Modeling, Simulation and Visualization of a Software Defined Network Environment using Cadmium

Assignment1

AmirHoseein Ghorab
amirhoseeinghorab@cmail.carleton.ca
101206662

Carleton University

Contents

Part 1: System and model

System Description and Model Structure

A simplified SDN network environment will be modeled and simulated in this project. The system contains four different components.

- SDN controller
- SDN enabled witch
- Webserver
- Client (User)

The client asks for a service (e.g., Http service) from the web server by sending a request. This request is received by the switch in the middle of the way. Unlike the traditional switch that only forwards the packets based on its rule sets in the switch table, the SDN enabled switch buffers the packet and compares its destination IP with the switch flow table entry. If the destination is found in the flow table, the packet will be sent to the destined server. Otherwise, the packet will be sent to the controller over a secure connection asking for the action that needs to be done on the packet. The SDN controller analyzes the packet and sends the packet back to the switch setting the packet's destination into the switch's flow table. Then the switch forwards the packet to the web server. The webserver receives the packet and sends an ACK packet to the client through the switch. This procedure will be repeated for each new type of flow with a new destination. The module structure of the aforementioned network is shown in Figure 1.

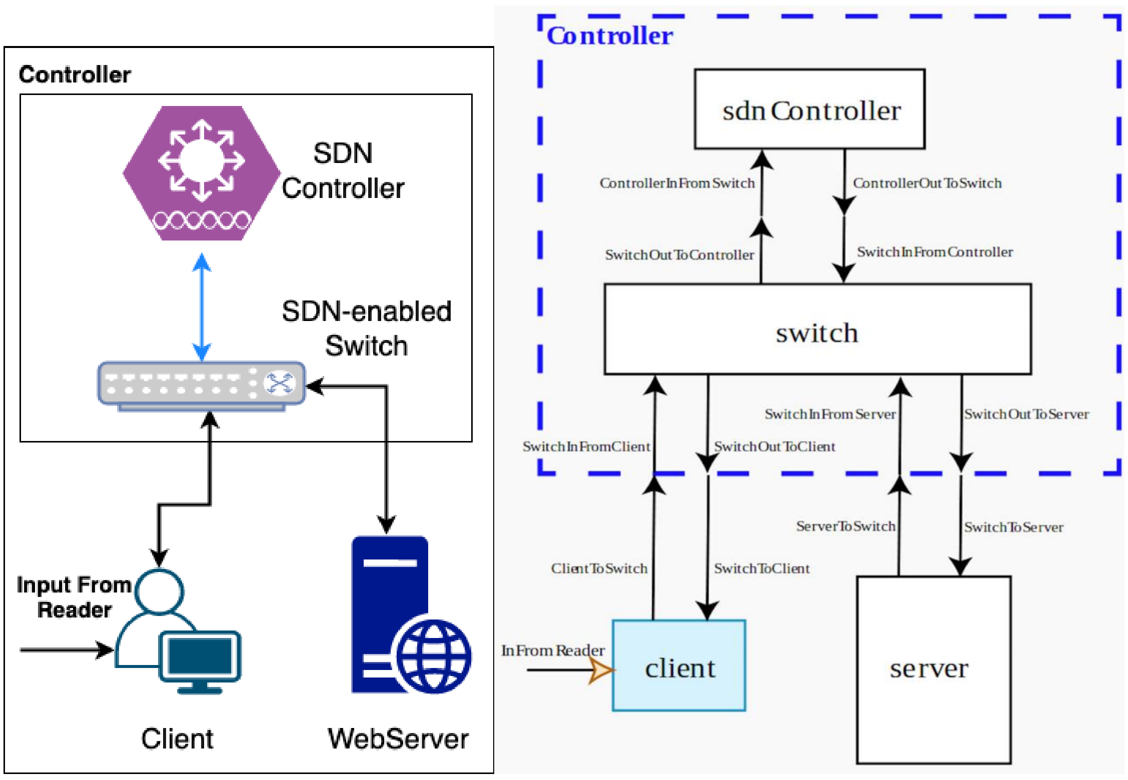
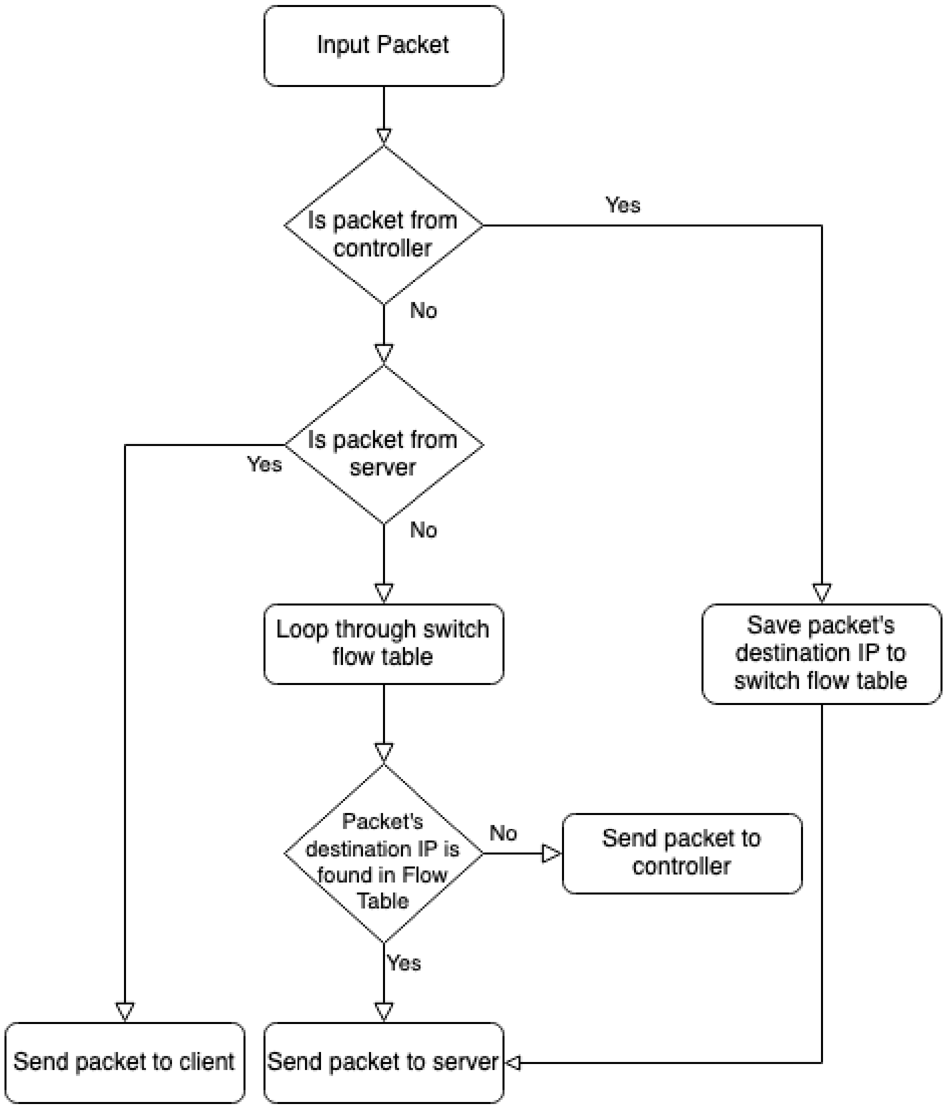


Figure 1: SDN environment and structure

A simplified SDN enabled switch's algorithm is shown in Fig. 3.



Input Packet
Is packet from controller
Yes
No
Is packet from server
Yes
No
Loop through switch flow table
Save packet's destination IP to switch flow table
Packet's destination IP is found in Flow Table
No
Send packet to controller
Yes
Send packet to client
Send packet to server

Figure 2: Switch's algorithm for handling input packets

Part 2: Formal specification

The formal specifications <S, X, Y, δint, δext, λ, ta> for the atomic models are defined as follows:

- **Server**

  State variables:

  - Transmitting
  - Data

  S = {passive, active}

  X = {in}

  Y = {out}

  δint (active) = passive

  δext (in, passive) = active

  δext (in, active) = active

  λ(active)

  {Sends input packet (as an ACK) to port out}

  ta(passive) = INFINITY

  ta(active) = transmitting time

The simplified implementation of **δint**, **δext** and **output** function are as follow.

```
// internal transition
internal_transition() {
        state.transmitting = false;
}
// external transition
external_transition() {
        Receiving packet from input port and save it
        state.transmitting = true;
}
// output function
output () {
        Send saved data to the output port
}
```

- **Client**

  State variables:

  - Ack: Set to be true when receiving the data from server
  - PacketNum : Current packet sequence
  - totalPacketNum : Total number of packets that need to be sent to server

- DstIP : Packet destination IP address
- Sending: Set to be true when sending data to server
- Model_active
- Data

S = {sending, ack, passive}

X = {InFromReader, InFromSwitch}

Y = {OutToSwitch}

The implementation of **δint**, **δext** and **output** function are as follow.

```
// internal transition
internal_transition() {
    if (state.ack) {
        Increase packet sequence number
        state.ack = false;
    } else {
        if (state.sending) {
            state.sending = false;
            state.model_active = true;
        } else {
            state.sending = true;
        }
    }
}

// external transition
external_transition(input from reader) {
    Receiving packet from reader and save it
    state.ack = false;
    state.sending = true;
}
external_transition(input from switch) {
    Receiving packet from server and save it
    state.ack = true;
    state.sending = false;
}

// output function
output (active && sending) {
    Send packet to the output port
}
```

- **Switch**

  State variables:
  - Mood: An enum which holds the state of the switch (TransmittingToDestination, TransmittingToController, TransmittingToClient, Passive)
  - FlowTable: A vector type variable which holds IP destinations
  - Data

S = {TransmittingToDestination, TransmittingToController, TransmittingToClient, Passive}

X = {inFromController, inFromServer, inFromClient}

Y = {outToController, outToServer, outToClient}

The implementation of **δint**, **δext** and **output** function are as follow.

```
// internal transition
internal_transition() {
    state.mood = ModelMood::Passive;
}
// external transition
external_transition() {
    state.data = x;
    for (Input data from client)
    {
        If (packet's destination is found in flow table) {
            state.mood = ModelMood::TransmittingToDestination;
        } else {
            state.mood = ModelMood::TransmittingToController;
        }
    }
    For (Input data from server) {
        state.mood = ModelMood::TransmittingToClient;
    }
    For (Input data from controller) {
        Save packet's destination IP to flow table
        state.mood = ModelMood::TransmittingToDestination;
    }
}
// output function
output () {
    if (state.mood == ModelMood::TransmittingToDestination){
        Send data to server
    }
    else if(state.mood == ModelMood::TransmittingToController){
        Send data to controller
    }
    else if(state.mood == ModelMood::TransmittingToClient){
        Send data to client
    }
}
```

The formal specifications < X, Y, D, M, EIC, EOC, IC, SELECT> for the coupled model Controller and SDN Simulator are defined as follows:

- **Coupled Model: Controller**

  Controller = < X, Y, {sdnController, switch}, EIC, EOC, IC, SELECT >

  X = {inputFromReaderToClient}

  Y = {}

EIC = { (inputToSwitchFromClient, switch.inFromClient), (inputToSwitchFromServer, switch. inFromServer)}

EOC = {(switch.outToClient, outputFromSwitchToClient), (switch.outToServer, outputFromSwitchToServer)}

IC = {(switch.outToController, sdnController. inFromSwitch), (sdnController.outToSwitch, switch.inFromController)}

SELECT: ({sdnController, switch}) = switch;

- **Coupled Model: SDN**

SDN = < X, Y, {Controller, client, server}, EIC, EOC, IC, SELECT >

X = {inputFromReaderToClient}

Y = {}

EIC = { (inputFromReaderToClient, client.inFromReader)}

EOC = {}

IC = {(outputFromSwitchToClient, client.inFromSwitch), (outputFromSwitchToServer, server.inFromSwitch)}

SELECT: (client, Controller, server) = client;

(Controller, server) = Controller;

- **Coupled Model: TOP**

TOP = < X, Y, {InputReader, SDN}, EIC, EOC, IC, SELECT >

X = {}

Y = {}

EIC = {}

EOC = {}

IC = {(inputReader.out, inputFromReaderToClient)}

SELECT: (InputReader, SDN) = SDN;

Part 3: Testing strategy

For each atomic model (Client, Server, SDN Controller, and Switch) an input test file has been provided in the "/test" folder. The test input applies different conditions on the model to verify the correctness of each model. The structure of the file for all input is as follow:

{<Time> <Packet sequence (or total packet) (int)> <Destination IP address (String)>}

NOTICE, some parts of the input/output results have been removed to reduce the size of the report document. The full results can be found in the **simulation_results** folder ("simulation_results/").

- **Switch Test Cases** (test path: **"**test/switch_client_input_test.txt", result path: "simulation_results/switch_client_input_output_state.txt" and "simulation_results/switch_client_input_output_messages.txt"):

  Test Case 1:

  For the switch atomic model, part of the test file is shown below

Time	Packet sequence	Destination IP
00:00:10	1	192.168.1.5
00:00:20	2	192.168.1.5
00:00:30	3	192.168.1.5
00:00:40	4	192.168.1.5

  After running the model with the test file as an input, the result is as follow:

```
00:00:00:000
State for model input_reader is next time: 00:00:00:000
State for model switch is : Passive
00:00:00:000
State for model input_reader is next time: 00:00:10:000
State for model switch is : Passive
00:00:10:000
State for model input_reader is next time: 00:00:10:000
State for model switch is : Transmitting To Controller --> Packet sequence number #1 destination IP 192.168.1.5
00:00:15:000
State for model input_reader is next time: 00:00:10:000
State for model switch is : Passive
00:00:20:000
State for model input_reader is next time: 00:00:10:000
State for model switch is : Transmitting To Controller --> Packet sequence number #2 destination IP 192.168.1.5
00:00:25:000
State for model input_reader is next time: 00:00:10:000
State for model switch is : Passive
00:00:30:000
State for model input_reader is next time: 00:00:10:000
State for model switch is : Transmitting To Controller --> Packet sequence number #3 destination IP 192.168.1.5
00:00:35:000
State for model input_reader is next time: 00:00:10:000
State for model switch is : Passive
```

```
00:00:40:000
State for model input_reader is next time: 00:00:10:000
State for model switch is : Transmitting To Controller --> Packet sequence number #4 destination IP 192.168.1.5
```

According to the SDN model that was defined in Part1 and 2, when a new packet arrives at the switch, the switch compares the packet's destination IP with the switch flow table. If the packet destination IP is not found, the switch sends the packet to the controller. In the above test, at time 00:00:10:000 (orange bold font) new packet arrives. Since this packet's destination IP is not found in the switch flow table, the switch sends this packet to the controller (State for model switch is: Transmitting To Controller). However, since the switch is not connected to the controller in this test, there would be no answer from the controller resulting in packet loss. The same thing happens to other received packets.

Test Case 2:

In this test part of the input file is shown below.

```
00:00:10 1 192.168.1.5
00:00:20 2 192.168.1.5
.
.
.
00:01:50 1 192.168.1.7
00:02:00 2 192.168.1.7
.
.
.
```

Output result after running the test.

```
00:00:00:000
State for model input_reader is next time: 00:00:00:000
State for model switch is : Passive
00:00:00:000
State for model input_reader is next time: 00:00:10:000
State for model switch is : Passive
00:00:10:000
State for model input_reader is next time: 00:00:10:000
State for model switch is : Transmitting To Controller --> Packet sequence number #1 destination IP 192.168.1.5
00:00:15:000
State for model input_reader is next time: 00:00:10:000
State for model switch is : Passive
00:00:20:000
State for model input_reader is next time: 00:00:10:000
State for model switch is : Transmitting To Destination --> Packet sequence number #2 destination IP 192.168.1.5
00:00:25:000
State for model input_reader is next time: 00:00:10:000
State for model switch is : Passive
                                        .
                                        .
```

.

```
00:01:50:000
State for model input_reader is next time: 00:00:10:000
State for model switch is : Transmitting To Controller --> Packet sequence number #1 destination IP 192.168.1.7
00:01:55:000
State for model input_reader is next time: 00:00:10:000
State for model switch is : Passive
00:02:00:000
State for model input_reader is next time: 00:00:10:000
State for model switch is : Transmitting To Destination --> Packet sequence number #2 destination IP 192.168.1.7
00:02:05:000
State for model input_reader is next time: 00:00:10:000
State for model switch is : Passive
```

In this test the switch is connected to the controller, therefore, only the first sequence of each packet with new destination IP is sent to the controller. Then the destination IP is saved to the switch's flow table and the rest of the packets are forwarded to the destination as expected.

- **SDN Controller Test Cases** (test path: "test/sdnController_test.txt", result path: "simulation_results/SDN_output _state.txt" and "simulation_results/SDN_output _messages.txt"):

  In this test, the SDN controller receives packets from the test input file and add the flow rule (destination IP) to the switch's flow table.

  Input test file:

```
00:00:10 1 192.168.1.5
00:00:20 1 192.168.1.7
00:00:30 1 192.168.1.12
```

  Output results.

```
00:00:00:000
State for model input_reader is next time: 00:00:00:000
State for model sdnController is : Passive
00:00:00:000
State for model input_reader is next time: 00:00:10:000
State for model sdnController is : Passive
00:00:10:000
State for model input_reader is next time: 00:00:10:000
State for model sdnController is : Adding flow rule to Switch's flow table for packet flow to destination: 192.168.1.5
00:00:15:000
State for model input_reader is next time: 00:00:10:000
State for model sdnController is : Passive
00:00:20:000
State for model input_reader is next time: 00:00:10:000
State for model sdnController is : Adding flow rule to Switch's flow table for packet flow to destination: 192.168.1.7
00:00:25:000
```

```
State for model input_reader is next time: 00:00:10:000
State for model sdnController is : Passive
00:00:30:000
State for model input_reader is next time: 00:00:10:000
State for model sdnController is : Adding flow rule to Switch's flow table for packet flow to destination: 192.168.1.12
```

In this test, the SDN controller adds new destination IP to switch’s flow table for each new input packets.

- **Server Test Cases** (test path: “test/server_input_test.txt”, result path: “simulation_results/server_output _state.txt” and “simulation_results/server_output _messages.txt”):

  Input file:

```
00:00:10 1 192.168.1.5
00:00:20 2 192.168.1.5
00:00:30 3 192.168.1.5
```

  Output results:

```
00:00:00:000
State for model input_reader is next time: 00:00:00:000
State for model server is : Passive
00:00:00:000
State for model input_reader is next time: 00:00:10:000
State for model server is : Passive
00:00:10:000
State for model input_reader is next time: 00:00:10:000
State for model server is : Send back Ack for packet sequence #1 From Destination: 192.168.1.5
00:00:11:000
State for model input_reader is next time: 00:00:10:000
State for model server is : Passive
00:00:20:000
State for model input_reader is next time: 00:00:10:000
State for model server is : Send back Ack for packet sequence #2 From Destination: 192.168.1.5
00:00:21:000
State for model input_reader is next time: 00:00:10:000
State for model server is : Passive
00:00:30:000
State for model input_reader is next time: 00:00:10:000
State for model server is : Send back Ack for packet sequence #3 From Destination: 192.168.1.5
```

The server is a simple atomic model. It receives the packet and sent the packet back to the client (ACK) after some delay.

- **Client Test Cases** (test path: **"**test/client_input_test.txt", result path: "simulation_results/client_output_state.txt" and "simulation_results/client_output _messages.txt"):

The client model receives the input data from the file input reader model and generates packets. The format of the input data is as follows.

{<Time> < Number of packets that need to be generated> <detonation IP>}

However, the client does not generate the packets continuously. The client generates new packets only if it receives the ACK from the previous packet. If no ACK receives, the client sends the packet again.

Test input file:

```
00:00:10 6 192.168.1.5
00:04:40 3 192.168.1.7
.
.
.
```

Output results:

```
00:00:00:000
State for model input_reader is next time: 00:00:00:000
State for model client is : Passive
00:00:00:000
State for model input_reader is next time: 00:00:10:000
State for model client is : Passive
00:00:10:000
State for model input_reader is next time: 00:04:30:000
State for model client is : Sending Packet sequence #1 to destination 192.168.1.5 | total packet number: 6
00:00:20:000
State for model input_reader is next time: 00:04:30:000
State for model client is : Passive
00:00:40:000
State for model input_reader is next time: 00:04:30:000
State for model client is : Sending Packet sequence #1 to destination 192.168.1.5 | total packet number: 6
00:00:50:000
State for model input_reader is next time: 00:04:30:000
State for model client is : Passive
00:01:10:000
State for model input_reader is next time: 00:04:30:000
State for model client is : Sending Packet sequence #1 to destination 192.168.1.5 | total packet number: 6
00:01:20:000
State for model input_reader is next time: 00:04:30:000
State for model client is : Passive
```

```
00:01:40:000
State for model input_reader is next time: 00:04:30:000
State for model client is : Sending Packet sequence #1 to destination 192.168.1.5 | total packet number: 6
00:01:50:000
State for model input_reader is next time: 00:04:30:000
State for model client is : Passive
00:02:10:000
State for model input_reader is next time: 00:04:30:000
State for model client is : Sending Packet sequence #1 to destination 192.168.1.5 | total packet number: 6
00:02:20:000
State for model input_reader is next time: 00:04:30:000
State for model client is : Passive
```

In this test, since the client is not connected to the server (no ACK), it tries to send the first sequence (#1) of data again and again.

- **TOP Model Test Cases** (test path: **"**test/sdn_input_test.txt", result path: "simulation_results/sdn_output_state.txt" and "simulation_results/sdn_output _messages.txt"):

  In this test, the TOP model integrates atomic models client, server, and coupled model Controller. The packets are sent from client to server through the Controller coupled model.

  Test input file:

```
00:00:10 6 192.168.1.5
00:04:40 3 192.168.1.7
.
.
.
.
```

  Output results:

```
00:00:00:000
State for model input_reader is next time: 00:00:00:000
State for model client is : Passive
State for model server is : Passive
State for model sdnController is : Passive
State for model switch is : Passive
00:00:00:000
State for model input_reader is next time: 00:00:10:000
State for model client is : Passive
State for model server is : Passive
State for model sdnController is : Passive
State for model switch is : Passive
00:00:10:000
State for model input_reader is next time: 00:04:30:000
State for model client is : Sending Packet sequence #1 to destination 192.168.1.5 | total packet number: 6
State for model server is : Passive
State for model sdnController is : Passive
State for model switch is : Passive
00:00:20:000
State for model input_reader is next time: 00:04:30:000
State for model client is : Passive
State for model server is : Passive
State for model sdnController is : Passive
State for model switch is : Transmitting To Controller --> Packet sequence number #1 destination IP 192.168.1.5
00:00:25:000
State for model input_reader is next time: 00:04:30:000
State for model client is : Passive
State for model server is : Passive
State for model sdnController is : Adding flow rule to Switch's flow table for packet flow to destination: 192.168.1.5
State for model switch is : Passive
00:00:30:000
State for model input_reader is next time: 00:04:30:000
State for model client is : Passive
State for model server is : Passive
State for model sdnController is : Passive
State for model switch is : Transmitting To Destination --> Packet sequence number #1 destination IP 192.168.1.5
00:00:32:000
State for model input_reader is next time: 00:04:30:000
State for model client is : Passive
State for model server is : Send back Ack for packet sequence #1 From Destination: 192.168.1.5
State for model sdnController is : Passive
State for model switch is : Passive
00:00:33:000
State for model input_reader is next time: 00:04:30:000
State for model client is : Passive
State for model server is : Passive
State for model sdnController is : Passive
State for model switch is : Transmitting To Client --> Packet sequence number #1 destination IP 192.168.1.5
00:00:35:000
State for model input_reader is next time: 00:04:30:000
State for model client is : Receiving ACK for Packet sequence #1 from destination 192.168.1.5
State for model server is : Passive
State for model sdnController is : Passive
State for model switch is : Passive
00:00:35:000
State for model input_reader is next time: 00:04:30:000
State for model client is : Sending Packet sequence #2 to destination 192.168.1.5 | total packet number: 6
State for model server is : Passive
State for model sdnController is : Passive
State for model switch is : Passive
00:00:45:000
State for model input_reader is next time: 00:04:30:000
State for model client is : Passive
State for model server is : Passive
State for model sdnController is : Passive
State for model switch is : Transmitting To Destination --> Packet sequence number #2 destination IP 192.168.1.5
00:00:47:000
State for model input_reader is next time: 00:04:30:000
State for model client is : Passive
State for model server is : Send back Ack for packet sequence #2 From Destination: 192.168.1.5
State for model sdnController is : Passive
State for model switch is : Passive
00:00:48:000
State for model input_reader is next time: 00:04:30:000
State for model client is : Passive
State for model server is : Passive
State for model sdnController is : Passive
State for model switch is : Transmitting To Client --> Packet sequence number #2 destination IP 192.168.1.5
00:00:50:000
State for model input_reader is next time: 00:04:30:000
State for model client is : Receiving ACK for Packet sequence #2 from destination 192.168.1.5
State for model server is : Passive
State for model sdnController is : Passive
State for model switch is : Passive
```

.

.

.

In this test, at time 00:00:20:000, the first sequence of the packet (sequence #1), is sent to the controller by the switch. Then the controller adds this packet destination IP (192.168.1.5) to the switch flow table at time 00:00:25:000. Now the destination IP has been added to the switch’s flow table so when a new packet (Sequence number #2) arrives at time 00:00:45:000, it is sent to the destination immediately and no transmission to the controller is needed.

Part 4: Visualizing results using ARSLabDEVS web viewer

In the project folder, there is a folder called “SDN_WebViewer” which contains:

- Diagram.svg
- Messages.log
- Options.json
- Structure.json
- SDN_output_messages.webm (This is a recorded video created by ARSLabDEVS Web Viewer)

Viewing results on Web Viewer

- Load “Diagram.svg”, “Messages.log”, “Options.json”, “Structure.json” to the web viewer
- Click on the “Load simulation”

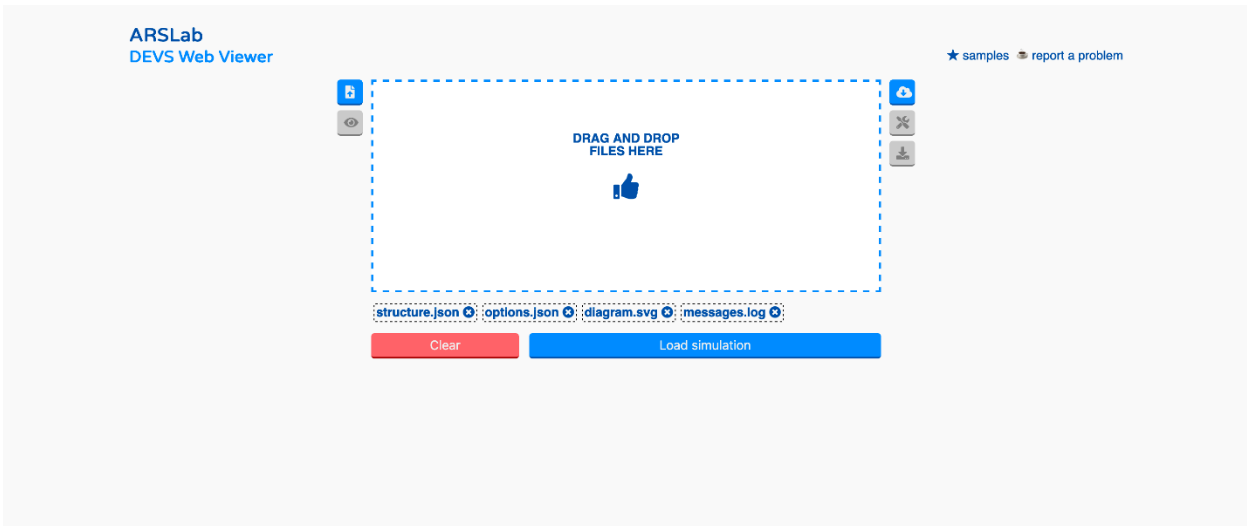


Fig 3. Loading the simulation files

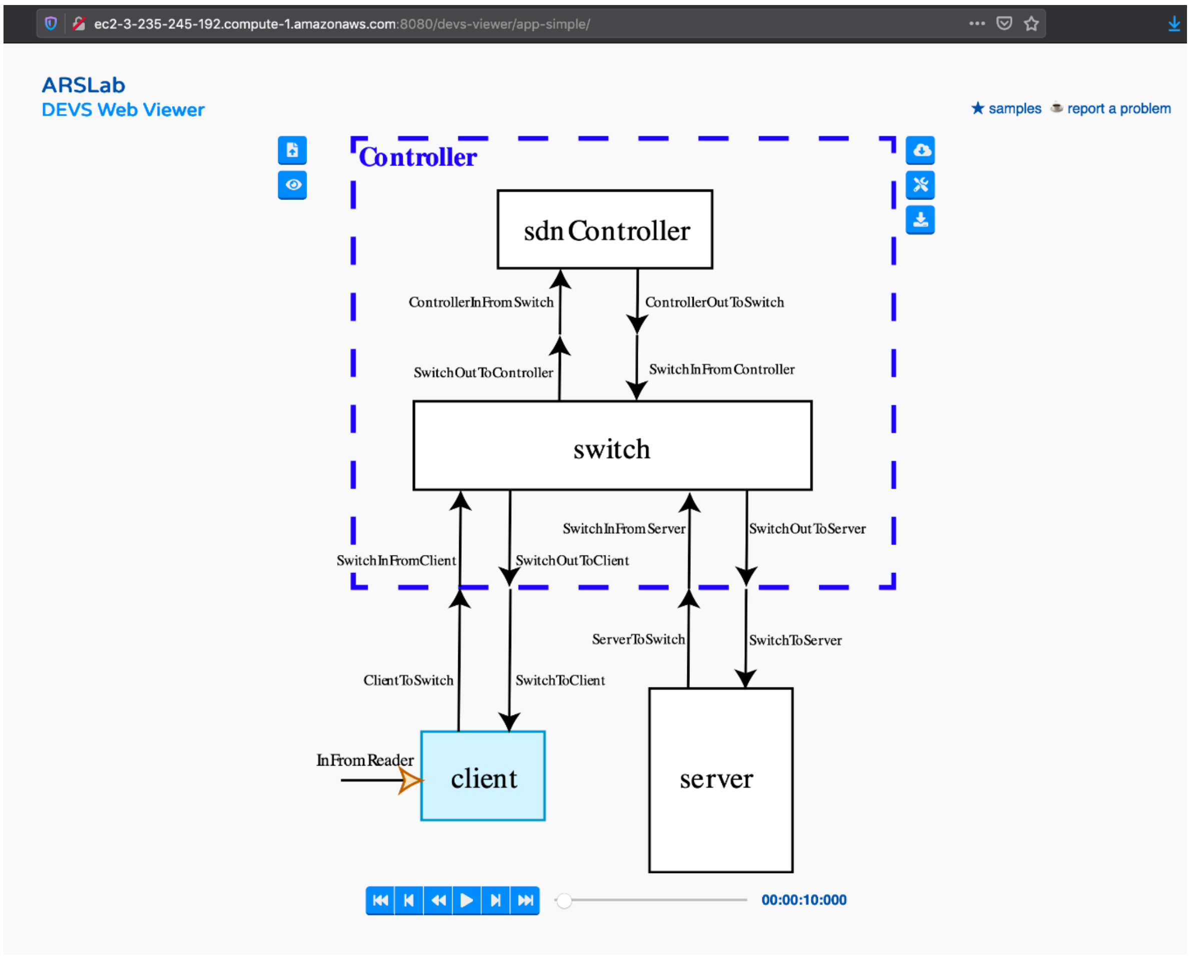


Fig 4. Running the simulation