ARTICLE

Intent-based networking and reactive forwarding performance comparison on resolving multi-link failure

Galura Muhammad Suranegara*, Salwa Tasya Fathira Purba, Endah Setyowati, and Diky Zakaria

Universitas Pendidikan Indonesia, Bandung, Indonesia

ABSTRACT

This study adopted a different approach by leveraging forwarding mechanisms. Software-defined networking (SDN) had two scenarios for handling link failures: intent-based forwarding and reactive forwarding. This research focused on comparing intent-based forwarding and reactive forwarding in SDN-based networks. The study tested both forwarding mechanisms on three different topologies and deliberately disrupted one or more links to evaluate the failure resolution capabilities. The results indicated that intent-based forwarding excelled in recovery time, especially in topology C used in this study, for both single-link and multi-link scenarios. Therefore, for applications that required high-reliability service, intent-based forwarding was more recommended, despite having a more complex process compared to reactive forwarding.

INTRODUCTION

Software-defined network (SDN) is a relatively new paradigm that is assumed to be an upgraded version of the implements the separation concept between the control plane and the data plane. Unlike traditional computer networks that implement the control plane and data plane within the same device, making it difficult to configure. Therefore, SDN is assumed to make it easier for network developers to configure and conduct experiments on new networks or protocols (Thirupathi et al. 2019). SDN could be managed through a controller for controlling the traffic operation of some systems.

The controller in the context of SDN is software that acts as the brain of the network, running on a server. The controller is responsible for managing policies and traffic flows across the network by communicating with the hardware infrastructure through an Application Programming Interface (API) (Zhu et al. 2020). In the SDN model, the controller receives instructions from applications and translates them into commands that can be understood by the hardware.

The presence of an SDN controller could decrease manual configuration for each device and simplify configuration for complex topologies (Zhu et al. 2020). The integrated system between SDN and controller could support network configuration and enable users to perform real-time monitoring, minimizing network routing steps and protocol switching. One of the most used SDN controllers is the Open Network Operating System (ONOS). ONOS is an open-source SDN under the Apache 2.0 license, designed to provide high scalability, with scale-out capabilities, and to demonstrate good

*Corresponding author
Email Address: galurams@upi.edu
Date received: 10 September 2024
Date revised: 11 November 2024
Date accepted: 30 December 2024
DOI: https://doi.org/10.54645/202417SupOMK-17

KEYWORDS

Citizen science project, Digital encyclopedia, Mammals, Species literacy

performance in a network program. The specific target audience for ONOS is service providers and mission-critical networks (Akbar and Basuki 2022).

Although SDN-based networks are an upgraded version of traditional networks, they are not free from disruptions, such as link failures. Link failures can occur due to unbalanced network loads (Setiawan and Farosh 2023). High traffic density in a system also increases the possibility of link failure during packet transmission. Therefore, a system could be considered as ideal if it has both traffic management mechanisms and failure management scenarios (Alsaeedi et al. 2019). Therefore, addressing link failures in SDN networks is crucial, as it affects the reliability and performance of the network.

This issue can occur in any network, even in SDN-based networks. Researchers around the world have tried to address this problem, each with its weaknesses and limitations. Some studies tackle the link failure problem by adjusting the placement of SDN controllers. In addition, link failures have also been addressed by using middleboxes as a traffic engineering-based solution (Ibrahim et al. 2023). This study employs a different approach by utilizing intent-based and reactive forwarding techniques. SDN has two standard forwarding mechanisms that provide failure management and traffic management mechanisms: proactive forwarding and reactive forwarding. ONOS controller also implements intent-based networking to develop the reactive forwarding method, known as intent-based networking (Abbas et al. 2021). Intent-based networking includes several types, such as host-to-host intent, point-to-point intent, multipoint-to-single-point, and others (Monika et al. 2020).

Unlike proactive forwarding, in a reactive forwarding scenario, alternative paths are only created if there is a request for a new entry from the user. The decision to forward a packet is made when a series of new data packets is received by the switch (Akbar and Basuki 2022). The switch sends a copy of the header of the packet to the controller, which then installs rules to redirect the switch in forwarding the next packets on that flow. This allows for flexibility in flow redirection policies without needing to know the initial traffic. Additional input flows can only be added if necessary to reduce the load on the switch (Bianco et al. 2017). Previous research regarding the link failure recovery mechanism in SDN, focused on rerouting techniques to find the response to a single link failure through Fast Failover (FF) (Wang et al. 2023; Petale 2020). Only a few studies describe the performance analysis results of the routing mechanisms provided by the ONOS Controller. Therefore, we conducted this research to assist the developers in determining the services that should be used in the systems they build. This research aims to analyze the performance of reactive forwarding and host-to-host intent-based forwarding in handling link failures and having low recovery times. The reliability in handling link failures was evaluated based on the time taken by the forwarding methods to recover and continue the packet transmission after one of the links failed.

MATERIALS AND METHODS

This research employed a quantitative method with a comparative approach. The research obtained quantitative data in the form of calculations for QoS parameters and recovery time parameters. The obtained data were compared between the two variables: reactive forwarding and intent-based forwarding. The results of these calculations and comparisons served as a benchmark to determine which mechanism exhibited the highest performance and scalability. The testing scenario is shown in Figure 1.

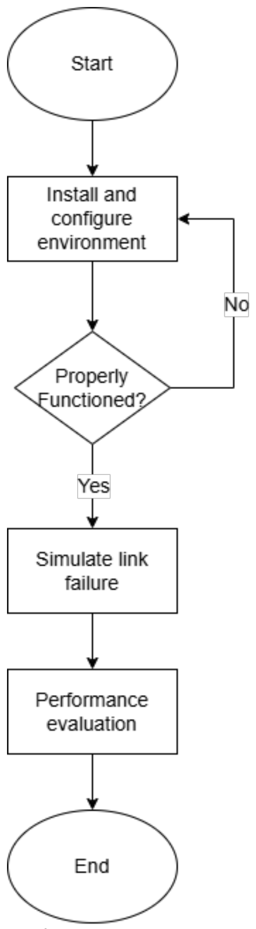

Figure 1: Testing scenario

After configuring the system environment, it is necessary to check whether the system is functioning properly. Subsequently, a link failure scenario can be executed on one or more links used in the topology during data transmission. This yielded performance data for a forwarding mechanism in handling link failures, which then be analyzed and compared.

The testing was conducted on a PC with the following specifications:

(i) Core Processing Unit (CPU): Ryzen 7 5800x

(ii) Random Access Memory (RAM): 32 GB

(iii) Network Interface Card (NIC): 1 Gbps

A system was built under the Ubuntu Linux Server operating system version 16.04.3 (Xenial). This system included Java JDK 8, ONOS Controller version 1.15.0, Mininet version 2.0.0, Iperf3 version 3.0, and Wireshark version 4.4.0. Java JDK 8 was utilized to develop, compile, and run Java applications, providing essential tools such as the compiler, runtime environment, and libraries. The ONOS Controller version 1.15.0 was tested for stability and compatibility in running intent-based forwarding. Mininet version 2.0.0 served as an emulator to simulate the network topology. Iperf3 version 3.0 was employed to measure network bandwidth and performance by testing the throughput between a server and a client. Finally, Wireshark version 4.4.0 was used for capturing and inspecting data packets in real-time to diagnose network issues and monitor traffic.

In this research, the entire system was running on Ubuntu Linux Server 16.04.3 (Xenial) operating on a Virtual Machine. To build the system environment, installations of Java 8, Mininet, ONOS controller, Iperf3, and Wireshark were required. The system environment built is shown in Figure 2.

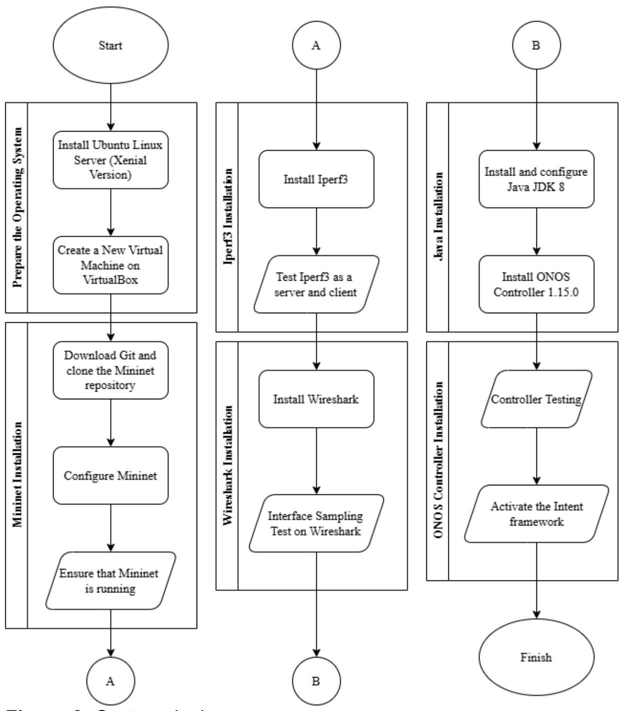
Start
Prepare the Operating System
Install Ubuntu Linux Server (Xenial Version)
Create a New Virtual Machine on VirtualBox
Mininet Installation
Download Git and clone the Mininet repository
Configure Mininet
Ensure that Mininet is running
A
Iperf3 Installation
Install Iperf3
Test Iperf3 as a server and client
Wireshark Installation
Install Wireshark
Interface Sampling Test on Wireshark
B
Java Installation
Install and configure Java JDK 8
Install ONOS Controller 1.15.0
ONOS Controller Installation
Controller Testing
Activate the Intent framework
Finish

Figure 2: System design

Java is necessary to support the operation of the ONOS controller and intent programming. Mininet is an open-source network simulation software used to create virtual network environments. With Mininet, users could create network topologies consisting of virtual hosts, switches, and routers within a Linux operating system. The goal was to test and develop new network applications and protocols without the need for actual physical hardware.

ONOS is an open-source SDN platform under the Apache 2.0 license, designed to provide high scalability, scale-out capabilities, and demonstrate good performance in network programs. The specific targets of ONOS are service providers and mission-critical networks. Iperf3 is a tool for distributing traffic within a network system, making it useful for testing system reliability. Wireshark is also needed as a tool to capture packet transmission and also to assist in monitoring and analyzing packet traffic in the network.

This research used three different custom topologies, each consisting of 2 hosts and a varying number of switches. Topology A had 7 switches arranged as shown in Figure 3, consisting of 9 nodes and 20 links. Topology B had 9 switches arranged as shown in Figure 4, consisting of 11 nodes and 24 links. Topology C had 12 switches arranged as shown in Figure 5, consisting of 14 nodes and 28 links.

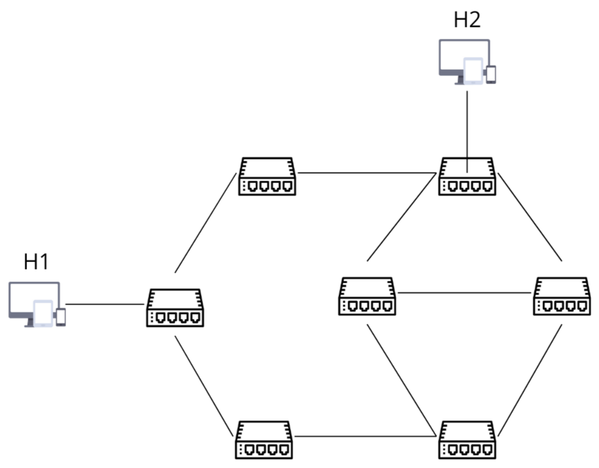

Figure 3: Topology A

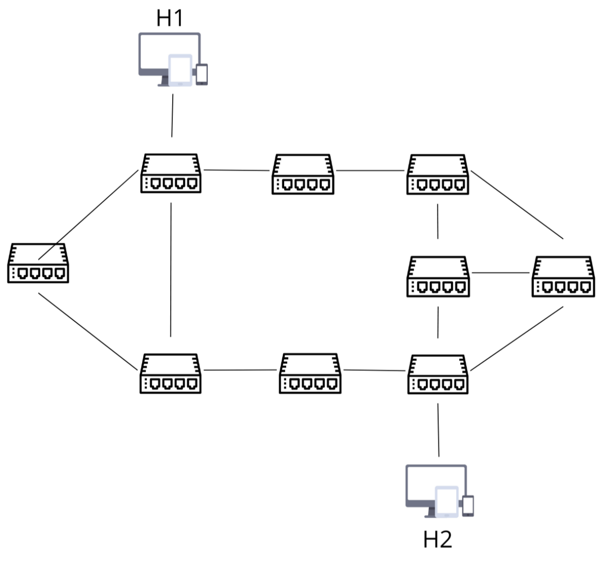

Figure 4: Topology B

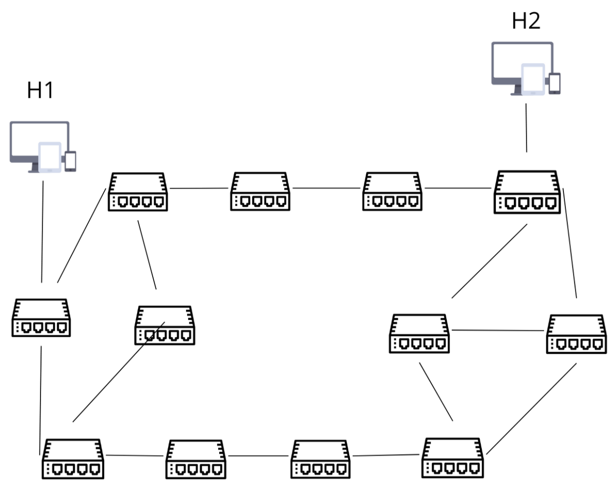

Figure 5: Topology C

The difference in the topology schemes and the use of varying numbers of switches aim~~s~~ to examine the performance of both forwarding mechanisms, namely reactive forwarding and intent-based reactive forwarding, to see if they are affected by the complexity of the topology in managing traffic and handling link failures.

A forwarding mechanism can be considered reliable when it can handle link failures within a built system. Therefore, to measure the reliability of the forwarding mechanism in handling link failures, recovery time testing is used. This monitored the time required by a forwarding mechanism to find an alternative path when a link failure occurs. The testing scenario aimed to evaluate the performance of the forwarding mechanism in handling link failures, as shown in Figure 6.

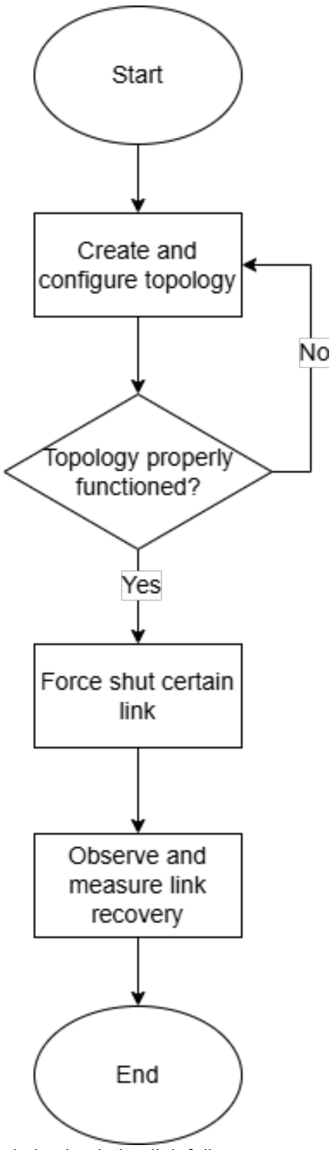
Start
Create and configure topology
No
Topology properly functioned?
Yes
Force shut certain link
Observe and measure link recovery
End

Figure 6: Scenario in simulating link failure

The recovery time testing was conducted using two methods: single-link disconnection and multi-link disconnection, which simultaneously disconnected three links. During testing, ICMP packets were sent through the ping application with 10 repetitions for each topology simulation.

To support packet transmission monitoring during link disconnection, two interfaces in Wireshark were required. The first interface used was the 'any' interface, recording all transmission activities within the topology. The second interface was focused on one specific interface that the packets traversed after the link disconnection.

30 ICMP packets were sent, and at the 10th ICMP packet, the link was disconnected. The forwarding mechanism determined an alternative path, causing a route change. The first incoming packet was visible in the Wireshark application on the previously specified interface. The recovery time was measured by subtracting the time of the last 10th sequence packet sent from the time of the first packet received after the link disconnection.

For the single link disconnection, the link to be disconnected was s1-eth2. Considered in each topology, s1-eth2 was one of the default links traversed during packet transmission. It was important to first observe the packet transmission process under normal conditions and during the link disconnection to determine which alternative path was used. This was done to identify the interface that was used for monitoring the packets that entered first after the link disconnection.

Otherwise, for the multi-link disconnection, three links were disconnected simultaneously. This was intended as further proof to measure the reliability of the forwarding mechanism in finding alternative paths after the links were disconnected. A bash script was used to facilitate the simultaneous disconnection of multiple links.

RESULTS AND DISCUSSION

Single Link Failure

Topology A

In topology A, measurements are conducted in Wireshark through the 'any' interface and the s6-eth2 interface. The s6-eth2 interface is chosen because when the link is disconnected, the first packet passes through switch 6 port 2. After conducting link failure for the intent-based networking and reactive forwarding, the recovery time values are shown in Figure 7.

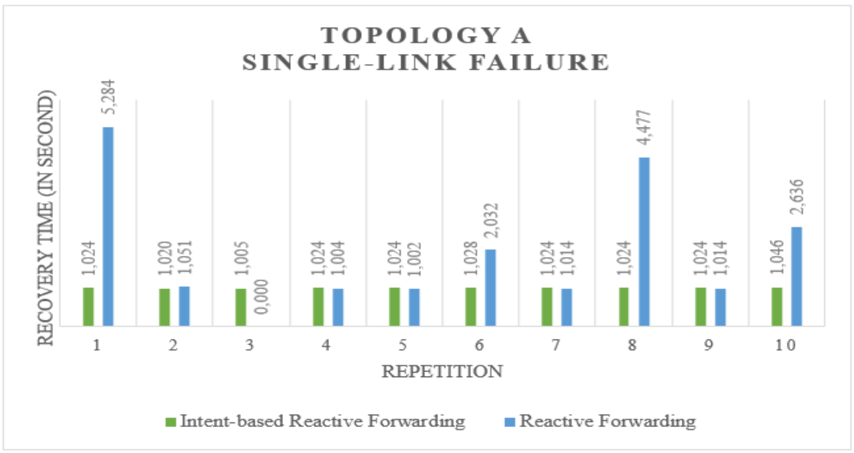

Figure 7: Comparison of recovery time test results for single link failure on topology A

The average recovery time required for intent-based networking to handle link failures is 1,9 seconds, while the average recovery time for reactive forwarding is 1.02 seconds. There is no significant difference between the two forwarding mechanisms; however, the reactive forwarding intent shows more stable performance across repetitions. The recovery times for reactive forwarding are more fluctuating, with the highest recovery time reaching nearly 6 seconds. Reactive forwarding does not require any recovery time when a link disconnection occurs during the 3rd repetition. In contrast, intent-based networking remains stable for around 1 second.

Topology B
In topology B, measurements are conducted in Wireshark through the 'any' interface and the s7-eth2 interface. After conducting link failure for the intent-based networking and reactive forwarding in s1-eth2, the recovery time value overview is shown in Figure 8.

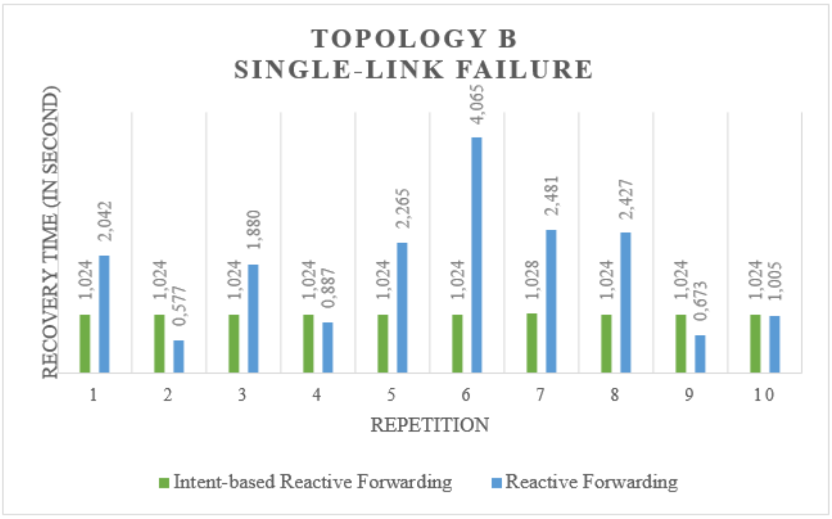

Figure 8: Comparison of recovery time test results for single link failure on topology B

In this topology, intent-based forwarding still performs stable recovery time for each repetition. Intent-based networking still performs stable recovery time on a scale of 1 second for ten repetitions. In the 5th and 6th repetitions, reactive forwarding demonstrates speed in determining alternative paths when a link failure occurs, resulting in a recovery time of less than 1 second.

Meanwhile, intent-based forwarding remains stable for 1 second in each repetition. Indicating a longer time compared to intent-based networking. This longer recovery time may be due to the greater number of nodes and links in the topology compared to topology A. The average recovery time for the reactive forwarding intent in handling link failures is 1.02 seconds, whereas for reactive forwarding it is 42 seconds.

Topology C
In topology C, measurements are conducted in Wireshark through the 'any' interface and the s11-eth2 interface. After conducting link failure for the intent-based forwarding and reactive forwarding in s1-eth2, the recovery time value overview is shown in Figure 9.

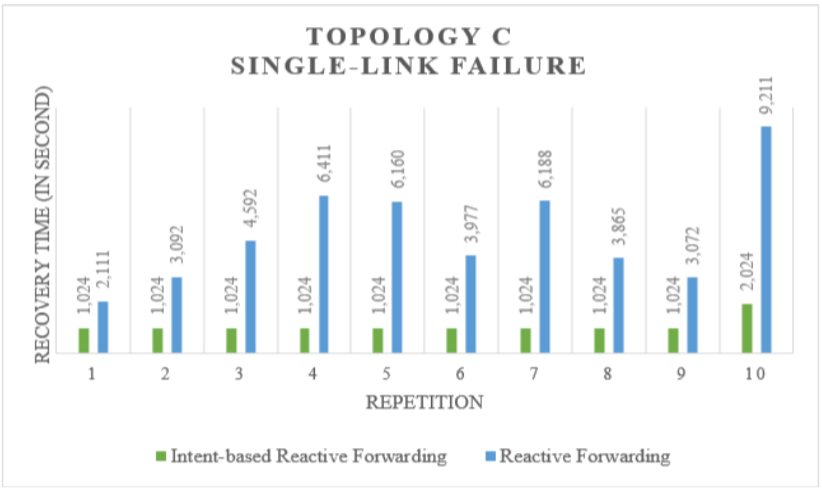

Figure 9: Comparison of recovery time test results for single link failure on topology C

The intent-based forwarding showed a more stable and faster recovery time in each repetition rather than reactive forwarding. Reactive forwarding even shows relatively high recovery times. The average recovery time for the reactive forwarding intent is 11 seconds, whereas, for reactive forwarding, it is 48 seconds.

In handling a fairly complex topology, the reactive forwarding intent still demonstrates stable performance in managing link failures. Even though in the 10th repetition, the recovery time was increased to almost 2 seconds. On the other hand, reactive forwarding takes a longer time to find an alternative path after a link disconnection. Thus, this test shows a significant difference between the performance of reactive forwarding and intent-based forwarding in managing link failures.

Multi-Link Failure

Topology A
In topology A, measurements are conducted in Wireshark through the 'any' interface and the s6-eth3 interface. The s6-eth3 interface is chosen because when the link is disconnected, the first packet passes through switch 6 port 3. The link to be disconnected is s1-eth4, s1-eth2 and s6-eth2. The link to be disconnected is selected based on the alternative paths that were taken if the link is disconnected. Therefore, disconnecting multiple links simultaneously reveals the final paths that were traversed and the recovery time required for each forwarding mechanism. After conducting link failure for the intent-based forwarding and reactive forwarding, the recovery time value overview is shown in Figure 10.

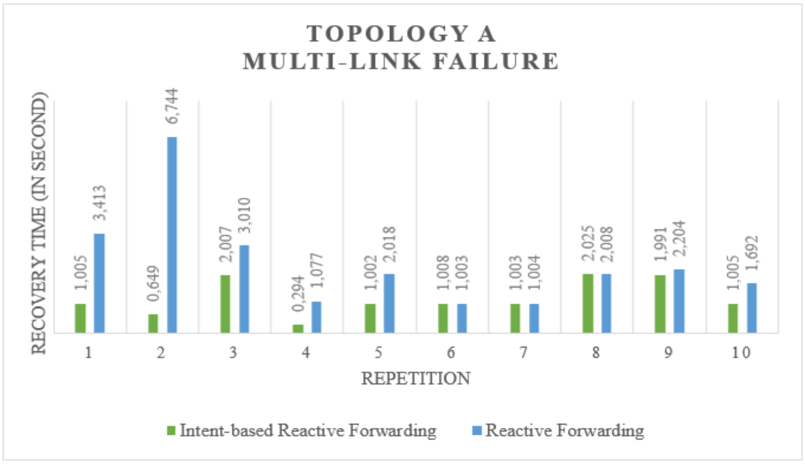

Figure 10: Comparison of recovery time test results for multi-link failure on topology A

In this testing scheme, reactive forwarding showed a relatively high recovery time of 6.7 seconds in the second trial. However, starting from the sixth to the tenth trial, the recovery time became stable and matched the recovery time produced by the intent-based forwarding mechanism. The average recovery time produced by the reactive forwarding mechanism was 2.4 seconds. The intent-based forwarding mechanism produced an average recovery time of 1.1 seconds.

The performance of the intent-based forwarding mechanism is quite stable, only showing high recovery times in a few trials. This could happen because the system is still learning the link failure patterns, allowing it to produce better link failure management.

Topology B
In topology B, measurements are conducted in Wireshark through the 'any' interface and the s6-eth3 interface. The s5-eth4 interface is chosen because when the link is disconnected, the first packet passes through switch 5 port 4. The link to be disconnected is s1-eth2, s1-eth3 and s7-eth1. After conducting link failure for the intent-based forwarding and reactive forwarding, the recovery time value overview is shown in Figure 11.

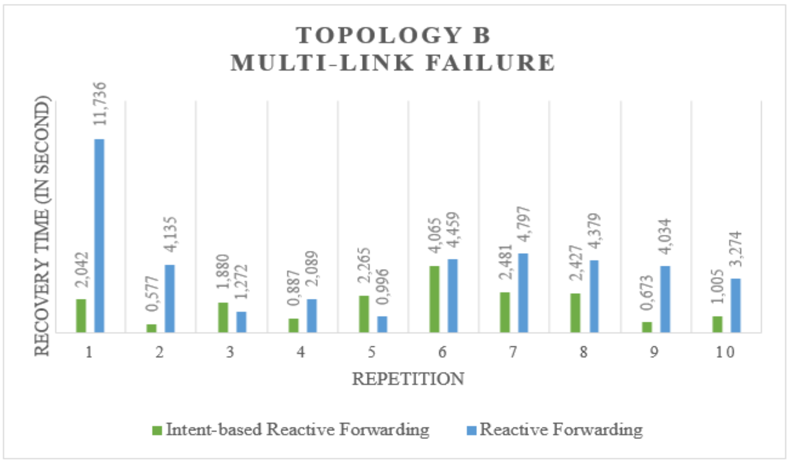

Figure 11: Comparison of recovery time test results for multi-link failure on topology B

As the data shows, the results obtained from both forwarding mechanisms were quite fluctuating. However, reactive forwarding showed a high recovery time of up to 11.7 seconds in the first trial. The highest recovery time produced by the intent-based forwarding mechanism occurred in the sixth repetition, almost reaching 4 seconds. However, in each mechanism, several test results yielded relatively low recovery times below 2 seconds. The average recovery time produced by the intent-based forwarding mechanism was 1.8 seconds, while reactive forwarding produced an average recovery time of 4.1 seconds.

Topology C
In topology C, measurements are conducted in Wireshark

through the 'any' interface and the s9-eth3 interface. The s9-eth3 interface is chosen because when the link is disconnected, the first packet passes through switch 9 port 3. The link to be disconnected is s1-eth3, s2-eth2 and s9-eth2. After conducting link failure for the intent-based forwarding and reactive forwarding, the recovery time value overview is shown in Figure 12.

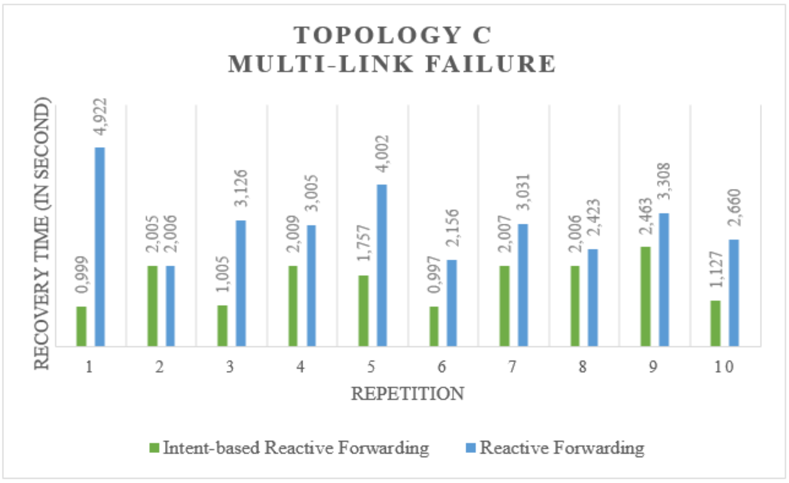

Figure 12: Comparison of recovery time test results for multi-link failure on topology C

Based on the results of this experiment, a significant difference can be observed. The reactive forwarding mechanism requires a longer recovery time compared to the intent-based forwarding mechanism. The average recovery time produced by the intent-based forwarding mechanism was 1.6 seconds, while the reactive forwarding mechanism produced an average recovery time of 3.06 seconds.

This occurs because the reactive forwarding mechanism takes longer to find an alternative path after a link failure. The recovery time results from the intent-based forwarding mechanism are also quite fluctuating. However, it still requires more than 1 second to re-establish an alternative packet delivery path after a link failure.

Discussion

Based on the test results for single link failure, intent-based forwarding demonstrated stable recovery times. Even when the link was disconnected in the relatively complex topology C, it still showed good performance. In contrast, reactive forwarding exhibited more fluctuating recovery times in the trials for each topology. The highest recovery time produced by reactive forwarding showed a significant difference compared to that produced by intent-based forwarding. This difference is clear where the recovery time generated by reactive forwarding is higher than the recovery time generated by intent-based forwarding.

The multi-link failure test is essentially aimed at testing the system's reliability in finding alternative paths when multiple transmission links are simultaneously disconnected. Unlike the single link failure test results where intent-based forwarding showed stable performance, in this test, intent-based forwarding displayed quite varied recovery times. However, the highest recovery time produced did not exceed 4 seconds. The performance of the intent-based forwarding mechanism is quite stable, only showing high recovery times in a few trials. This could happen because the system is still learning the link failure patterns, allowing it to produce better link failure management.

On the other hand, reactive forwarding produced average recovery times of 2.4 seconds, 4.1 seconds, and 3.06 seconds. These results are relatively high compared to the average recovery time produced by the intent-based forwarding mechanism, which is only around 1 second. Thus, it can be concluded that in this test, intent-based forwarding demonstrated better performance compared to reactive forwarding. Essentially, reactive forwarding aims to avoid congestion by distributing traffic evenly across available paths (Wang et al. 2023). As a result, the complexity of the rerouting process increases in complex topologies. In contrast, the intent-based forwarding mechanism exhibits stable performance that is not affected by the complexity of the topology. The time required by intent-based forwarding to find an alternative path is faster compared to the performance shown by reactive forwarding.

CONCLUSION

Based on the testing results for recovery time, intent-based forwarding demonstrates a stable recovery time compared to reactive forwarding. Even when links are disrupted in relatively complex topologies, intent-based forwarding continues to show good performance and significantly reduces the time needed to find alternative paths after a link failure. This is due to the consistency of intents in identifying network resources, namely the hosts and their intent IDs. In contrast, reactive forwarding exhibits more fluctuating recovery times in trials for each topology. Additionally, the performance of reactive forwarding is affected by the complexity of the topology. This occurs because reactive forwarding aims to avoid path congestion by evenly distributing traffic across available routes. As a result, the complexity of the rerouting process increases in complex topologies (Ali et al. 2020). This indicates a clear advantage of using intent-based mechanisms in network services where rapid failure recovery is crucial.

ACKNOWLEDGMENT

We would like to express our sincere gratitude to Universitas Pendidikan Indonesia for its invaluable support in facilitating this research. The resources, academic environment, and encouragement provided by UPI have significantly contributed to the completion of this study. Thank you.

CONFLICT OF INTEREST

The authors declare that there is no conflict of interest.

CONTRIBUTIONS OF INDIVIDUAL AUTHORS

Galura Muhammad Suranegara is the primary contributor, providing grant support and editing the final manuscript. Salwa Tasya Fathira Purba conducted the majority of the experiments, organized and presented the data, and drafted the initial manuscript. Endah Setyowati and Diky Zakaria offered support and provided guidance for this research.

REFERENCES

Abbas, K, Khan, TA, Afaq, M, Song, WC. Network slice lifecycle management for 5G mobile networks: An intent-based networking approach. IEEE Access 2021; 9, 80128-80146.

Akbar, FS, Basuki, A. evaluasi intent-based reactive forwarding dan reactive forwarding pada onos controller untuk pemulihan kegagalan jaringan dalam software defined networking. Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer 2022; 6(12), 5854-5861.

Ali J, Lee GM, Roh BH, Ryu DK, Park G. Software-defined networking approaches for link failure recovery: A survey. Sustainability 2020; 12(10), 4255.

Alsaeedi, M, Mohamad, MM, Al-Roubaiey, AA. Toward adaptive and scalable OpenFlow-SDN flow control: A survey. IEEE Access 2019; 7, 107346-107379.

Bianco A, Giaccone P, Mashayekhi R, Ullio M, Vercellone V. Scalability of ONOS reactive forwarding applications in ISP networks. Computer Communications 2017; 102, 130-138.

Ibrahim, A A, Hashim, F, Sali, A, Noordin, NK, Navaie, K, Fadul, SM. Reliability-aware swarm based multi-objective optimization for controller placement in distributed SDN architecture. Digital Communications and Networks 2023; 10(5), 3.

Monika, P, Negara, RM, Sanjoyo, DD. Performance analysis of software defined network using intent monitor and reroute method on ONOS controller. Bulletin of Electrical Engineering and Informatics 2020; 9(5), 2065-2073.

Petale, S, Thangaraj, J. Link failure recovery mechanism in software defined networks. IEEE Journal on Selected Areas in Communications 2020; 38(7), 1285-1292.

Setiawan Y, Farosh GM. Analisis load balancing round robin dan fault detection pada software defined network berbasis P4. Indonesian Journal of Computer Science 2023; 12(2), 5.

Thirupathi V, Sandeep CH, Kumar N, Kumar PP. A comprehensive review on SDN architecture, applications and major benefits of SDN. International Journal of Advanced Science and Technology 2019; 28(20), 607-614.

Wang, Z, Li, Y, Guan, S. A robust-link controller placement model for large-scale software defined networks. Transactions on Emerging Telecommunications Technologies 2023; 34(6), 4765.

Yin, H, Chen, J. A cross entropy-based approach to controller placement problem with link failures in SDN. Journal of Circuits, Systems and Computers 2023; 32(14), 2350240.

Zhu L, Karim MM, Sharif K, Xu, C Li, F Du X, Guizani, M. SDN controllers: A comprehensive analysis and performance evaluation study. ACM Computing Surveys (CSUR) 2020; 53(6), 1-40.