Introduction to Computer Communications

Programming Assignment 1: multiple access channel

Submitted by: Nicole Frumkin (ID 211615372) & Or Shalev (ID 211381942)

Documentation

In this assignment, we implemented a simple network communication system using an Aloha-like protocol with two main components: a channel program and a server program. We made several design choices to ensure reliability, efficiency, and accurate simulation of the Aloha-like protocol behavior.

Channel.c

We designed our channel program to act as a central hub that enables multiple servers to communicate through a shared medium. We implemented collision detection in a way similar to the classic Aloha protocol.

- **Linked list for server management**
  We used a linked list to store information about each server that's connected. This approach allows us dynamic addition of unlimited amount of servers.
- **Select() for timing**
  We used select() function to implement timing and to handle reading from a number of sockets connected instead of threads.
- **Socket sets for monitoring**
  We used the windows struct fd_set to track active connections and detect collisions.

The program initializes a TCP socket and listens for incoming connections.

Once a server has connected, we add it to the linked list and initialize all the relevant fields.

Then the program runs in an infinite loop, and checks for incoming packets each slot_time in a for loop that traverses all the available sockets.

If we detected in the for loop 2 sockets or more that sent a message, the channel detects a collision and updated the relevant fields accordingly. It send a special "NOISE" signal to all connected servers.

If only one packet arrived, it extracts the frame size from the packet header and then the channel sends it back to all servers. This enables a dynamic frame size.

Finally, the channel calculates the statistics and other data for the printout.

Server.c

We chose an header in size of size 18 bytes because Ethernet header frames are usually 16. But we added extra two bytes to store the frame size.

The first 6 bytes are a fictional destination MAC address, the next 6 bytes are a fictional source MAX address, the next 2 bytes are the EtherType and the 4 last bytes are the frame size.

First we connect the channel using TCP. The server reads the file and sends it in frames.

After the server sends a message, it receives one back. It checks if it equal to the unique "NOISE" signal, and if it is, it detects a collision.

When it detects a collision, it uses an exponential backoff to handle it. It waits a random amount of time that increases with each collision.

After 10 consecutive collisions, the server reports a failure.

Similarly to the channel, in the end it calculates the necessary statistics.

Data Structures

- **Input struct**
  It stores all the command line arguments for both our server and channel programs: chan_port, slot_time, chan_ip, file_name, frame_size, seed and timeout.
- **Output structures**
  - **Server**
    The output structure is used the store the data we need for the output print: file_name, success, file_size, num_of_packets, total_time, max_transmissions, avg_transmissions and avg_bw.
  - **Channel**
    The output structure is a node in a linked list that stores all the relevant information about the sockets: socket, frame_size, sender_address, port_num, num_packets, total_collisions, avg_bw, start_time, end_time, send_in_slot, data_buffer, data_size and next.