Assignment 3 Part A Design Documentation

Jiaye Wang jiw561 11231145

March 22, 20221

Design

Only three source files are included for this application: `seonsor.c` `tcp.h`, and `tcp.c`. The `tcp.c/h` files simply contain the the methods to generate TCP socket for client and server. Main implementions are inside `sensor.c`. Note that there is no basestation for this application.

General Design

To run the application:

```
./sensor <ID> <port> <storage space> <msg>.
```

The application restricts only to use uppser-case English letters as `ID`, **which implies maximum of 26 sensors are allowed at the same time.** Each `ID` of course has a unique `port`. Finally `storage space` is at least 1 and `msg` is up to 10 characters.

Since only 26 sensors are allowed at one time and every sensor's packet, `msg` is unique. Therefore, each sensor only has to buff 25 + 1 (+1 is itself's) packets later in the transmission. Each buffer entry except sensor's own entry is initialized as empty, use `#` to represent.

The `packet` structure contains two fields, `id` and `data`. It just uses for better transmission organization.

Sensor Design

Use `fd_set` and `select()` to observe the server socket and `stdin` file descriptors. Before any incoming connections, the application allows user to input. Once the user input starts with a `C` and a port `p`, then a new connection establishs with port `p`, hence, something happened to the sever socket, which it is an incoming connection, and the sever accepts the connection. Thus, all preparations for the connection have been completed

The protocol transmit data only in one direction. For instance, if sensor A is listening on port 30000, the sensor B, listening on port 30001 called command `C30000`. Only A will be 'infected' by B, and B will not 'infected' by A. However, A can call `C30001` to 'infect' B. Moreover, the incoming packet does not buffer if sensor's `storage space` is full. Unless, the sensor calls the `B` command to transmit data to the base station and clean the buffer.

The sensor output to `stdout` to simulate it has transmitted data to the base station. All the buffer entries except its own entry will be reset to empty. Note that there are no separate variables to maintain the storage space, rather the application can through the buffer and count number of non-empty entries, hence `O(n)`, and of course can reduce to `O(1)`.

Since the buffer size is only 26, so that scan all the entry of the buffer is pretty fast, hence `O(n)`. Therefore, the buffer size or allowed number of `ID` can be expand (to reasonable amount).

Finally, the `Q` command will terminate the application and close all the socket.

Testing

The test involves three sensors, A, B, and C, and perform some triangular transmission. This can be $A \rightarrow B$, $B \rightarrow C$ and C to base station to check if it contains both A and B' packets. Before C transits to the base station, C can 'infect' A or B, hence all three sensors should have same buffer now. Then test whether the sensor can store packets if its storage space is full. Finally, check does the sensor buffer is cleaned except its own data after sent to the base station.

Limitation

- The application only works on Tuxworld 8. Since the command line argument does not have a host name option and the application assumes all the sensor will run on the same machine. The user is able to change the host name inside `sensor.c` if they want to run on other machine.
- The transmission between sensors can be two directions, but the design for this application is only one direction.
- The code is too verbose, many places can be divided into sub-functions.
- Since the author not fully understand how the `fd_set` and `select()` work, so there is not a prompt message for user input.