Computer Networks Programming Assignment — PA1

Elad Landau — 206999054
Daniel David Goren — 322610874

This project implements a simplified communication protocol over a shared channel using concepts similar to **ALOHA**. Multiple servers send a file through a central channel, handling collisions with exponential backoff.

Files

- `server.cpp` — Sends a file over a shared channel, splitting it into frames and handling collisions.
- `channel.cpp` — Acts as a channel that routes data between servers, simulates collisions, and sends ACKs or noise.
- `protocol.h` — Defines the `Frame` structure and headers used in communication.
- `Makefile` — Builds both the `server` and `channel` executables.

Building

Make sure you're on **Linux** and have `g++` installed. To compile:

```
make
```

This will create: - `my_Server` — the server executable - `my_channel` — the channel executable

To clean up:

```
make clean
```

Running the Simulation

Start the Channel

```
./my_channel <chan_port> <slot_time>
```

- `chan_port`: The port number the channel listens on.
- `slot_time`: Duration of each time slot in **milliseconds**.

Example:

```
./my_channel 6342 100
```

Press **Ctrl+D** to end the channel and print a summary report.

Start a Server

```
./my_Server <chan_ip> <chan_port> <file_name> <frame_size> <slot_time> <seed> <timeout>
```

- `chan_ip`: IP address of the channel (e.g., `127.0.0.1`).
- `chan_port`: Same as used above.
- `file_name`: Path to the file to be sent.
- `frame_size`: Max payload size (must be `MAX_PAYLOAD_SIZE`).
- `slot_time`: Same unit and value as the channel.
- `seed`: Seed for the random number generator (affects backoff).
- `timeout`: Timeout in **seconds** for waiting on ACK.

Example:

```
./my_Server 127.0.0.1 6342 test.bin 1024 100 42 5
```

Implementation Limitations

- The maximum allowed **frame size** is limited to `MAX_PAYLOAD_SIZE` bytes (`MAX_FRAME_SIZE - sizeof(FrameHeader)`).
- If a server is started with a larger `frame_size`, it will exit with an error.
- `MAX_FRAME_SIZE` is defined as 4096 bytes.

Output

Each server logs: - Whether the transmission was successful - Number of frames sent - Retransmission stats - Average bandwidth used

The channel logs (on termination via Ctrl+D) for each server: - Number of collisions encountered

---

Design Rationale & Implementation Highlights

Ethernet-style Frame Header

The assignment required an Ethernet-style frame header. The `FrameHeader` includes:

- `source_id`, `dest_id`: 6-byte MAC-like identifiers.
- `ether_type`: Set to 0x0800 for IPv4 (as an example).
- `payload_type`: Distinguishes data (`0x01`) from noise (`0xFF`) frames.

Collision Detection and Noise Frames

- The channel detects collisions when multiple servers send in the same slot.
- It broadcasts a noise frame to all clients (`payload_type == 0xFF`).
- Each sender involved in that time slot increments its collision counter.

Server Protocol and Backoff

- After sending a frame, the server waits for a response (ACK).
- If a noise frame or no response is received, it retries up to 10 times.
- Exponential backoff is applied using `slot_time × random(k)` where `k` `[0, 2^attempts - 1]`.

Event Loop & Multiplexing

- `select()` is used in the channel to monitor all sockets and detect `stdin` EOF (Ctrl+D).
- All sockets are non-blocking to avoid hanging behavior.

Server Identification

- Server `source_id` is derived from the `getpid()` system call, packed into the first 4 bytes.

Termination and Reporting

- Channel prints stats on each connected server: number of collisions.
- Server prints final metrics: success/failure, total time, average bandwidth, etc.
