Software-Defined Networking Module in NS-3: A Custom SDN Module Implementation

Sanjana Ganesh Nayak
Daksh Mathur
ECE 6110

May 9, 2025

Introduction

Software-Defined Networking (SDN) is a creative approach to designing, building, and managing networks. By decoupling the control plane from the data plane, SDN introduces a centralized architecture where a controller defines traffic decisions, while switches and routers focus solely on forwarding packets. This separation enhances network flexibility, programmability, and enables dynamic traffic control.

In this project, we implemented a custom SDN framework in the NS-3 network simulator. Our implementation comprises three major components: an SDN controller, a custom SDN switch, and a flow table mechanism to enforce routing rules. The objective was to replicate SDN functionality within NS-3, validate its operation through simulation, and pave the way for future work in programmable networks.

Background on SDN and NS-3

Traditional network devices are typically designed with a monolithic architecture, where the data plane (responsible for packet forwarding) and the control plane (responsible for routing decisions) are tightly integrated. This coupling limits network flexibility, as any change in control logic often requires manual configuration or hardware updates across all devices. Software-Defined Networking (SDN) offers a paradigm shift by logically centralizing the control plane in a separate software

entity known as the SDN controller. This separation enables network-wide programmability, easier management of complex policies, and more responsive traffic engineering.

NS-3, a discrete-event network simulator widely used in academic and industrial research, provides a rich set of tools for simulating traditional network architectures. However, it lacks native support for SDN concepts, such as controller-switch abstractions and programmable flow tables. This gap motivated our effort to develop a custom SDN module within NS-3. Our goal was to design and implement a modular SDN framework that reflects realistic software-defined behaviors and integrates seamlessly with NS-3's existing node, device, and protocol models.

System Design and Architecture

Our SDN implementation follows a modular design that reflects the separation of concerns and functional decomposition found in real-world SDN architectures. By clearly delineating responsibilities between different components, we ensure that each module can be independently developed, tested, and extended. This design philosophy enables a scalable and maintainable framework within NS-3, aligning with the core principles of SDN. Below, we describe each of the major components in detail.

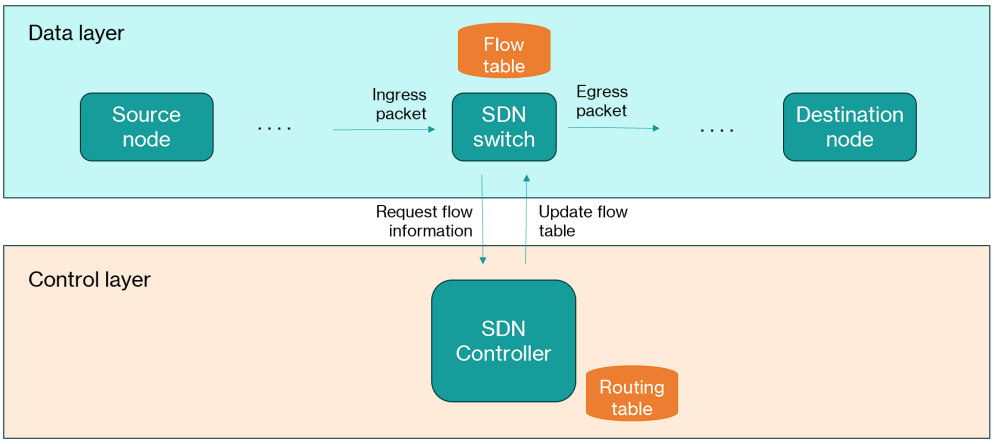

Figure 1: Overview of the custom SDN architecture implemented in NS-3.

Implementation Details

SDN Controller

The SDN controller serves as the brains for network intelligence and decision-making. It maintains a complete view of the network topology through internal data structures that map graph nodes and links, enabling shortest-path routing based on either hop count or other customizable weights. The controller is responsible for computing these routes, populating a routing table with next-hop decisions, and responding to switches when they encounter packets that do not match any known flow entry. Upon such an event, the switch triggers a `HandlePacketIn` method that forwards the packet to the controller. The controller then calculates the appropriate path using the known topology, updates the output mapping, and sends the packet out using `SendPacketOut`. Additionally, the controller maintains ARP mappings and a utility to display the full routing table for debugging or verification purposes. It supports dynamic interaction with multiple SDN switches, each of which registers with the controller using `AddSwitch`.

SDN Switch

The SDN switch acts as the interface between the data plane and the control plane in our framework. Each switch contains a flow table and a list of devices it controls. When a packet is received on any of its interfaces, it triggers the `ReceivePacket` method, which determines whether the packet matches an existing flow rule using the `LookupFlow` method. If a matching entry is found, the switch uses the associated output device to forward the packet through the `ForwardPacket` method. Otherwise, it communicates with the SDN controller to obtain new routing instructions. The switch is also responsible for installing new flow entries through `InstallFlowEntry` and updating flow statistics. It supports multiple devices and interacts closely with the `SDNFlowTable` and `SDNController`, enabling a modular and extendable switch architecture.

ARP Handling

We implemented ARP handling with the help of openflow module within the switch. We check if the protocol is ARP, then we copy the packet and extract the source IP, source MAC, and destination IP. We get the MAC of the switch which the controller maps to the destination MAC. We build the ARP reply with the IPs and MACs and send it back to the source. Once the source gets the

reply it is able to send the UDP packets.

SDN Flow Table

The SDN flow table serves as the core data structure within each switch to manage forwarding behavior. A flow entry in this table is defined by a combination of source and destination IP addresses, ports, and transport-layer protocol. Each entry also tracks usage statistics including packet and byte counts. When a new rule is added, the `AddFlowEntry` method creates a structured entry containing all these fields and links it to a specific output device. The table supports flow lookup using header fields via the `FindMatchingFlow` function and supports deletion and modification of rules through dedicated APIs. Flow statistics are updated in real-time using the `UpdateFlowStats` method, and complete table contents can be retrieved or printed for diagnostics. This design allows for scalable packet forwarding and real-time monitoring within the switch.

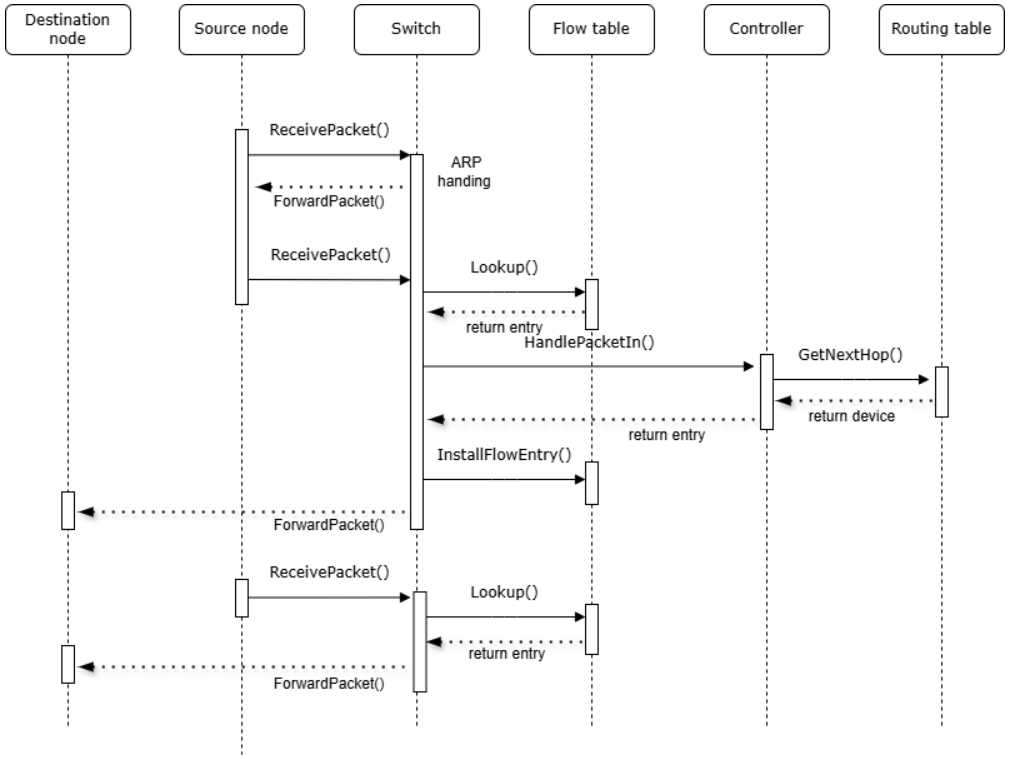

Figure 2: Sequence diagram for 2 packet transfer with 2 nodes and 1 switch

Evaluation and Results

We evaluated the correctness and performance of our implementation using PCAP packet traces and log analysis. Our controller and switch logic performed as expected, with flow rules dynamically installed based on network topology. ARP packets were handled correctly, with replies and resolutions observed in the PCAP outputs. We validated that data plane behavior matched control decisions by comparing flow entry installations with actual packet transmission observed in the traces. The packet counts and byte counts in our logs aligned with the values tracked in the SDNFlowTable statistics, confirming accurate per-flow monitoring. In multi-packet scenarios, the PCAP output confirmed reuse of installed rules, with packet forwarding occurring without additional controller intervention. Network paths were computed accurately, including multi-hop forwarding, and all data packets reached their destinations through paths derived by the controller. We created two different example scripts to demonstrate SDN behavior in these topologies.

One Switch

A basic setup with two nodes (N1 and N2) and a single switch (S1) was used. When N1 sends a packet to N2, switch S1 receives the packet. If no flow entry exists, it queries the controller. The controller computes the path and responds with a flow rule. The packet is then forwarded, and the flow table is updated accordingly.

```
========== SDN Controller Detailed Routing Information ==========
Full Routing Table (Source → Destination → Next Hop):
----------------------------------------------------------
From Node 0:
  → To Node 1: Next hop is Node 2
  → To Node 2: Next hop is Node 2

From Node 1:
  → To Node 0: Next hop is Node 2
  → To Node 2: Next hop is Node 2

From Node 2:
  → To Node 0: Next hop is Node 0
  → To Node 1: Next hop is Node 1

============================================================
```

Figure 3: Routing table for a network with 2 nodes and 1 switch

```
SDNSwitch: SDNSwitch created
SDNFlowTable: SDNFlowTable created
SDNSwitch: Packet received at SDN switch
SDNSwitch: ARP Request from IP 10.1.1.1 asking for 10.1.1.2
SDNSwitch: ARP Reply sent to 10.1.1.1 for target 10.1.1.2
SDNSwitch: Packet received at SDN switch
SDNFlowTable: Looking up: 10.1.1.1:49153 -> 10.1.1.2:9 using UDP
SDNSwitch: No flow match
SDNController: src=10.1.1.1 dst=10.1.1.2 UDP srcPort=49153 dstPort=9
SDNFlowTable: New Flow entry added
SDNSwitch: forwarding from 10.1.1.1 to 10.1.1.2 with mac 02-06-00:00:00:00:00:03
=== SDN Flow Table Statistics ===
Flow: 10.1.1.1:49153 -> 10.1.1.2:9 (UDP)
  Packets: 1
  Bytes:   558
```

Figure 4: ns3 output for 1 packet transmission for a network with 2 nodes and 1 switch

```
SDNSwitch: SDNSwitch created
SDNFlowTable: SDNFlowTable created
SDNSwitch: Packet received at SDN switch
SDNSwitch: ARP Request from IP 10.1.1.1 asking for 10.1.1.2
SDNSwitch: ARP Reply sent to 10.1.1.1 for target 10.1.1.2
SDNSwitch: Packet received at SDN switch
SDNFlowTable: Looking up: 10.1.1.1:49153 -> 10.1.1.2:9 using UDP
SDNSwitch: No flow match
SDNController: src=10.1.1.1 dst=10.1.1.2 UDP srcPort=49153 dstPort=9
SDNFlowTable: New Flow entry added
SDNSwitch: forwarding from 10.1.1.1 to 10.1.1.2 with mac 02-06-00:00:00:00:00:03
SDNSwitch: Packet received at SDN switch
SDNFlowTable: Looking up: 10.1.1.1:49153 -> 10.1.1.2:9 using UDP
SDNSwitch: Flow matched. Forwarding...
SDNSwitch: forwarding from 10.1.1.1 to 10.1.1.2 with mac 02-06-00:00:00:00:00:03
=== SDN Flow Table Statistics ===
Flow: 10.1.1.1:49153 -> 10.1.1.2:9 (UDP)
  Packets: 2
  Bytes:   1116
```

Figure 5: ns3 output for 2 packets transmission for a network with 2 nodes and 1 switch

Two Switches

We extended the setup to include multiple switches (S1 and S2). The controller computes multi-hop routes and installs rules across devices. This verifies end-to-end communication via multiple hops, proper ARP resolution through intermediate devices, and reuse of installed flow rules for future packets.

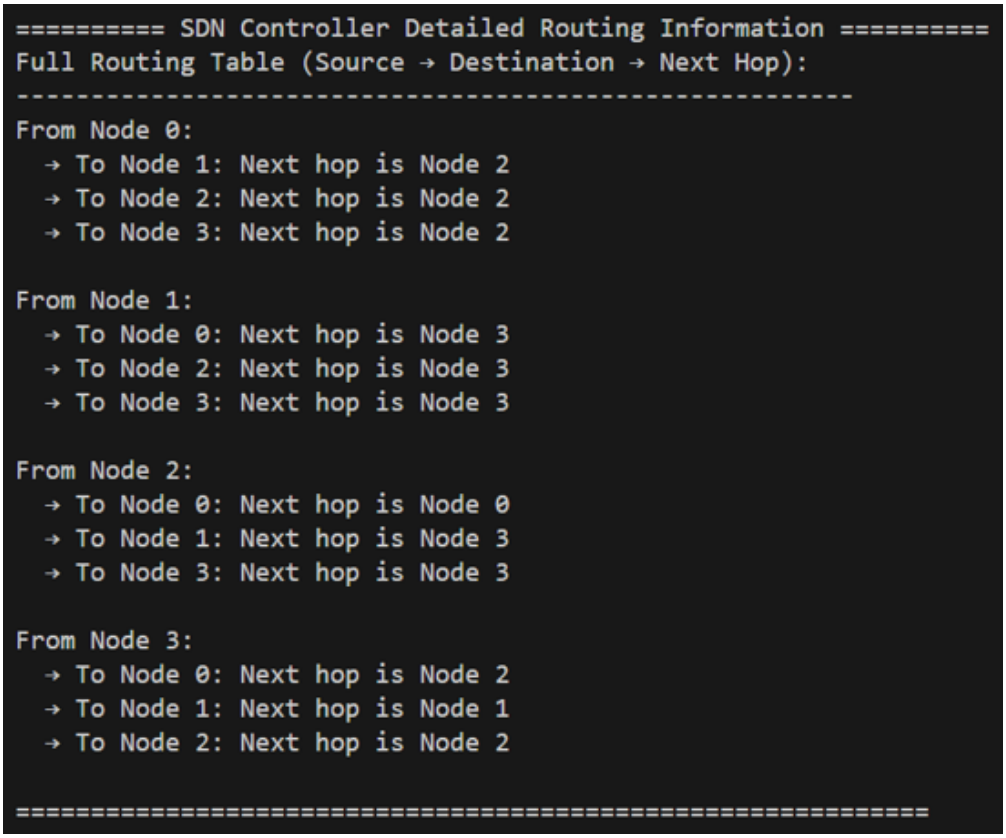

Figure 6: Routing table for a network with 2 nodes and 2 switches

Challenges and Contributions

A key challenge was the lack of a built-in SDN controller-switch interface in NS-3. Additionally, accurate MAC resolution and ensuring flow persistence across simulations proved to be non-trivial. The final challenge was creating complex topologies. We realized that if a switch had more nodes connected to it our topology was returning incorrect MAC addresses and the forwarding was not working correctly. We decided to stick with linear topologies to demonstrate SDN module capabilities and the future potential of it with this fix. Despite these challenges, we developed a custom

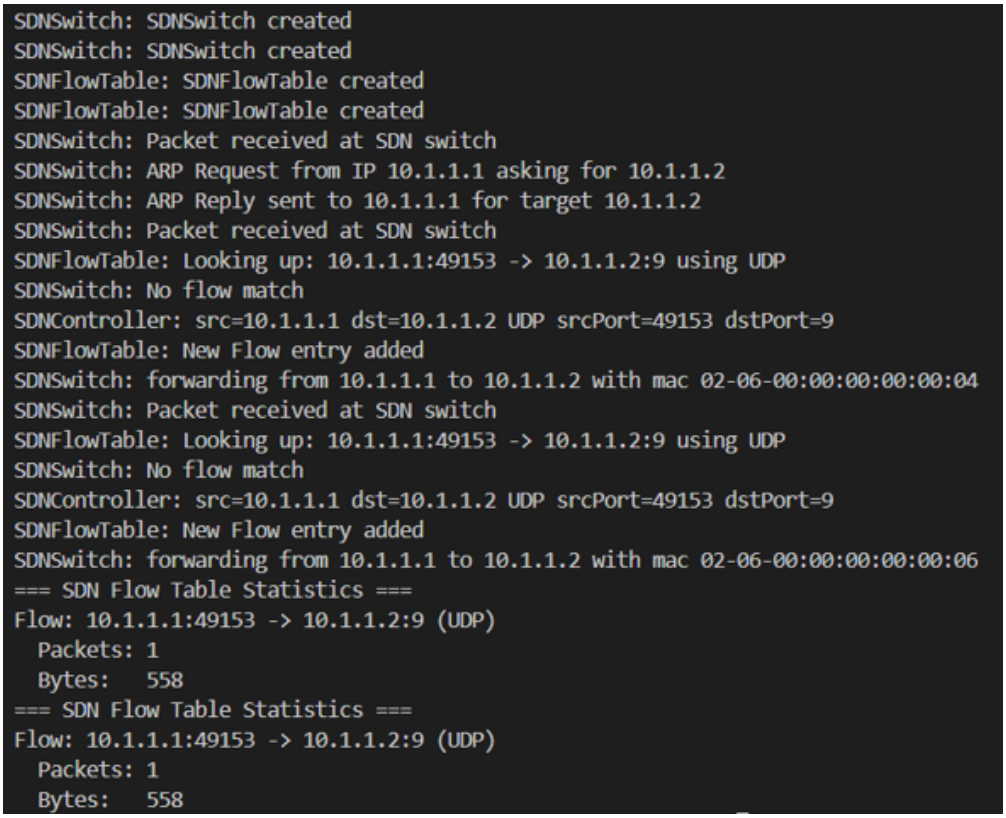

Figure 7: ns3 output for 1 packet transmission for a network with 2 nodes and 2 switches

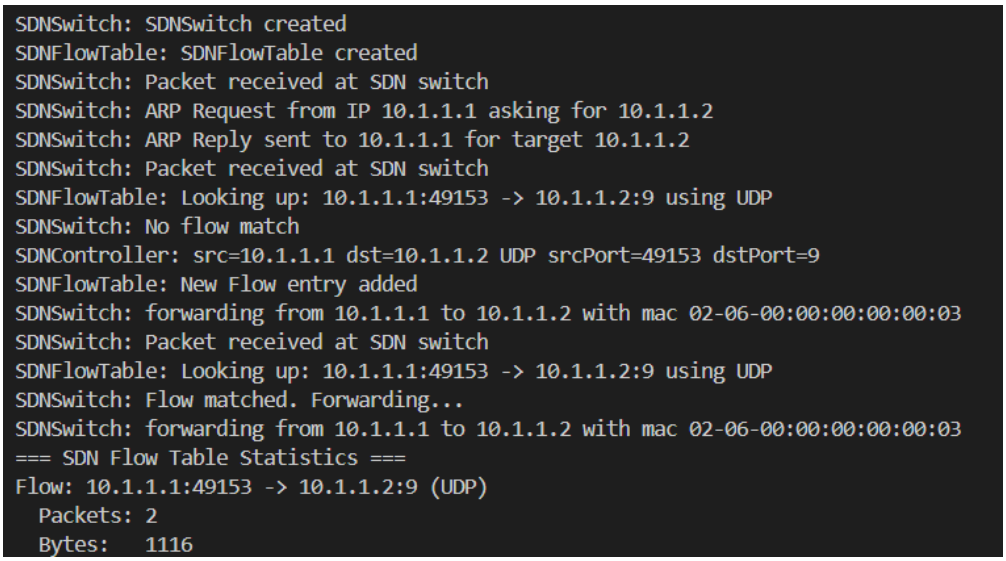

Figure 8: ns3 output for 2 packets transmission for a network with 2 nodes and 2 switches

SDN controller and switch in NS-3. We integrated ARP handling within the switch and enabled flow rule installation. The two example scripts are simple enough for users to understand the SDN module and test out the basic functionalities.

Future Work

One of the key future directions for this project is to expand the SDN implementation to support a broader range of link types. Currently, our work has focused on CSMA connections, but extending compatibility to point to point and wireless technologies such as WiFi and LTE would greatly enhance the applicability of the simulation framework. Furthermore, implementing hierarchical or distributed control planes can improve the scalability of the SDN architecture. This would allow the controller logic to be partitioned or delegated across tiers, enabling better load balancing and faster failover in large or complex network topologies.

Another promising avenue is to improve the realism and flexibility of simulations by integrating RESTful APIs into the controller framework. This would allow external systems or scripts to dynamically modify flow rules, observe network state, or inject control commands during runtime. Additionally, enriching the routing logic with features such as QoS-awareness and traffic prioritization would enable more comprehensive evaluations of advanced SDN applications.

Conclusion

We successfully implemented an end-to-end SDN simulation environment in NS-3. By introducing core SDN components (controller, switch, and flow table) we established a working foundation for programmable networking within the simulator. Our approach enabled dynamic route installation, ARP resolution, and accurate flow handling. This system can be extended for future experiments in SDN research, such as traffic engineering, multi-domain control, and machine learning-based routing decisions. We have included a README file in our code for instructions to run our sdn module successfully in an ns-3 environment.