CMPUT 379 Assignment 3 Report

By Nathan Klapstein #1449872

Table of Contents

Contents

Objectives

a3sdn

The main objective of a3sdn was to expand upon the developments of a2sdn a linear software-defined network (SDN) that composed of one main controller and multiple switches (sw1, sw2, … sw7). Some extensions a3sdn introduced was that communication between switches was done via named pipes (FIFOs) and a common traffic file while communication between switches and controllers was done via TCP sockets. Additionally, the requirements of non-blocking I/O still apply for a3sdn. Also, the requirement for switches to disconnect and have the SDN still operate "effectively" was added.

Controller

The controller for a3sdn is required to handle the creation and management of routing rules to be applied to active switches. Communication between the controller and each connected switch is done via TCP sockets (establishing the connection as a server). The controller is required to keep tabs on each connected switch such as noting each switch's accepted source and destination IPs and their neighboring switches.

Switch

Each switch for a3sdn is required to store, understand, and apply routing logic provided by the controller. Communication between the switch and the controller is done via TCP sockets (the switch connects to the controller as a client). Communication between other neighboring switches is done via a named pipes (FIFO)s

Note: for a more detailed report on the objectives required by a3sdn please read:

https://eclass.srv.ualberta.ca/mod/url/view.php?id=3306105

Design Overview

a3sdn

- C++ was used over C due to its added functionality
- Class-based structure was used to separate functionality
- Used a cmake file over a makefile for development (more modular/futureproof)
- Connection class was created to represent two connection FIFOs (one sending and one receiving) between a switch-switch
  - This kept things simple and helped construct an API to interact easily between switches and controllers.
- Custom address class was used to control how addresses were validated/parsed and accessed
- Packet class was created to keep transfers between switches and the controller uniform
- poll() was used due to its simplicity of handling multiple and dynamic file descriptors
- Majority of print statements have starting strings of ERROR: WARNING: INFO: and DEBUG: for ease of debugging
- Signals were handled using a signalfd (see signalfd.h) thus, allowing for poll to query them
  - This kept implementation uniform between polling stdin, the connection FIFOs, and signals
- Controller and Switch classes inherit from the abstract class Gate to avoid code duplication
  - Gate class implemented the ==, <, and > operators for deduping and sorting the list of switches used by the controller

- Specific functionality was attempted to be separated to avoid "god-like" files within a3sdn
- Specific object classes were created (e.g. SwitchID) to keep type checking and state verification simple and low maintenance.

Controller

- Controller class was created to represent a controller
    - handled construction and argument parsing
    - start() method started the controller's poll() loop
    - Class functionality for responding to OPEN, QUERY Packets
    - Class functionality for sending ACK, and ADD Packets
    - list() provides statistics on all received Packet types for debugging usage
- multiple accepted TCP socket connections were stored in a vector clientSocketConnections

Switch

- Switch class was created to represent a switch
    - Handled construction and argument parsing
    - start() method started the switch's poll() loop
    - Class functionality for responding to ACK, ADD, and RELAY Packets
    - Class functionality for sending OPEN, QUERY, and RELAY Packets
    - list() provides statistics on all received Packet types for debugging usage along with packets delivered to the switch.
    - The switch will error and exit if a controller is not active (this seemed acceptable as we don't want any dangling switches running for no reason)
- FlowEntry structure was used for modeling flow rules for the switches
- FlowTable (a vector of FlowEntrys) was used for modeling the rules list for the switch

Project Status

a3sdn

Status: Completed – Limited Testing

Code Source: https://github.com/nklapste/a3sdn

Some difficulties encountered in the design of a3sdn was in maintaining a uniform format. Multiple refactoring/restructuring efforts were needed to obtain to the current build of a3sdn. Using a more easily implemented Object Orientated (OO) language such as Java would likely remedy this issue without any major reductions in speed.

Another major difficulty was in developing code for starting, reading, and accessing TCP sockets for c++. I initially had much difficulty in allowing multiple messages (null terminated '\0' strings) be sent between the controller and switch. It took many designs, reworks, and research before a working system was established. It is quite mind boggling why a simpler API (and defacto standard for creating such connections) has not been established for c++ (for c I can understand why we keep things raw and rough).

Software such as a3sdn is better off written first in a high-level language (e.g. python/java) for prototyping structure and API concepts (i.e. defining standard packet format, control flows, etc…). Then finally, if speed is a priority re-constructing the software in a lower language (e.g. C, C++, **Rust**) for speedups. Looking at major SDN software packages such as lighty.io, Cherry, Facuet, ONOS, ect… all are written in higher level languages such as Java, Python, Go, over older languages such as C, and C++. This can be likely attributed to the increased modularity, ease of development, and agnostic behavior between operating systems. C and C++ still have structures that can lead to "undefined behavior" (many examples are online) and incorporating a language that contains such dangers into software that can be potentially routing packets of private/protected information is poor engineering.

Testing and Results

a3sdn

Testing for a3sdn consisted of simple runtime testing over the basic supported commands and normal use cases.

Some basic test traffic files were created for testing the general functionality of a3sdn. They can be seen bundled with the source code named tf1.txt, tf2.txt, and tf3.txt (which are inherited from a2sdn), and tf4.txt (which tests a3sdn functionality).

Note: the traffic files tf1.txt, tf2.txt, tf3.txt, and tf4.txt are bundled within the submit.tar and are also located within the project's repository for convenience.

tf4.txt

Running tf4.txt and spawning 1 controller and 3 switches provides the following results.

Controller

The controller was first started using the following command:

```
./a3sdn cont 3 9982
```

The last few controller output lines after running all the switches and later disconnecting all the switches is shown below:

```
INFO: switch: sw1 disconnected: fd:5 ip:127.0.0.1 port:49968
INFO: switch: sw2 disconnected: fd:6 ip:127.0.0.1 port:49970
INFO: switch: sw3 disconnected: fd:7 ip:127.0.0.1 port:49972
exit
Controller information:
[sw1] port1= null, port2= sw2, port3= 100-110
[sw2] port1= sw1, port2= sw3, port3= 500-550
[sw3] port1= sw2, port2= null, port3= 200-220
Packet Stats:
        Received:    OPEN:3, ACK:0, QUERY:17, ADDRULE:0, RELAYIN: 0, ADMIT:0
        Transmitted: OPEN:0, ACK:3, QUERY:0, ADDRULE:17, RELAYOUT:0
INFO: exit command received: terminating
```

Switch sw1

Switch sw1 was then started next using the following command:

```
./a3sdn sw1  test/tf4.txt  null sw2  100-110 localhost 9982
```

The last few output lines after adding/disconnecting all the switches is shown below:

```
exit
sw1 FlowTable:
[0] (srcIP= 0-1000 dstIP 100-110 action=FORWARD:3 pri= 4 pktCount= 8)
[1] (srcIP= 0-1000 dstIP 700-700 action=DROP:0 pri= 4 pktCount= 1)
[2] (srcIP= 0-1000 dstIP 200-220 action=FORWARD:2 pri= 4 pktCount= 2)
[3] (srcIP= 0-1000 dstIP 500-550 action=FORWARD:2 pri= 4 pktCount= 3)
Packet Stats:
        Received:    OPEN:0, ACK:1, QUERY:0, ADDRULE:6, RELAYIN: 5, ADMIT:8
        Transmitted: OPEN:1, ACK:0, QUERY:7, ADDRULE:0, RELAYOUT:5
INFO: exit command received: terminating
```

Switch sw2

Switch sw2 was then started next using the following command:

```
./a3sdn sw2  test/tf4.txt  sw1  sw3  500-550 localhost 9982
```

The last few output lines after adding/disconnecting all the switches is shown below:

```
exit
sw2 FlowTable:
[0] (srcIP= 0-1000 dstIP 500-550 action=FORWARD:3 pri= 4 pktCount= 8)
[1] (srcIP= 0-1000 dstIP 700-700 action=DROP:0 pri= 4 pktCount= 1)
[2] (srcIP= 0-1000 dstIP 200-220 action=FORWARD:2 pri= 4 pktCount= 5)
[3] (srcIP= 0-1000 dstIP 100-110 action=FORWARD:1 pri= 4 pktCount= 5)
Packet Stats:
        Received:    OPEN:0, ACK:1, QUERY:0, ADDRULE:5, RELAYIN: 9, ADMIT:8
        Transmitted: OPEN:1, ACK:0, QUERY:5, ADDRULE:0, RELAYOUT:10
INFO: exit command received: terminating
```

Switch sw3

Switch sw3 was then started next using the following command:

```
./a3sdn sw3  test/tf4.txt  sw2  null 200-220 localhost 9982
```

The last few output lines after adding/disconnecting all the switches is shown below:

```
exit
sw3 FlowTable:
[0] (srcIP= 0-1000 dstIP 200-220 action=FORWARD:3 pri= 4 pktCount= 8)
[1] (srcIP= 0-1000 dstIP 700-700 action=DROP:0 pri= 4 pktCount= 1)
[2] (srcIP= 0-1000 dstIP 100-110 action=FORWARD:1 pri= 4 pktCount= 3)
[3] (srcIP= 0-1000 dstIP 500-550 action=FORWARD:1 pri= 4 pktCount= 3)
Packet Stats:
        Received:    OPEN:0, ACK:1, QUERY:0, ADDRULE:5, RELAYIN: 5, ADMIT:8
        Transmitted: OPEN:1, ACK:0, QUERY:5, ADDRULE:0, RELAYOUT:6
INFO: exit command received: terminating
```

Note: After starting all the switches they were then disconnected in the following order (sw1, sw2, sw3).

Acknowledgments

cplusplus.com for C++ documentation:

http://www.cplusplus.com/

C++ undefined behavior references:

https://en.cppreference.com/w/cpp/language/ub

List of SDN controller software:

https://en.wikipedia.org/wiki/List_of_SDN_controller_software

CLion IDE for C++ refactoring and development tools:

https://www.jetbrains.com/clion/

Tutorial on Cmake:

https://www.cs.swarthmore.edu/~adanner/tips/cmake.php