACK saying "I got every packet up to and including k", where k is the highest, in-order packet received so far.

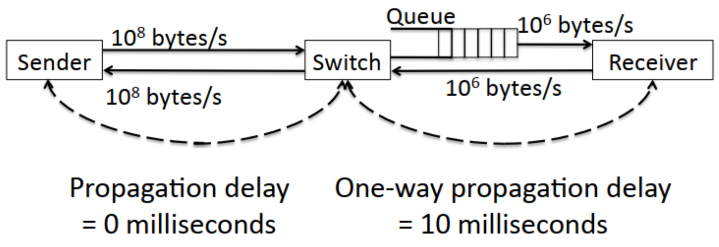
Suppose that The sender sends a stream of new packets according to the sliding window protocol, and in response gets the following cumulative ACKs from the receiver:

1 2 3 4 5 5 5 5 5 5 5

1. Now, suppose the sender times out and retransmits the first unacknowledged packet. When the receiver gets that retransmitted packet, what can you say about the ACK, a, that it sends?
   (a) $a = 6$
   (b) $a \geq 6$
   (c) $6 \leq a \leq 12$
   (d) $a = 12$
   (e) $a \leq 12$
2. Determine the greatest lower bound of a.

Sliding Window Protocols

In the network below, the receiver sends end-to-end ACKs to the sender. The switch in the middle simply forwards packets.



Network assumptions:
Max queue size = 30 packets
Packet size = 1000 bytes
ACK size = 40 bytes
Sender window size = 10 packets

At what rate will the protocol deliver files from the sender to the receiver? Assume that there is no other traffic in the network and packets can only be lost because the queues overflow.

Character Stuffing

1. Using the Character Stuffing method, the receiver gets the following sequence as a payload:

   ESC, ESC, ESC, FLAG, ESC, ESC

   Find the original characters before stuffing.

2. Is it possible to receive the following sequence as a payload in the receiver?

   ESC, FLAG, ESC, ESC, FLAG

Practical Problems

Introduction

OMNeT++ is an object-oriented modular discrete event network simulation framework. It has a generic architecture, so it can be (and has been) used in modeling of wired and wireless communication networks, protocol modeling, modeling of queuing networks and in general, modeling and simulation of any system where the discrete event approach is suitable, and can be conveniently mapped into entities communicating by exchanging messages. This homework guides you through building a data link layer protocol, showing you along the way some of the commonly used OMNeT++ features. To practically follow the steps, having a good C++ knowledge, and being in general familiar with C/C++ development (editing source files, compiling, debugging etc.) is necessary.

Getting Familiar With OMNeT++

Download and install OMNeT++ using the Installation guide. Run a sample project to make sure of the correct installation.

Stop and Wait

In this part, we simulate stop-and-wait protocol by following the steps below from Learn OMNeT++ with TicToc tutorial.

1. **TicToc Network**: Begin with a network that consists of two nodes. The nodes will do something simple: one of the nodes will create a packet, and the two nodes will keep passing the same packet back and forth. The nodes are called tic and toc. Use Part 1 - Getting Started to run the simulation and to get familiar with setting up a project in OMNeT++ and adding NED, C++, Omnetpp.ini files.

2. **stop-and-wait**: Proceeding to 3.8 Timeout, cancelling timers, transform your model to a stop-and-wait simulation.

3. **frame length**: Use 3.6 Modeling processing delay to add frame length in time for tic node. Model this time in your simulation as a delay in tic before sending every message. Thus, this is not an accurate method to simulate frame length in time, but nevertheless it achieves the desired result. Consider frame length of ACK to be zero. Add proper log to your simulation and provide enough screenshots from your logs and simulation model to explain how the model is working properly.

4. **Link Utilization**: Compute link utilization using simulation time and frame length time in your code. By following further steps from Learn OMNeT++ with TicToc tutorial provide a plot of link utilization in time and compare your results with the following formula from course lectures:

$$U = \frac{1-p}{1+2a}$$

Change propagation delay, frame length and probability of frame error in your simulation. Analyze and explain how link utilization behaves as these parameters change.

Visualizing Data Link Activity

Visualizing network traffic is an important aspect of running simulations. INET provides various visualizers to help you understand the network activity, including *DataLinkVisualizer* which focuses on the data link level. This module allows you to see the visual representation of the data link level traffic in the form of arrows that fade as the traffic ceases.

In this part, we will implement two simulation models of a showcase from the INET documentation step by step, to get more familiar with OMNeT++ and its features. For each section, add proper logs to your simulation and provide enough screenshots from your logs and simulation model to explain how the model is working properly.

1. **Enabling Visualization of Data Link Activity**:

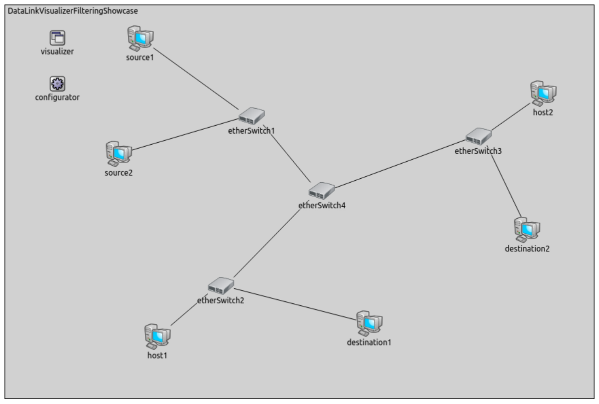
   In INET, data link activity can be visualized by including a *DataLinkVisualizer* module in the simulation. Adding an *IntegratedVisualizer* module is also an option because it also contains a *DataLinkVisualizer* module. Data link visualization is disabled by default; it can be enabled by setting the visualizer's *displayLinks* parameter to true.

   Now we want to configure a simulation for a wired network that contains two *StandardHosts* as *wiredSource* and *wiredDestination*. The *linkVisualizer* module's type should be *DataLinkVisualizer*.

2. **Filtering Data Link Activity**:

   In complex networks with many nodes and several protocols in use, it is often useful to be able to filter network traffic and visualize only the part of the traffic we are interested in. The following simulation shows how to set packet filtering in *DataLinkVisualizer*.

   (a) First, implement the network shown in the image below:



source1 pings *destination1*, and *source2* pings *destination2*. For this network, the visualizer's type is *IntegratedVisualizer*. Set the data link activity visualization in a way that it is filtered to display only ping messages. Adjust the *fadeOutMode* and the *fadeOutTime* parameters so that the activity arrows do not fade out completely before the next ping messages are sent.

(b) Now, add *source3* and connect it to the *etherSwitch1*. Add *destination3* and connect it to the central Ethernet switch (*etherSwitch4*) using *etherSwitch5*. *source3* pings *destination3*. One of the features of *dataLinkVisualizer* is filtering the traffic of specific nodes. Let's assume we want to display traffic between the hosts *source3* and *destination3* only, along the path *etherSwitch1*, *etherSwitch4*, and *etherSwitch5*. Set the visualizer's *nodeFilter* parameters to achieve this goal.

What Should I Do?

You must upload (.cc/.ned/omnetpp.ini) files in a folder named Part1 for Stop and Wait simulation and Part2 for Visualizing Data Link Activity. You should also add wanted screenshots and answer the questions in your report. Report should include

- brief explanation of how your code works
- screenshots from network and logs to justify every step
- plots of link utilization and explanation of its behaviour as differnet parameters change

Compress all files and rename the compressed file to STUDENT_ID_HW2.zip.
If you have any questions regarding the problem statement or understanding the concept, feel free to ask in Telegram.