An SDN Northbound Application for Adaptive Routing in ONOS

CS 331 - Computer Networks

Harshith (22110283) | Naga Bhuvith (22110163) | Sriman Reddy (22110124)
Pavan Deekshith (22110190) | Banavath Diraj Naik (22110044)

Project Introduction

What is SDN?

Software-Defined Networking (SDN) revolutionizes networks by separating the Control Plane from the Data Plane.

Our Project

We built a Java-based Northbound Application for the ONOS controller, an open-source brain for managing the network.

The Goal

To implement an adaptive routing algorithm that dynamically reroutes traffic based on real-time link congestion.

Tools & Technology

ONOS Controller

The open-source SDN controller software that acts as the network's "brain."

Mininet-IP

A network emulator used to create our virtual network of hosts and switches.

Open vSwitch (OVS)

The software-based virtual switch that supports the OpenFlow protocol, allowing ONOS to manage it.

Static vs. Adaptive Routing

Static Routing (The Baseline)

Our baseline calculates the path once (shortest hops) and never re-evaluates it.

- Path is "locked in" for the flow's duration.
- Completely **unaware** of network congestion.
- Predictable, but inefficient.

Adaptive Routing (Our App)

Our application continuously monitors the network and reroutes traffic.

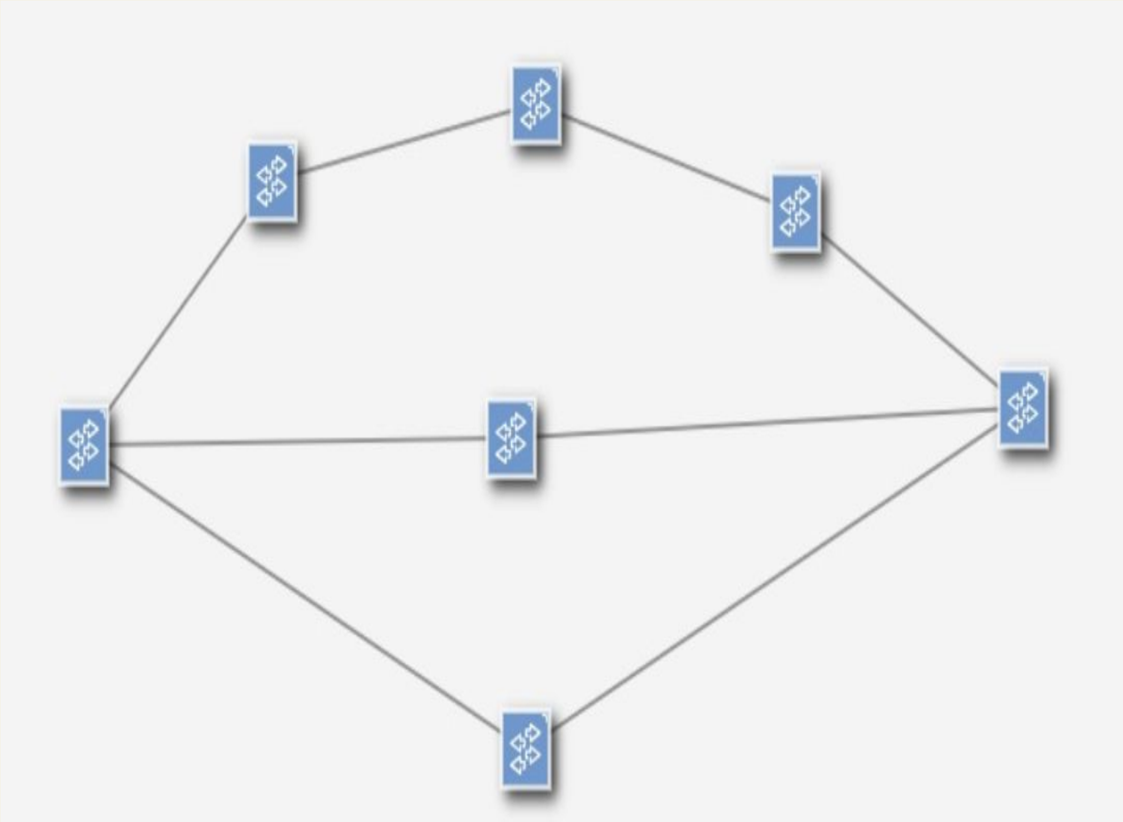
- Monitors link load **every second**.
- **Proactively** triggers new path calculations if congestion is detected.
- Finds the "cheapest" (least congested) path.

Network Topology

The 7-Switch, 2-Host Network

We created this specific topology in Mininet-IP for one crucial reason: **Rerouting**.

- It provides a primary, short path (e.g., 4 hops) between the two hosts.
- It also provides multiple longer, alternative paths (e.g., 4 or 6 hops).
- This rerouting is what our algorithm exploits. A simple linear topology would offer no choices.



Pseudo Code of Adaptive routing

Algorithm 1 Part 1: Reactive Packet-In Handling

```
procedure onPacketIn(packet)
                                    ▷ A switch doesn't know where to send a packet
    source_host ← packet.getSource()
    destination_host ← packet.getDestination()
                                    ▷ Find the best path RIGHT NOW based on current link costs
    best_path ← calculateDijkstraPath(source_host, destination_host)
                                    ▷ Install rules in all switches on that path
    installFlowRules(best_path, source_host, destination_host)
                                    ▷ Send the original packet on its way
    sendPacketOut(packet, best_path)
end procedure
```

Algorithm 2 Part 2: Proactive Monitoring Loop

```
procedure monitorNetwork_Every_1_Second
                                    ▷ 1. Update all link costs
    for all link IN network do
        current_rate ← getLinkTrafficRate(link)
        utilization ← current_rate / link_capacity
                                    ▷ The dynamic cost formula:
        link.cost ← ALPHA + (BETA × utilization)
    end for
                                    ▷ 2. Check all active flows for congestion
    for all flow IN active_flows do
        is_congested ← false
        for all link IN flow.current_path do
            if link.utilization > 0.70 then      ▷ 70% utilization threshold
                is_congested ← true
                break
            end if
        end for
                                    ▷ 3. Reroute if congestion is detected
        if is_congested then
            log("Path is congested! Finding a new one.")
            new_path ← calculateDijkstraPath(flow.source, flow.destination)
            if new_path ≠ flow.current_path then
                                    ▷ Install new rules and remove the old ones
                updateFlowRules(flow, new_path)
            end if
        end if
    end for
end procedure
```

The Core Algorithm: Finding the "Cheapest" Path using Dijkstra's

The Cost Function :

$$Cost = \text{ALPHA} + (\text{BETA} \times \text{Utilization})$$

ALPHA (0.35): The base hop cost. Ensures the shortest path is chosen if all links are clear.

BETA (25.0): The congestion penalty. Heavily punishes busy links.

Link Utilization is the measure of how busy a network link is, typically expressed as a percentage of its total maximum capacity (e.g., 60 Mbits/sec on a 100 Mbits/sec link is 60% utilization).

A clear 4-hop path (Cost: 1.4) is "cheaper" than a 2-hop path with 50% congestion (Cost: 13.2).

Case 1: Baseline (Ideal Network)

Setup & Analysis

An "ideal" network with no added delays and no queue limits.

- **Static Latency:** 0.104 ms
- **Adaptive Latency:** 1.633 ms
- **Packet Loss:** 0% for both

Analysis: This result is expected. In a "perfect" network, congestion has no effect. The adaptive route's overhead (polling, calculating) makes it slightly slower, providing no benefit.

Static

```
--- 10.0.0.2 ping statistics ---
45 packets transmitted, 45 received, 0% packet loss, time 45052ms
rtt min/avg/max/mdev = 0.026/0.104/0.329/0.049 ms
```

Adaptive

```
--- 10.0.0.2 ping statistics ---
45 packets transmitted, 45 received, 0% packet loss, time 44233ms
rtt min/avg/max/mdev = 0.514/1.633/4.118/0.885 ms
```

Case 2: With Switch Processing Delay

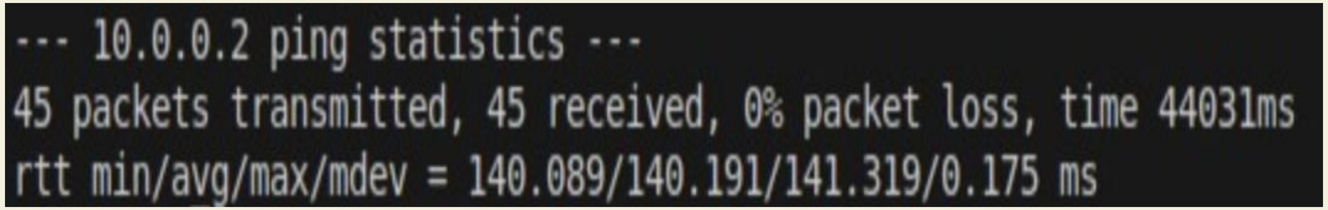
Setup & Analysis

We added a processing delay to each switch to simulate a more realistic network.

- **Static Latency: 140.191 ms**
- **Adaptive Latency: 122.728 ms**

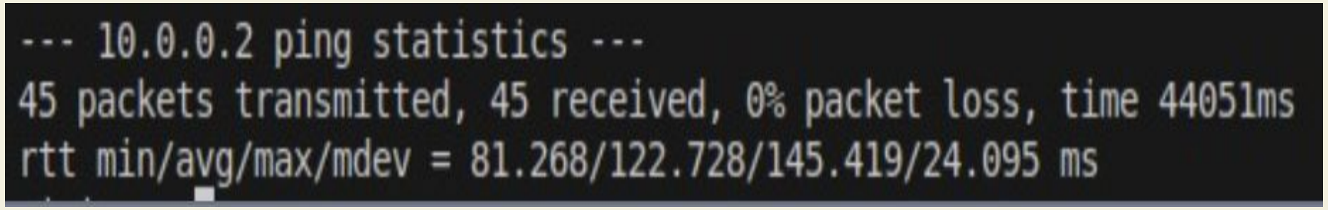
Analysis: The benefit is now clear. The static route finds a shortest path, stacking up delays. The adaptive app finds a longer but less busy path, which is **12.5% faster** overall.

Static



```
--- 10.0.0.2 ping statistics ---
45 packets transmitted, 45 received, 0% packet loss, time 44031ms
rtt min/avg/max/mdev = 140.089/140.191/141.319/0.175 ms
```

Adaptive



```
--- 10.0.0.2 ping statistics ---
45 packets transmitted, 45 received, 0% packet loss, time 44051ms
rtt min/avg/max/mdev = 81.268/122.728/145.419/24.095 ms
```

Case 3: Realistic Network (Delay + Limited Buffers)

The Key Trade-Off

This test adds limited buffers, meaning switches will **drop packets** under heavy congestion.

Static

```
[  1] Server Report:
[ ID] Interval       Transfer     Bandwidth         Jitter   Lost/Total Datagrams
[  1] 0.0000-30.0350 sec  23.4 MBytes  6.54 Mbits/sec   0.051 ms 10044/26754 (38
%)
```

Adaptive

```
[  1] Server Report:
[ ID] Interval       Transfer     Bandwidth         Jitter   Lost/Total Datagrams
[  1] 0.0000-30.0005 sec  28.2 MBytes  7.87 Mbits/sec   0.181 ms 6672/26753 (25%
)
```

Static Route

- **Packet Loss: 38%**
- Throughput: 6.54 Mbits/sec
- Jitter: 0.051 ms

Adaptive Route

- **Packet Loss: 25%** (34.2% reduction)
- Throughput: 7.87 Mbits/sec (20.33% increase)
- Jitter: 0.181 ms (a negligible trade-off)

Conclusion

SDN-based adaptive routing outperforms traditional static routing under realistic network conditions.

- Adaptive control intelligently balances load by selecting less congested paths proving that the shortest path isn’t always the fastest.
- Results Summary:
  - Ideal network: Static routing sufficient; adaptive adds no benefit.
  - Realistic delays: 12.5% latency reduction using adaptive routing.
  - Constrained buffers: 34.2% fewer lost packets and 20.33% higher throughput.
- SDN abstraction enables centralized intelligence → proactive, optimized resource management.
- Trade-off: Slight jitter increase (0.051 → 0.181 ms) due to path switching — acceptable for better throughput and reliability.

References

- Open Networking Foundation. (2025). *ONOS: Open Network Operating System*. https://opennetworking.org/onos/
- Lantz, B., Heller, B., & McKeown, N. (2010). *A network in a laptop: rapid prototyping for software-defined networks*.
- McKeown, N., et al. (2008). *OpenFlow: enabling innovation in campus networks*.
- Open Networking Foundation (ONF), *ONOS OnePing Application Repository*, GitHub. Link
- Vera Martínez, S., and Torra, A. (2023). *Reinforcement Learning-Based Routing in SDN Networks*. Bachelor Thesis, Universitat Politècnica de Catalunya. Available at: Link
- Singh, S. (2025). *A New Dynamic Routing Approach for Software Defined Network*. *International Journal on Recent and Innovation Trends in Computing and Communication (IJRITCC)*. Available at: http://www.ijritcc.org
- Irfan, T., Hakimi, R., Risdianto, A. C., and Mulyana, E. *ONOS Intent Path Forwarding using Dijkstra Algorithm*. Published in *Proceedings of International Conference on Network and Communication Technologies*, 2024.

Questions?

Thank you for your attention.