Assignment 3 – Project Report
CMPUT 379 – Fall Term 2018
Student Name: Jacob Bakker

Objectives

The objectives of this assignment are to become familiar with developing client-server programs capable of sending and receiving formatted, meaningful data to and from one another. Implementing the client-server relationship required implementing non-blocking I/O functionality for monitoring and receiving incoming data as it becomes available.

Design Overview

Listed below are the primary classes for implementing the Controllers and Switches and the transfer of messages through sockets and FIFOs:

Controller Class:

- Server connectivity is achieved through the Cont-Server class. All polling as well as sent and received data involving connected Switches is done through the Cont_Server.
- On startup, the Controller will poll its connected Switches for their OPEN Packets. For each Packet received, the Controller saves the Switch's info and sends an ACK Packet.
    - The Controller will not read any additional Packets from Switches it has acknowledged until all Switches have been connected (i.e. if nswitch = 4, the Controller will not read additional data from running Switches until 4 Switches have been accepted).
- When queried by a Switch about some Header, the Controller will examine the IP addresses of all other Switches to find one that servers the Header's destination IP.
    - If one exists, the Controller compares its ID to that of the querying Switch to determine what port the Header should be forwarded to.
    - If the destination Switch ID is greater than the querying Switch, the port number will be 2; otherwise, it will be 1 (see the "Assumptions" section).
    - If none exist, the Controller will set the action type of the Rule to DROP.
- The Controller detects Switch closure through writing to the Switch via the Cont_Server class. If the write returns -1, Cont_Server throws an exception with an error code indicating a Switch's closure, causing the Controller to stop polling that Switch and notify the user of its closure.

Switch Class:

- Server connectivity is achieved through the Sw-Client class, which handles polling and data transferal.
- On startup, the Switch class sends an OPEN packet containing info on its ID, neighboring Switches, and served IP addresses to the Controller before waiting for an ACK Packet. On being acknowledged, the Switch opens FIFOs to adjacent Switches (if any) in addition to installing the initial rule described in the assignment specification.
- Traffic file lines containing headers are passed to the Header class for parsing.

- If the Header specifies a delay, the Timer class is used to delay reading from the traffic file for the given delay.

Packet Class:

- The (de)serialization functions for Packet split messages into a Packet type (e.g. OPEN) and a string message; save for ACK Packets, the message is a serialized object (e.g. Rule, Header).
- Contains functions "write_to_fd()" and "read_from_fd()" for writing to and reading from given file descriptors.
    - These functions are invoked both by the Cont_Server and Sw_Client classes for transferring messages between Controllers and Switches via TCP sockets as well as by the Switch class for relaying Packets containing Headers to each other via FIFOs.

Listed below is a list containing brief descriptions of the simpler features of the project (e.g. headers, rules):

- Cont_Server includes basic functions for polling clients for incoming data in addition to receiving and sending Packets using the read and write functions available in the Packet class.
- Sw_Client includes basic operations for polling the server in addition to receiving and sending Packets via functions in the Packet class,
- The Rule class includes basic (de)serialization functions for storing Rules as Packet messages in addition to a "print()" function for displaying the Rule attributes during logging operations. The function "is_match()" is used to check whether a given Header matches a Rule for use by Switches.
- The Header class includes basic (de)serialization functions and a "print()" function for reasons outlined above. The Header class is intended to handle the parsing of lines from the traffic file of the formats "swi x y" and "swi delay z", allowing Switches to pass these lines to the header to be deserialized. Furthermore, the delay value "z" is stored for reference by the Switch.
- The number of Packets received and sent for each Packet type is monitored by the ContStats class for Controllers and the SwStats class for Switches. Included is a "print()" function for the "list()" functions in both the Controller and Switch classes.
- The Timer class implements a basic interface for setting a timer for a specific duration and checking whether that specific duration has passed since the Timer was started. This allows Switches to delay reading from the traffic file by starting the Timer for z milliseconds on reading a Header with the format "swi delay z".

Project Status

Server functionality seems to be fully implemented, as Controllers and Switches can poll one another for incoming data and send or receive data from one another without blocking. Controllers can detect which Switches have unexpectedly terminate without crashing.

Switches can parse the traffic file format given in the assignment specification without issue, admitting headers and starting traffic-reading delays as required. Switches are also able to communicate with one another via FIFOs.

The difficulties encountered during the project mostly centered on server and FIFO functionality, mainly getting the server and client(s) to serialize data, send it, and have the recipient deserialize and use the data.

Testing and Results

- Tested "Example: 2 switches with transmitted, received, and delayed messages" from CMPUT 379 eClass.
    - List commands produced identical results, and log messages for received and transmitted messages from the example were present in the test.
    - The messages did not appear in the intended order, with log messages for ADD Packets from the Controller appearing before the log message for the QUERY Packet it was responding to.
- Testing termination of Switches (both by CTRL-C and the "exit" command) while the Controller was still running.
    - The Controller did not crash and printed a descriptive message containing the ID of the closed Switch.

Assumptions

- Required FIFOs are created before running "a3sdn", and these FIFOs have the naming format "fifo-x-y" where x is the Switch ID of the writer and y is the ID of the reader.
- Neighboring Switches are in ascending order (e.g. for SW3, SWJ is either null or its ID is less than 3, while SWK is either null or greater than 3.

Acknowledgements

- The Controller server class (Cont_Server.cc) features code for starting the server and accepting clients that is derived from the CMPUT 379 lecture notes "4. Sockets", pages 21-24 (Citations listed above Cont_Server constructor and "accept_clients()").
- The Switch client class (Sw_Client.cc) features code for connecting to a server that is derived from the CMPUT 379 lecture notes "4. Sockets", pages 16-20. (Citation listed above Sw_Client constructor).
- The structure of the project's make file (a3Makefile) is derived from the following post on StackOverflow detailing how to structure a make file to compile a project split between a header folder ("include/") and a program folder ("src/"): "https://stackoverflow.com/a/30602701"