SCALABILITY IMPROVEMENT OF ACTIVE PROBING FOR FAILURE DETECTION IN LARGE-SCALE SOFTWARE DEFINED NETWORKS USING COMMUNITY DETECTION ALGORITHMS

Firdaus Sahran[1], Nor Badrul Anuar[2]

[1,2]Department of Computer System and Networking, Faculty of Computer Science and Information Technology, University of Malaya, 50603 Kuala Lumpur, Malaysia

Email: firdaussahran@um.edu.my[1], badrul@um.edu.my[2]

ABSTRACT

Software-defined networking (SDN) is a networking approach that separates the network's control plane and data plane to provide better network control. The nature of SDN requires an essential focus on its reliability and availability, which includes failure detection aspects. Several techniques have been proposed to enhance SDN failure detection, including passive and active approaches. However, the Route Inspection (RI) algorithm, an example of the active probing method in failure detection, has scalability issues when deployed in large-scale networks as it generates inefficient and lengthy probe paths. To address this, we propose using Community Detection (CD) algorithms to improve the probe path generation of the RI algorithm, resulting in the 'Failure Detection using Community Detection and probing (FDCD)' framework. With the aim to reduce travel paths, the proposed framework minimises the network topology using CD algorithms to reduce the topology size into smaller communities. Then, the RI algorithm calculates the probe path in the communities instead of the single large topology, thus improving the efficiency of the failure detection mechanism. This paper adopts three CD algorithms: Louvain, Label Propagation (LP), and Girvan-Newman (GN). We evaluate the framework using three well-known network topologies, COST266, PIORO40, and GERMANY50, in Mininet to observe the framework's performance. The results indicate that the proposed approach improves the algorithm's scalability against its baseline approach, improving the average probe round trip time by 66.87% and the average path installation time by 2.67%.

Keywords: Software-Defined Networking, Failure Detection, Active Probing, Community Detection, Scalability

1.0 INTRODUCTION

SDN is a promising network architecture that enables centralised network traffic control and dynamic reconfiguration by separating the control and data plane [1]. In traditional networks, switches and routers make forwarding decisions based on their routing protocols, leading to a distributed control system that is complex to manage and inflexible. Meanwhile, the centralised control plane in SDN and a controller manage network policies and configurations. Due to the essential nature of the applications in which SDN is implemented, however, SDN failures may have serious repercussions. A single network failure may result in a severe interruption, including downtime, data loss, and financial losses. Consequently, companies reliant on SDN spend considerably developing and implementing dependable failure detection techniques. According to a new analysis by MarketsandMarkets, the SDN market will reach $28.8 billion by 2025, propelled by the demand for efficient network infrastructure and rising investment in software-based networking solutions [2]. Consequently, assuring the network's dependability and availability is highly significant.

Ensuring the reliability and availability of networks is essential in SDN, and several techniques are present to achieve this, especially in failure detection. The technique includes data-driven approaches and probing mechanisms, which inhibit different characteristics to detect network failure in SDN. The data-driven approach relies on network statistics and monitoring traffic to detect failures, which is less disruptive to the network performance but suffers from the lack of real-time detection and accuracy [3], [4]. The probing-based approaches combat the issues mentioned through probe packets sent in the network to constantly monitor nodes and links to provide fast detection [5]. However, certain algorithms used to generate the probe path, such as the RI algorithm, lack the scalability and efficiency in large network topologies due to its nature to reach every node and link at least once in a probing cycle [5], [6]. Moreover, recent studies deploying the probing approach for single controller setups were only tested in small-scale networks. One of the solutions for failure detection by Daha et al. proposed CD algorithms by clustering the nodes into smaller network portions [7]. Although the work focused more on recovery actions, we believe the CD algorithms play a more important role in failure detection for large-scale SDNs.

This study proposes adopting CD algorithms to improve probe path generation to address the scalability and efficiency issues. The CD algorithms establish links and nodes within the network topology, reducing the network size into smaller segments. Subsequently, the RI algorithm generates a probe path within the communities instead of the original network size. This approach minimises the probe's hops or travel paths, even at the expense of increasing

probe packets. To evaluate the proposed approach, three well-known network topologies, COST266, PIORO40, and GERMANY50, are selected as datasets for the study. Additionally, three CD algorithms, namely Louvain, LP, and GN, are selected to observe their differences towards community generation. The approach is combined into a framework tested in an emulated environment to evaluate its performance. By adopting the proposed framework, the scalability issue of the RI algorithm in large-scale SDN networks is addressed, resulting in improved efficiency of the failure detection mechanism. The algorithm combination of Louvain and RI improved the average probe round trip time and average path installation time by 2.67%, respectively. The results highlighted that the proposed framework contributes positively towards SDN failure detection.

In essence, the contribution of this paper can be summarised as follows. Firstly, we introduce an improvement to the RI algorithm's scalability for active probing in SDN. The proposed improvement involves adopting CD algorithms that segment the network into smaller sections to shorten the probe path taken by the RI algorithm. Secondly, we propose the FDCD framework to support our solution, consisting of several modules: a) Topology Analyzer, b) Community Calculator, and c) Probe Route Finder. Finally, we tested the framework in Mininet to verify its performance against existing approaches.

We organise the remainder of the paper as follows. Section 2 discusses the related work in the domain and analyses the problem. Next, we present the framework in Section 3 and provide comprehensive interpretations for each framework component in Section 4. Section 5 presents the results and discussions on our experiments' evaluation and validation processes. Finally, Section 6 concludes the paper.

2.0 RELATED WORK

In this section, we discuss the related work in SDN failure detection, including the data-driven and probe-based approaches, and analyse the problem regarding using the RI algorithm for probe path planning.

2.1 SDN Failure Detection

SDN has emerged as a promising paradigm for designing and managing modern computer networks. By decoupling the control plane from the data plane, SDN enables network operators to dynamically control the behaviour of network devices and applications and respond to changing traffic patterns and network conditions in real-time. However, the dynamic nature of SDN networks also poses new challenges for network management, particularly in failure detection and recovery. Failure detection in SDN is a critical task that ensures the proper functioning and availability of the network. It is important to tackle the issues in failure detection, especially link failures, as it prevents further problems such as link crashes, link elimination, timing failure and response failures. The link failures in SDN follow a Poisson distribution that describes a random failure event at a particular time [8]. The cumulative density function and exponential distribution denote a link failure probability before time *t*, calculated using the following equation.

$$p_f(t) = 1 - e^{-\lambda t} \tag{1}$$

The probability density function (P) is a derivative of Equation 1, which is equal to $P_f(t) = \lambda e^{-\lambda t}$. Therefore, on a particular time t, the following equation calculates the reliability in the absence of failure occurring within a time interval $(0, t)$.

$$R(t) = \int_t^{+\infty} \lambda_k e^{-k^{\tau}} d\tau \tag{2}$$

Where λ_k represents the failure rate of K_{th} link in SDN. In general, λ is equal to $\lambda = 1/m$, where m denotes the mean time between failures. Let $\lambda_K > 0$, Equation 2 above can be rewritten by the following.

$$\begin{aligned} R(t) &= \int_t^{+\infty} \lambda_k e^{-k^{\tau}} d\tau \\ &= \lambda_k \frac{1}{-\lambda_k} e^{-\lambda_k \tau} \Big|_t^{+\infty} \\ &= (-1)\left(e^{-\lambda_k(\infty)} - e^{-\lambda_k t}\right) \\ &= e^{-\lambda_k t} \end{aligned} \tag{3}$$

The mathematical formulation shows that the reliability of links decreases over time and, therefore, requires appropriate actions to maintain optimal performance. The actions include proper failure detection techniques, which fall into two categories: data-driven and probe-based.

The data-driven approach is a passive, less-invasive approach to monitoring network traffic and uses statistics to identify abnormal behaviours. Monitoring network traffic provides insight into network conditions and pinpoints abnormality sources or causes. Polling network statistics is an instance of a data-driven approach [4]. The authors developed an adaptive polling technique to monitor fine-grained traffic metrics, resolving issues of accuracy and communication cost overhead. Other data-driven approaches include using rule dependency graphs to construct potential dependencies and clustering rules [3]. These examples rely on existing data or pre-calculated information and encompass various methods, enabling diverse solutions for SDN. Additionally, the availability of existing data leads to a machine learning-based approach, which provides root cause analysis using different algorithms to detect network failures [9]. Although the data-driven approach is flexible due to its non-intrusive nature and a wide variety of available solutions, it has several disadvantages that hinder the requirements of a fast-evolving SDN. The limitations of these solutions come from the lack of immediate detection, accuracy, and scalability. As the approach relies on passively monitoring network traffic, the detection mechanism is less quick and often must be manually tended. Also, network anomalies seen in the data that may indicate a failure sometimes lead to false positives or false negatives, thus reducing the accuracy [10]. The massive amount of data collected in these systems also causes scalability issues for large networks as administrators require more resources to manage them. In order to solve the limitations mentioned before, a more proactive and invasive approach, referred to as the probe-based approach, provides the means to achieve fast, accurate, and scalable failure detection.

The probe-based approach constitutes a method that actively monitors links via probe packets. Unlike the previous data-driven approach, which relies on statistics and pre-calculated information, probes offer fast and real-time failure detection. The approach uses special monitoring packets to traverse each link and node in the network topology in a cycle to detect anomalies or failures. The Bidirectional Forwarding Detection (BFD) is one of the earliest protocols used in network probing. It implements a control and echo mechanism to detect the liveness of links, paths, or nodes between preconfigured endpoints. A link fails in standard per-link BFD deployments when the receiver misses three BFD packets in a row [11], [12]. The BFD packet exchanges occur rapidly within milliseconds, ensuring fast, low latency, and real-time failure detection capabilities. However, due to the nature of the BFD mechanism in very short intervals, generating and processing enormous packets in the network is resource-intensive. Current works in SDN only tested the BFD implementation in a small-scale, five-switch topology [13]. In a larger network, the BFD implementation leads to more resource usage, causing inefficiency and scalability issues.

Several recent works attempted to address scalable and efficient requirements using the active probing approach. Ramtin Aryan et al. presented SDN Spotlight, a real-time anomaly detection framework to detect installation failures, rule conflicts, and network loops [14]. The authors used a single probing packet to reduce resource usage, specifically the ternary content-addressable memory (TCAM). They proposed two different approaches for the detection: Hedge-SDN, which operated around the boundaries of a specific path, and Open-SDN, which used specific switch IDs to hook the probe packets. The authors claimed that the framework does not yield false positives or negatives when detecting loops and forwarding failures. In other studies, authors focused on the deployment of the RI algorithm as a probe path planning algorithm for the probe path solution [5], [6], [15]. The RI algorithm generates the shortest path to traverse the network in a monitoring cycle. In Chan et al., the authors present a failure detection solution utilising the RI algorithm by periodically probing the network using packets generated by the main controller and received by the standby controllers [5]. While deploying multi-controllers provides reliability in the network, the increasing complexity may cause extra overhead in communication and other network resources. Additionally, some deployments may lack the feasibility and cost to set up extra network controllers for failure detection. The studies mentioned above also lack the implementation in larger-scale network topologies to prove the algorithm's scalability in such conditions. The RI algorithm causes a long monitoring path to traverse a large network and may take much time to complete a cycle. Thus, recent efforts focus on the single-controller setup to reduce as much complexity as possible when it comes to network management to have a minimal impact on the failure detection performance.

Daha et al. provide an example of the work that utilises the capabilities of a single-controller setup in a large network environment [7]. The authors introduced the CD concept in link failures, which groups several nodes and links into smaller network portions to make recovery more efficient via the proposed CDRA framework. The framework deployed several CD algorithms, Louvain, Infomap, and GN, to establish network communities in well-known network topologies such as COST266 and Atlanta [16]. However, the authors focus more on recovery and fail to evaluate failure detection aspects.

Based on the related work discussed above, the active probing approach promises to achieve fast, real-time, and scalable failure detection. However, a weakness identified in the RI algorithm presents an issue as it generates a long

probe path in a large network that may affect the probe travel time. Guided by the idea of community detection, we aim to improve the scalability of the RI algorithm for large-scale networks in a single-controller environment.

2.2 Problem Analysis

In active failure detection using the active probes approach, a network monitor injects test traffic into the network alongside normal traffic. Probe packets differ from regular traffic packets as they collect network knowledge, such as switch and link information, rather than delivering the payload to their intended destination. The performance of active probe-based failure detection varies depending on the network size, scale, number of devices, probes, and probe path. The RI algorithm is one of the algorithms used to solve the probe path formulation. The algorithm determines the shortest length of an Eulerian circuit, which requires any route to travel through each vertex and edge at least once before returning to the starting point. When each node or vertex has an even number of degrees, the problem is easy to solve. However, network topologies often have multiple vertices with odd degrees that lack the property of an Eulerian circuit. The circuit needs to be converted into an Eulerian circuit, which involves duplicating edges to ensure the algorithm works properly, as seen in Fig. 1. This phenomenon increases the overall length of the probe path, causing performance issues.

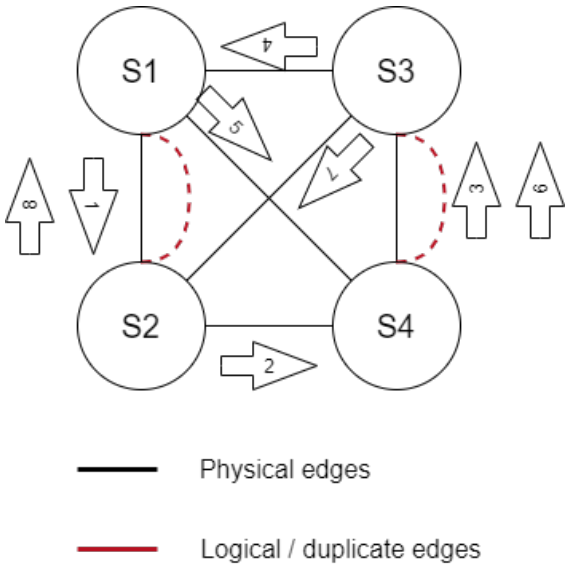

Fig. 1 Eulerian circuit in network topologies

The algorithm's efficiency is limited as the probe packet travels the same nodes and links more than once to complete the circuit. The probe packets must go through increasing switch hops, leading to scalability issues as network size increases (S1→S2→S4→S3→S1→S4→S3→S2→S1). The behaviour can result in a reduction of probe performance for failure detection. Furthermore, probes consume limited bandwidth that would be more beneficial for forwarding normal traffic to its destination.

As such, active probe-based failure detection using the RI algorithm requires further optimisation. Other methods, such as CD techniques, have shown promise in reducing network size, thus improving the scalability performance of the probes for failure detection. This study identifies these issues and presents our contribution to solving the problem, a framework for failure detection using active probing in SDN.

3.0 THE 'FAILURE DETECTION USING COMMUNITY DETECTION AND PROBING' (FDCD) FRAMEWORK

The present work proposes a framework for failure detection using active probes in SDN, incorporating CD algorithms. The framework comprises several modules, each elaborated in detail in this section. The proposed approach aims to enhance the performance of the active probe mechanism, particularly in large-scale topologies, and specifically for detecting link failures between nodes. A summary of the operational characteristics of the framework is provided to offer a clear comprehension of the proposed approach.

3.1 The FDCD Framework

This study proposes the "Failure Detection using Community Detection and probing" (FDCD) framework to address the limitations of existing SDN failure detection approaches. The FDCD framework enables efficient failure detection while minimising network resource usage. The proposed approach utilises active failure detection using CD algorithms and active probes, which aim to improve the efficiency of the probe paths following the RI algorithm. The framework architecture is structured around the three known SDN layers: application, control, and infrastructure, with the proposed modules situated on top of the control layer, as illustrated in Fig. 2. The FDCD framework comprises three modules, namely the Topology Analyzer, Community Calculator, and Probe Route Finder, which are responsible for monitoring the network for failure detection.

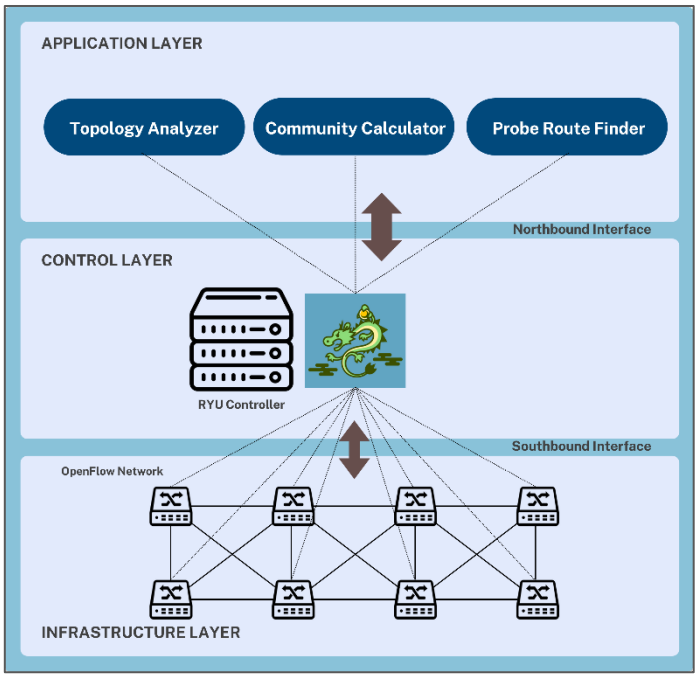

Fig. 2 The FDCD framework for active failure detection using community detection in SDN

The FDCD framework workflow starts with the network topology connected to the Ryu SDN controller, which periodically receives network information through OpenFlow channels. The Topology Analyzer module receives updates on the available network devices, such as the number of nodes and links. The Community Calculator module uses this information to apply CD algorithms to establish network communities or clusters of nodes. The Probe Route Finder module then determines the shortest path for the probe, using the established network communities and running the path-finding algorithm. Finally, the Probe Route Finder module relays the probe path information back to the controller, which inserts the path into the switch flow tables and initiates the failure detection process by injecting the network with specific probe packets. Fig. 3 visualises the workflow. In addition, a summary of the proposed modules in the FDCD framework is provided in Table 1, with further elaboration on the modules and algorithms used in subsequent subsections.

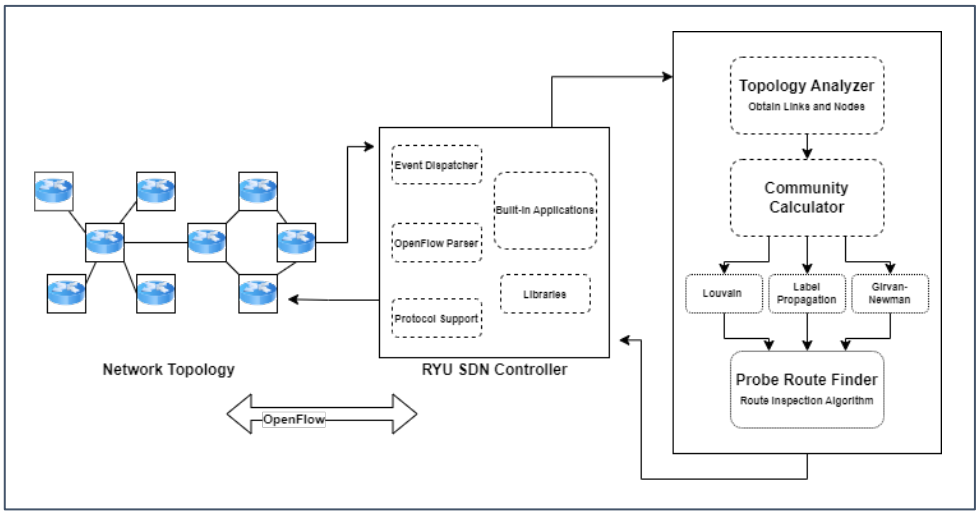

Fig. 3 Workflow of the proposed FDCD framework

Table 1 Summary of the proposed modules in the FDCD framework

Module	Description	Output
Topology Analyzer	Collect network topology information via API to feed the information to the Community Calculator	Set of network nodes and edges in the topology
Community Calculator	Generates community in the topology using the selected CD algorithms	Groups of nodes as communities in the network topology
Probe Route Finder	Calculates the optimal probe path using the RI algorithm Sends the generated probe path to the controller for flow insertion into the forwarding devices	Optimal probe path for the topology

3.2 The Topology Analyzer Module

The initial process of the framework involves collecting network information from the lower-level devices and the controller. Accordingly, the Topology Analyzer module is responsible for this process by fetching information via the REST API. This information builds the topology visualisation for the usage of other modules. The Ryu SDN controller documentation provides a complete guide for using REST APIs to fetch information from the controller. Table 2 summarises examples of several built-in API calls to deliver the network information to the proposed module.

Table 2 The Ryu SDN controller's built-in API calls

Ryu API Module	Description
`ryu.app.rest_topology`	Obtain node and link data
`ryu.app.ws_topology`	Sends notification on the change of link status (up/down)
`ryu.app.ofctl`	A set of API calls to use OpenFlow messages synchronously to obtain the datapath
`ryu.app.ofctl_rest`	A set of REST APIs for stats retrievals and switch updates

The Topology Analyzer module utilises these built-in Ryu applications to retrieve the necessary information to establish knowledge of the current network topology, such as the switch Datapath ID (DPID), hosts MAC addresses, IP addresses, and switch software and hardware information. The module stores the current information and forwards it to the next module, the Community Calculator, for further processing.

3.3 The Community Calculator Module

The Community Calculator module in the proposed FDCD framework utilises three selected CD algorithms to calculate communities based on the network topology information obtained by the Topology Analyzer. CD algorithms assign objects, such as network nodes and links, to smaller communities based on the notion that nodes should be connected to nodes in the same cluster but to a few in other communities [17]. Two primary types of CD clustering techniques are agglomerative and divisive. The agglomerative clustering approach is bottom-up and treats each data point as a cluster before merging into a single set containing all data. In contrast, the divisive clustering approach is top-down. It assumes that the entire data belongs to a single community before splitting it into several clusters recursively until individual data belongs to its group. The Louvain and LP algorithms adopted in this study are based on the agglomerative approach, while GN uses the divisive clustering technique. Furthermore, several key properties differentiate these algorithms, which justifies their selection in the FDCD framework and is discussed in the subsequent subsections.

a) Louvain algorithm

In 2008, researchers introduced the Louvain algorithm as a fast community unfolding method for networks [17]. The technique is an iterative, greedy approach to produce a hierarchy of communities. The modularity approach is the basis for this algorithm, as follows: Given a network graph $G = (V, E)$, where V is the set of nodes and E is the set of edges in the network, each singular node i identifies as a separate community. Then, the algorithm iterates from one node until the modularity gain becomes negligible (as a pre-defined threshold). The algorithm scans the vertices in an arbitrary but pre-defined order for each iteration in a linear fashion. The next step examines all neighbouring communities relative to node i. The algorithm calculates the modularity gain as if the node i were to move to neighbour communities from the original community. The algorithm assigns a neighbouring community that would yield the maximum modularity gain as a new community for i and updates the corresponding data structure that holds the source and target communities. However, if the gains are negative, the node or vertex stays in its current community. Next, the algorithm collapses all vertices of a community to a single 'meta-vertex' by placing an edge from that meta-vertex to itself with an edge weight that is the sum of weights of all the intra-community edges within that community and setting an edge between two meta-vertices with a weight that is equal to the sum of the weights of all the inter-community edges between the corresponding two communities [18]. It results in a condensed graph G' (V', E', ω'), which becomes the input to the next phase. Algorithm 1 below shows the pseudocode of the Louvain algorithm.

Algorithm 1 Louvain Algorithm

Input: Network topology graph, $G = (V, E)$
Output: Network graph community, C

```
Initialise nodes and edges
while i < V do
        Insert i in Community C
        Compute modularity M
        ΔM = M_new – M_old
        if M is positive, then
                continue
        else
                i ++
        end if
end while
for each C, do
        Merge all nodes in C
end for
return C
```

In summary, the Louvain algorithm focuses on modularity calculation, which computes the modularity gain of a node to its neighbours. If the gain is positive, the node moves into the neighbouring community, while a negative modularity

gain results in the node staying in its original community. The end-product from the algorithm operation becomes the input for the next module in the framework pipeline.

b) Label Propagation algorithm

The LP algorithm is a graph-based algorithm used to identify communities within a network [19]. The algorithm utilises the network structure to identify communities and does not require pre-defined information or functions about the community. Initially, each node is assigned a label based on its community members. During each iteration, nodes that are closely linked change their labels based on their neighbours, with the label changes propagating throughout the network until a stopping criterion is met.

Several stopping criteria exist for the algorithm, with the simplest one being to compare the current labelling with the previous iteration. If the current label is the same as the previous, the algorithm stops its process. When there is a tie with two or more labels for a node, the current label of the node takes priority to reduce the likelihood of generating more iterations. The choice of stopping criterion is important since the algorithm is vulnerable to oscillation phenomena, where the label of certain nodes or edges keeps changing with every iteration. The asynchronous approach reduced the oscillation phenomenon, although at the cost of increased time complexity. This study adopts a semi-synchronous algorithm, combining the advantages of synchronous and asynchronous approaches, making it efficient and stable for large networks.

The algorithm inspires the semi-synchronous approach for bipartite networks, which divides each propagation step into two stages [20]. In the first stage, new labels for the blue nodes are computed based on the current labelling of the red nodes, and in the second stage, the label of the red nodes is computed after updating all the blue nodes. Since the network is bipartite, both stages are parallelisable, making the propagation process of blue and red nodes independent. In the case of a tie, the blue labels are given priority, reducing the time required to reach a final consensus. Algorithm 2 explains the algorithm process. In summary, LP propagates labels throughout the network to form communities, using a semi-synchronous approach to eliminate issues related to label oscillations. The established communities serve as inputs for the next module, the probe route finder.

Algorithm 2 Label Propagation Algorithm

Input: Network topology graph, $G = (V, E)$
Output: Network graph community, C

1: Initialise labels at all nodes. For a given node x, $C_x(0) = x$
2: Set $t = 1$
3: **while** $t < T$ **do**
4: Arrange nodes in random order and set to X
5: **for** $x \in X$ **do**
6: $C_x(t) = f(C_{xi1}(t),\ldots, C_{xim}(t); C_{xi(m+1)}(t - 1),\ldots, C_{xik}(t - 1))$ (f returns label occurring with highest frequency among neighbours)
7: **if** label = max neighbour's number, **then**
8: break
9: else
10: $t = t + 1$
11: end **if**
12: end **for**
13: end while
14: return C

c) Girvan-Newman algorithm

The core concept of the GN algorithm stems from the edge information to decide which edges and nodes belong to a community. Unlike other algorithms, the algorithm focuses on eliminating edges with the highest number of shortest paths between nodes passing through them [21]. It identifies communities in a network by removing the edges iteratively.

In a network graph, the 'edge betweenness centrality' defines the extent to which an edge lies on the path between a set of nodes. An edge with a high level of betweenness has a prominent influence on a network due to its control over information passing between nodes. In the GN algorithm, the edge with high betweenness expects to join the communities, which means that the edge is an inter-community edge. Thus, communities formed using the GN algorithm have a low edge betweenness centrality, and a high betweenness edge connects each community.

The algorithm has four main steps to achieve its objective. After initialising the nodes and edges in a graph, an iteration calculates the edge betweenness centrality for each edge. If the edge has the highest betweenness centrality, the edge is removed from a community. After the calculation for each edge is completed, the final output establishes communities in a network graph. Algorithm 3 below describes the process of the algorithm.

Algorithm 3 Girvan-Newman Algorithm

Input: Network topology graph, $G = (V, E)$
Output: Network graph community, C

```
1:  Initialise all nodes and edges
2:  while edge exists, do
3:      Calculate betweenness centrality of edge i = B[i]
4:      if B[i] > max_B then
5:          max_B = B[i]
6:          max_B_edge = i
7:          remove edge
8:      end if
9:  end while
10: return C
```

3.4 The Probe Route Finder Module

The Probe Route Finder module generates probe paths within established communities using information from the previous module, the Community Calculator. This module employs the RI algorithm to plan the probe path inside each community. The advantage of the algorithm is its ability to enter the network at a point, reach all nodes and edges at least once, return to the starting point, and keep the distance travelled to a minimum.

The network graph must have the Eulerian property for the algorithm to function properly. If the graph lacks this property, the algorithm performs processes to convert the graph into an Eulerian graph. The algorithm accomplishes its objective of finding the shortest probe path in seven steps, elaborated below, and the pseudocode in Algorithm 4 describes the entire process. Finally, the module inserts the results of the probe path generation process into the network through the assigned probe generator in each community. The probe packets traverse each community according to the instructions provided by the controller based on the algorithm result.

Step 1: Identify whether the graph is an Eulerian (a closed walk covering every edge once with starting and ending positions the same). If the graph is Eulerian, return the sum of all edge weights.

Step 2: Identify nodes with odd degrees (the number of paths connected to a node is odd).

Step 3: List all possible pairings of odd nodes (for n odd vertices, the total number of pairings possible are $(n – 1) * (n – 3) * (n – 5) \ldots * 1$).

Step 4: For each set of pairings, find the shortest path connecting the pair.

Step 5: Find the pairing with a minimum shortest path connecting pairs.

Step 6: Add the new edges from the previous step to the original graph.

Step 7: The sum of the edges is the total link cost for the probe path.

Algorithm 4 Route Inspection Algorithm

Input: Network communities in a graph, $G = (V, E)$
Output: Probe path sequence, P

```
Initialise nodes and edges in the community
if the graph is Eulerian, then
        Find the shortest path of the graph
        path_cost = ∑edge_cost
        return P
else
        Identify nodes with odd degree
        List all possible pairings of odd-degree nodes
        Find the shortest path connection for each set
        Find the pairing with the minimum shortest path
        Add new edges to the original graph
        Find the shortest path of the new graph
        path_cost = ∑original_edge_cost + ∑new_edge_cost
        return P
```

4.0 LOOP HANDLING AND MONITORING INTER-COMMUNITY EDGES

Active failure detection is essential in SDN to ensure the network's availability and reliability. In this context, injecting probe packets into the network has become a popular technique to detect network failures. However, implementing this technique in a complex network environment with numerous nodes and links requires an efficient technique, as the probe may be required to travel on a link multiple times, which increases the risk of inducing a loop. This paper discusses the FDCD framework's implementation mechanism, which includes loop handling in the probe path and inter-community failure detection to ensure effective probe packet transmission and reception.

4.1 Loop Handling in the Probe Path

The FDCD proposed framework injects probe packets into the network to perform failure detection. The packets enter the network at a point, traverse the network to reach each node and link, and return to the entry point for data collection. The nature of the probe mechanism, specifically the RI algorithm, requires a node as both the source and destination of the packets. Theoretically, the process causes looping in the network, and such concepts would not work in traditional networking architecture. However, SDN programming allows packet header manipulation, which overcomes the loop problem. The technique relies on a 'helper' node, which temporarily becomes the destination of the probe packet [22]. For this study, the probe packet utilises the Internet Control Message Protocol (ICMP) to allow RTT measurement and estimation of end-to-end delay to determine the performance of the proposed framework.

Given a sample network topology, as seen in Fig. 4, a probe generator, for example, a host h1, inserts a packet that travels through every node and must return to itself. The complete path of the probe is as follows:

$$h1 \rightarrow n1 \rightarrow n2 \rightarrow n4 \rightarrow n3 \rightarrow n1 \rightarrow h1$$

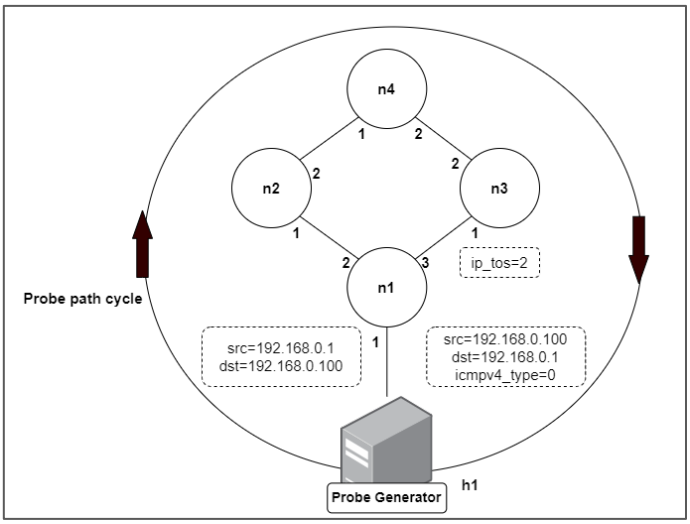

Fig. 4 Loop handling scenario for probe path cycle in a network

In this scenario, the host h1 could not receive the ICMP packet to itself, which causes a loop in the network. The helper node and header modification help remove the problem while ensuring the packet reaches end-to-end connectivity. The host h1 needs to create a packet with the source IP address of h1 and set the destination IP address to the helper node. However, the helper could not directly forward the packet it received as the host h1 could not accept it as a reply to its request. In the ICMP echo request, the icmp_type field is designated to the value 8, while the icmp_type value in the ICMP echo reply is 0. Therefore, the return packet should be modified accordingly for a satisfactory ICMP echo reply to the host h1 by swapping the source IP address (ipv4_src) and destination IP address (ipv4_dst). Also, the icmpv4_type needs to be set from 8 to 0. Table 3 shows the flow entries of all switches to illustrate the process of loop handling in the implementation.

Table 3 Flow entries of all nodes in the loop handling scenario

Node	Port In	Source Address	Destination Address	ToS Field	Actions
N1	1	192.168.0.1	192.168.0.100	1	out_port = 2
	3	192.168.0.1	192.168.0.100	1	out_port = 1 set_field = ipv4_src→192.168.0.100 set_field = ipv4_dst→192.168.0.1 set_field = icmpv4_type→0
N2	1	192.168.0.1	192.168.0.100	1	out_port = 2
N3	2	192.168.0.1	192.168.0.100	2	out_port = 1
N4	1	192.168.0.1	192.168.0.100	1	out_port = 2

In addition to the address and ICMP type header modifications, the Type of Service field (ip_tos) in the flow table can be exploited to fit multiple packet flows with different paths. The ip_tos field is an 8-bit field used as a tagging to differentiate packets originating from the same source. In the above case, the ip_tos tag for N3 changed to 2 because sending the probe packets back to N1 causes a loop. Setting a new ip_tos field generates a flow with different tagging that prevents the loop from occurring once the flow reaches N1. Theoretically, there are 256 possible different flows available per source. The tags are particularly useful when a probe source needs to produce multiple paths. For example, in this study, a probe source or a host may need to generate another set of probes to cover the inter-community probes. The CD process drops certain edges as they do not belong to any community. Thus, the probe source must generate a unique packet flow using different paths to cover the missing edges to solve the issue.

4.2 Inter-community Failure Detection

As mentioned, the CD algorithm process dropped certain edges that do not belong to any community. In addition, the Probe Route Finder module only considers edges that belong to a community for consideration in probe path

calculation. In order to solve the problem of probes being unable to reach the edge that is not in any community (i.e., the inter-community edges), the framework applies the tagging mechanism using packet headers. The ip_tos field helps the nodes (i.e., switches) differentiate flows, especially when the probe packets share the same source and destination.

In this study, the following situation requires this tagging process for the following reasons. Firstly, the generated probe path is inserted into the community to detect the failure in its assigned community. In the second situation, another set of probes monitors the inter-community edge directly against its neighbouring community. The phenomenon requires two sets of probe, which originates from the same probe source going in the same direction, which is back to itself but uses different sets of paths to achieve their objectives. Thus, tagging packet headers using the ip_tos field assists the nodes in recognising different packet flows and ensures both scenarios achieve their intended purpose. Fig. 5 illustrates an example of the scenario to justify how the tagging process provides complete coverage for the probe to perform failure detection within the established communities and inter-community.

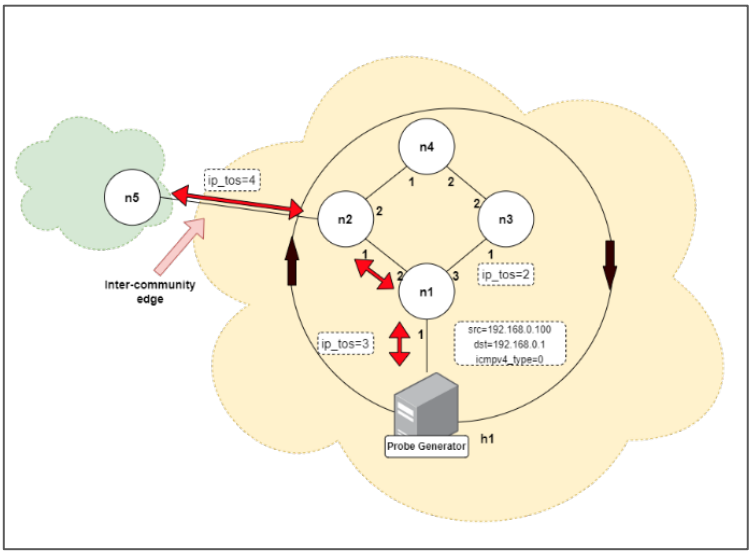

Fig. 5 Inter-community failure detection scenario

The scenario requires two probes to monitor link and node connectivity for failure detection. The instruction for the first probe set is to monitor the intra-community failures, as explained in the previous subsection. The packet headers in the first set occupy the ToS field with values 1 and 2 as a mechanism to allow the probe packet to return to its generator. Therefore, the second probe set needs to use different values to monitor the inter-community path, which includes the inter-community edge. In the example scenario, the second probe set uses 3 and 4 for the ip_tos field to allow the probe to reach the inter-community edge and return to the source, completing its designed task. The example scenario shows how the proposed framework intends to make failure detection in SDN more efficient. Instead of having the probe go through the entire network graph, CD algorithms create smaller boundaries for the probe to reach.

Concisely, the implementation aspects covering the loop handling and inter-community edges provide complete failure detection coverage using the FDCD framework.

5.0 EVALUATION

In order to understand the framework's advantages, a thorough evaluation is vital. Therefore, this chapter evaluates the FDCD framework through extensive experiments. In addition, evaluating the framework includes pitting it against existing solutions to demonstrate the achieved improvements.

5.1 Topology

Scientific experiments require a valid dataset in academic research because it enable accurate performance evaluation and comparison to the existing solutions. However, there is a lack of a use case for SDN-based topology implementations for consideration. The network topology may contain highly sensitive and private information, exposing an organisation's network structure vulnerable to outside attacks. Thus, publicly available network topology information is a viable option for a good reference point. Researchers can alleviate the issue by implementing real-

world and popular network topologies, allowing good comparison with existing approaches. One popular repository for network topology information, called the Survivable Network Design Library (SNDlib), contains multiple network test instances provided by network operators, manufacturers, and academic partners, which are available for public use [16].

Since 2005, the library's existence has invited numerous research efforts to deploy topology instances to solve network design issues. It consists of two parts: network and model. The network part describes the nodes and links, while the model part highlights the specific design parameters, such as directed or undirected links. This dataset's extensive availability of network instances allows an endless selection of network topology for reference. Therefore, the experiments in this study utilised three popular and well-known network topologies from the SNDlib repository. The three instances, COST266, PIORO40, and GERMANY50, have different characteristics, such as the number of nodes and links, to allow observations on differing network sizes. The following discusses the details of each dataset.

a) *COST266*. The topology originates from the project COST266-Advanced Infrastructure for Photonic Networks of the European Cooperation in the Field of Scientific and Technical Research (COST) [23]. It presents a fibre-optic network for European backbone network topology analysis.

b) *PIORO40*. A randomly generated network with a Synchronous Digital Hierarchy (SDH) cost structure [16].

c) *GERMANY50*. A topology originated from the NOBEL project, a European network that details a fibre optic cost model for various equipment. T-Systems International AG provides it as an extension to the NOBEL-GERMANY network topology [24].

Table 4 lists the topology used in this study and highlights the number of nodes and links present in each.

Table 4 Network topologies selected in the study

Topology	**Description**	**Nodes**	**Edges**
COST266	Originates from the project COST266-Advanced Infrastructure for Photonic Networks of the European Cooperation in the Field of Scientific and Technical Research (COST)	37	57
PIORO40	A randomly generated network with a Synchronous Digital Hierarchy (SDH) cost structure	40	89
GERMANY50	Provided by T-Systems International AG as an extension to the NOBEL-GERMANY network topology	50	88

The topologies and SNDlib provide a realistic deployment variable that closely resembles real-world scenarios to allow proper evaluation. In the SDN approach used in this study, the topologies exist in the infrastructure layer, and each node resembles an OpenFlow switch as a forwarding device. A single controller manages these nodes via OpenFlow channels, which allow them to receive forwarding instructions in the flow tables.

In addition, the experiment includes two different topology scenarios to offer a more realistic evaluation approach, as described as follows:

a) Scenario 1: No link weights to replicate a network topology with equal bandwidth, meaning every edge in the topology has an equal weight value of 1.

b) Scenario 2: Included with randomised link weights (1 Gbps, 5 Gbps, and 10 Gbps) to replicate more complex and realistic network conditions. To achieve the randomised weight, the random choice function in Python, `random.choice(sequence)`, is used to set the edge weight once at the beginning of the experiment. This step ensured that the following experiments consistently used the same edge weight values.

5.2 Tools

The emulation of the proposed framework for the experimental evaluation requires hardware and software tools to achieve the objective. The tools included Ubuntu 18.04 LTS, a long-term supported version of the Ubuntu operating system, Mininet 2.2 as the Emulation tool with built-in Open vSwitch, Ryu SDN controller, and Python 3.7 version in the virtualised environment. The environment sat on top of the underlying hardware, a 64-bit operating system desktop with an Intel® Core™ i7-4770 CPU @ 3.4 GHz CPU and 16 gigabytes (DDR4) of internal user memory. The study conducted ten experiments and computed the average results for a fairer evaluation. The evaluation metrics for this experiment include the average path installation time and average probe round trip time, discussed in the next section. Table 5 summarises the emulation environment for the experiment.

Table 5 The emulation environment for evaluation

System Description	Detailed Information
Operating System	Ubuntu 18.04
System Specification	x64-Intel(R)-Core(TM) i7-4770 CPU
Emulation Tool	Mininet 2.2
Remote Controller	RYU 4.34
OpenFlow Support	OpenFlow 1.3
Programming Language	Python 3.7
Network Topology	COST266, PIORO40, GERMANY50
Delay	1 ms
Packet Size (byte)	64

5.3 Results and Discussions

This section discusses the outcome of implementing the proposed framework on the Mininet emulation environment, focusing on metrics such as the average path installation time and average probe round trip time after ten experiment repeats.

a) Average path installation time

The path installation time is required for the controller to put the relevant configurations into the forwarding devices, including the flow rules. The path installation time is determined by the process of rule installation, which is influenced by elements such as the path's length and the controller's capacity to handle the task.

i) COST266 topology

From the results, combining the Louvain and RI algorithms resulted in the quickest path installation time. Contrarily, the baseline RI demonstrated the poorest performance in both the unweighted and weighted topology scenarios. In the unweighted topology scenario, the Louvain and RI algorithm combination obtained an average path installation time of 6.59 seconds, compared to 6.92 seconds for the baseline RI method. In the weighted topology scenario, the Louvain and RI combination reported an average path installation time of 6.61 seconds, whereas the baseline approach required 7.03 seconds on average to install the paths. The combination of Louvain and RI consistently outperformed the other algorithm combinations. The baseline method provided the longest path for the probe and hence required the longest installation time. Other algorithm combinations, such as LP + RI and GN + RI, outperformed the baseline RI algorithm but fared slightly worse than the Louvain + RI combination. In addition, there was only a small variation between the findings of the weighted and unweighted topologies, indicating a small difference in path generation. Table 6 presents the average path installation time in seconds for each algorithm combination compared to the baseline RI algorithm.

Table 6 Average path installation time for COST266 topology

Topology	L + RI	LP + RI	GN + RI	Baseline RI
COST266 (**U**)	6.59	6.71	6.88	6.92
COST266 (**W**)	6.61	6.70	6.95	7.03
Time difference (s)	**0.02**	**0.01**	**0.07**	**0.11**

ii) PIORO40 topology

The experiment done on the PIORO40 topology revealed that the combination of Louvain and RI outperformed other algorithm combinations, as the baseline approach fared badly in terms of average path installation time in the unweighted topology. The average path installation time was 7.36 seconds for the combination of Louvain and RI, 7.39 seconds for the combination of GN and LP, and 7.40 seconds for the combination of LP and GN. Meanwhile, in the weighted topology scenario, the GN combination recorded the lowest path installation time at 7.56 seconds, followed by the LP combination at 7.58 seconds and the Louvain combination at 7.60 seconds. Notably, the Louvain combination had the largest time difference between the two topology scenarios, with a difference of 0.24 seconds compared to other algorithms. It suggests that the community and path generation in Louvain combination scenarios had more significant differences than other algorithm combinations. Furthermore, since only the Louvain algorithm considered link weights to form a community, its results considerably impacted the path installation time between the weighted and unweighted topology scenarios. Nevertheless, the Louvain combination still performed better than the baseline RI. Table 7 summarises the average path installation time results in the PIORO40 topology.

Table 7 Average path installation time for PIORO40 topology

Topology	**L + RI**	**LP + RI**	**GN + RI**	**Baseline RI**
PIORO40 (**U**)	7.36	7.40	7.39	7.44
PIORO40 (**W**)	7.60	7.58	7.56	7.61
Time difference (s)	**0.24**	**0.18**	**0.17**	**0.17**

iii) GERMANY50 topology

Regarding the GERMANY50 topology, the combination of Louvain and RI produced the quickest average path installation time in the unweighted topology, 7.79 seconds. The LP combination came in second with 7.82 seconds, followed by the GN combination with 7.94 seconds. In comparison, the baseline RI algorithm provided the slowest results in the unweighted topology scenario, requiring an average of 7.95 seconds. The Louvain and LP combination exhibited the best performance for the weighted topology scenario with 7.85 seconds, followed by the GN combination and the baseline algorithm with respective times of 7.90 and 8.02 seconds. With the GERMANY50 topology, the time gap between the weighted and unweighted scenarios was smaller than in other topologies. The biggest difference was 0.07 seconds in the baseline result, while the second-highest difference was 0.06 seconds for the Louvain combination. The results indicated that the algorithm's behaviour affected the community formation and path planning, which calculated modularity gain, including link weights. Table 8 summarises the average path installation time results for the GERMANY50 topology.

Table 8 Average path installation time for GERMANY50 topology

Topology	**L + RI**	**LP + RI**	**GN + RI**	**Baseline RI**
GERMANY50 (**U**)	7.79	7.82	7.94	7.95
GERMANY50 (**W**)	7.85	7.85	7.90	8.02
Time difference (s)	**0.06**	**0.03**	**0.04**	**0.07**

The findings demonstrated a continuous increase in the average path installation time as the size of the topology rose, with an average of around seven seconds required in all scenarios. Due to the enormous size of the network, the controller must insert flow entries to each node, resulting in a longer path installation time. The investigation found that the combination of Louvain and RI had the best average performance for path installation time across all topologies, with a 2.67% improvement over the baseline method, which was the unmodified RI algorithm. The LP algorithm combination came in second with a 2.13% improvement over the baseline, while the GN algorithm combination only delivered a 0.8% improvement. The study also determined that the CD algorithms efficiently reduced the size of the network to smaller communities. However, the controller was still required to input flow instructions to each node, with the baseline approach showing worse results than the CD-based approaches. Thus, although the performance improvement in the average path installation time measure was marginal, network size played a far larger influence in determining path installation time.

In conclusion, the study's analysis reveals that the proposed failure detection strategy employing CD and probing can potentially improve the time efficiency of path installation in SDN networks. The combination of Louvain and RI

performs the best among the studied algorithm combinations. However, network size and topology must be considered when applying the proposed method.

b) Average probe round trip time

The average round trip time (RTT) of the probe is the amount of time it takes to travel from its origin to its destination and back again. In this study, the probe's source and destination were identical, as determined by the probe packets' source address. The experiment determined the time necessary for the probe to traverse the network and return to its starting position.

i) COST266 topology

In the COST266 topology, the Louvain and RI combination demonstrated the probe's best average round-trip time (RTT), edging out the LP combination by a small margin. In the unweighted and weighted scenarios, the RTT values for Louvain and RI were 35.62 ms and 35.66 ms, respectively. In contrast, the GN combination resulted in significantly longer average RTT values of 65.91 and 65.94 ms compared to both Louvain and LP combinations. Despite this, all CD techniques considerably outperformed the baseline RI, which recorded an average RTT of 79.90 ms for the unweighted topology scenario and 79.95 ms for the weighted topology scenario. The results are summarised in Table 9.

Table 9 Average RTT for the COST266 topology

Topology	**L + RI**	**LP + RI**	**GN + RI**	**Baseline RI**
COST266 (**U**)	35.62	35.74	65.91	79.90
COST266 (**W**)	35.66	35.75	65.94	79.95
Time difference (ms)	**0.04**	**0.01**	**0.03**	**0.05**

ii) PIORO40 topology

In the PIORO40 topology, the Louvain and LP combinations fared the best in both unweighted and weighted scenarios, resulting in the lowest RTT metric on average. In both instances, the GN combination had a faster average RTT than the baseline RI, which had the lowest performance. In the unweighted case, the average RTT for the Louvain and LP combinations was 45.40 ms and 45.45 ms, respectively, compared to 65.35 ms for the GN combination and 79.40 ms for the baseline algorithm. Similarly, in the weighted scenario, the average RTT for the Louvain and LP pairings was 45.55 ms and 45.62 ms, respectively. The GN combination and the baseline algorithm recorded 67.60 ms and 79.61 ms. The results are presented in Table 10.

Table 10 Average RTT for the PIORO40 topology

Topology	**L + RI**	**LP + RI**	**GN + RI**	**Baseline RI**
PIORO40 (**U**)	45.40	45.45	65.35	79.40
PIORO40 (**W**)	45.55	45.62	67.60	79.61
Time difference (ms)	**0.15**	**0.17**	**2.25**	**0.21**

iii) GERMANY50 topology

The pairing of Louvain and RI had the fastest average RTT in both unweighted and weighted instances of the GERMANY50 topology. It produced an average RTT of 52.75 ms for the unweighted and 52.85 ms for the weighted topologies. The LP combination produced the second-best performance, with an average RTT of 52.90 ms in the unweighted topology and 52.91 ms in the weighted topology scenario. The baseline RI algorithm performed worst in both scenarios, with an average RTT of 87.92 ms in the unweighted topology and 88.01 ms in the weighted topology. The results for the average probe RTT in the GERMANY50 topology are summarised in Table 11.

Table 11 Average RTT for the GERMANY50 topology

Topology	**L + RI**	**LP + RI**	**GN + RI**	**Baseline RI**
GERMANY50 (**U**)	52.75	52.90	72.96	87.92

GERMANY50 (**W**)	52.85	52.91	77.92	88.01
Time difference (ms)	**0.10**	**0.01**	**4.96**	**0.09**

This study evaluated the performance of the proposed framework in terms of the average probe round trip time (RTT) metric. The results show that the framework's combinations between CD algorithms and the baseline RI significantly improve the RTT metric's performance in all topology scenarios compared to the baseline approach. The Louvain and RI combination showed the best improvement across all topologies, followed closely by the LP and RI combination. However, the GN + RI combination only slightly improved the metric. The weighted and unweighted topology analysis only showed a minor difference in the average RTT performance, attributed to the lack of significant network traffic during the experiment, which made probe delivery consistent but using different paths. The GN + RI combination observed the largest difference in the GERMANY50 topology. The proposed framework significantly improved the RTT metric's performance compared to the baseline approach.

Table 12 Hop count analysis

Topology	**Total number of hop count to complete a cycle**			
	L + RI	**LP + RI**	**GN + RI**	**Baseline RI**
COST266 (**U**)	55	48	66	72
COST266 (**W**)	55	48	67	74
PIORO40 (**U**)	77	79	93	98
PIORO40 (**W**)	84	81	95	100
GERMANY50 (**U**)	82	72	101	105
GERMANY50 (**W**)	86	73	102	110

Further analysis of the number of hop counts reveals the correlation between the path installation time and probe round trip time. The LP combination produced the least total hop count to complete a probing cycle in each topology and weight scenario. Meanwhile, the baseline RI approach resulted in the highest number of hop counts to complete a probing cycle. The high number of total hops in the baseline approach means that the controller required more flow installation to allow the probes to reach their path correctly. These values resulted in a higher time for the probe to complete its round trip. Incidentally, the LP algorithm produced more communities in the topologies than the Louvain and GN algorithms. Although the GN approach produced fewer communities than Louvain and LP, it still performed better than the baseline approach for the hop count metric, resulting in faster installation and probe round trip time. Therefore, the evaluation proved that CD-based approaches managed to improve the efficiency of failure detection using active probes in SDN.

5.4 Benchmarking

The present study aims to evaluate the performance of the proposed framework, FDCD, against three existing works that utilise active detection techniques: heartbeat, port-based, and active probes. In order to ensure a fair comparison, the same topologies were used in the experiment, including COST266, PIORO40, and GERMANY50, and the same emulation environment and tools were employed, including the Mininet network emulator and Ryu SDN controller. The key performance metric for benchmarking was the average failure detection time during the process, which required multiple experiment repeats. Simulation of failure occurs by a randomly selected link that was manually shut down. The failure detection time was defined as the time from the link taken down to the first instance that the controller detected the failure. The average value was recorded after ten iterations. The following subsection presents and discusses the benchmarking results of the proposed framework and the existing approaches. Table 13 below lists the summary of these works with their detailed descriptions.

Table 13 Summary of existing works for benchmarking

Abbreviation	**Method**	**Reference**	**Description / Methodology**
RR_Heartbeat	Heartbeat	Renganathan Raja et al. (2016)	Periodically exchange LLDP packets to monitor failure and structure the topology in a subtree division

Sharma_Port	Port-based	Sharma et al. (2020)	Detection of physical port failures using the RouteFlow framework based on port status updates
Chan_AP	Active probes	Chan et al. (2018)	Multiple packet probe cycles based on the Chinese Postman Problem (RI algorithm)
This study	Active probes with community detection	-	FDCD framework uses CD algorithms to reduce network size into smaller communities

a) Average failure detection time

The benchmarking experiment attempted to determine the average failure detection time by simulating the failure of forwarding devices or network links by manually causing a failure on a random link or node.

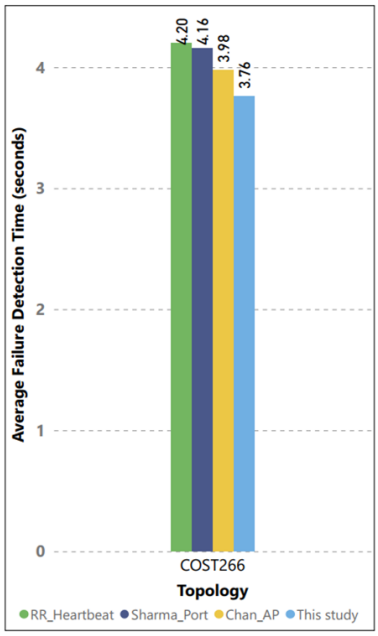

Fig. 6 Average failure detection time in COST266 topology

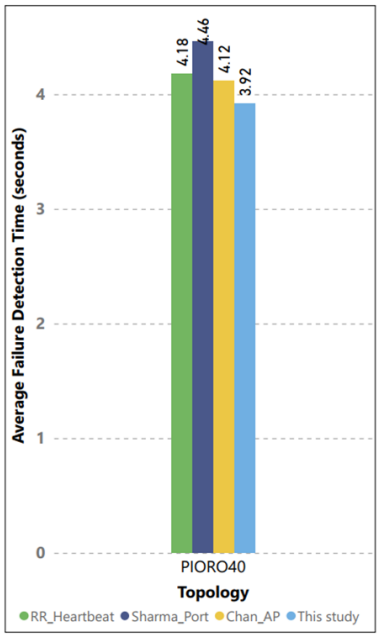

Fig. 7 Average failure detection time in PIORO40 topology

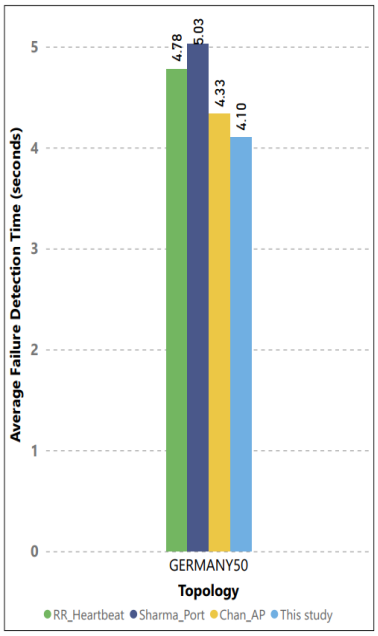

Fig. 8 Average failure detection time in GERMANY50 topology

i) COST266 topology

The benchmarking experiment in the COST266 topology revealed that the FDCD framework had the best performance in terms of the average failure detection time metric, with a time of 3.76 seconds observed. The Chan_AP method, which utilised a modified RI algorithm, followed closely with a failure detection time of 3.89 seconds. The Sharma_Port and RR_Heartbeat approaches, on the other hand, recorded 4.16 and 4.20 seconds, respectively. These results suggest that the strategy utilised by the FDCD framework to reduce network size produced superior performance compared to the other approaches. Fig. 6 visualises the results of the average failure detection time in the COST266 topology.

ii) PIORO40 topology

In the PIORO40 topology, the proposed FDCD framework produced the quickest average failure detection time (3.92 seconds) among the benchmarking methods. The Chan_AP and RR_Heartbeat methods closely followed with average failure detection times of 4.12 and 4.18 seconds, respectively. The Sharma_Port technique, on the other hand, had the longest average failure detection time of 4.46 seconds. The results indicate that as the size of the network grew, the

port-based method in Sharma_Port required a longer controller reaction time since it had to manage more nodes and links. The average failure detection time in the PIORO40 topology is illustrated in Fig. 7.

iii) GERMANY50 topology

In the GERMANY50 topology, the proposed FDCD framework demonstrated the best average failure detection time performance among known techniques. The average time was 4.10 seconds, followed by 4.33 seconds for the Chan_AP technique. The RR_Heartbeat strategy had the shortest average failure detection time at 4.78 seconds, whilst the Sharma_Port approach had the longest at 5.03 seconds. The findings were comparable to the prior topology, demonstrating that the performance of each technique was comparable to the increase in topology size. Fig. 8 provides a graphical representation of the average failure detection time in the GERMANY50 topology.

The benchmarking experiment compared the performance of the proposed FDCD framework to the Chan AP, RR Heartbeat, and Sharma Port approaches. The evaluation measured the average failure detection time for the COST266, PIORO40, and GERMANY50 topologies. Across all topologies, the FDCD framework had the best overall performance, with an average failure detection time of 3.93 seconds. The Chan AP approach clocked in at 4.145 seconds on average, while the RR Heartbeat approach recorded 4.39 seconds on average. The Sharma Port method had the poorest performance, with an average failure detection time of 4.55 seconds. Due to the utilisation of CD techniques, the proposed FDCD framework outperformed the alternatives – the Chan_AP strategy depended exclusively on the modified RI algorithm, the RR Heartbeat approach utilised a heartbeat technique, and the Sharma Port approach utilised a port-based approach responsive to failure. The investigation of each topology revealed that the average failure detection time rose as the network size increased from 37-nodes in the COST266, 40-nodes PIORO40 to 50-nodes in the GERMANY50 configuration. The increase is because of the controller's need to manage more nodes and create more intricate flow tables to enable network functioning. Nevertheless, the new FDCD framework consistently outperformed the other existing techniques. In conclusion, the proposed FDCD framework demonstrated promising results in lowering the failure detection time relative to existing methods. Its use of CD methodologies provides a more effective method for lowering network size and enhancing the efficacy of the failure detection procedure.

6.0 CONCLUSIONS AND FUTURE WORK

In conclusion, this study evaluated the performance of the proposed FDCD framework for failure detection in SDN using active probes. The evaluation involved two key metrics: average path installation time and average probe round trip time (RTT). The results showed that the proposed framework, which utilises CD algorithms to reduce the network size into smaller communities, significantly improved the RTT and path installation time metrics compared to the baseline approach. The results highlight that deploying the proposed framework increases the scalability of the RI algorithm for failure detection as network size increases. However, the performance gain in path installation time only improved slightly due to the large network size that still required the controller to insert flow instructions to each node.

Furthermore, the study benchmarked the proposed framework against three existing works using the average failure detection time metric. The proposed framework outperformed the other approaches due to its utilisation of CD methods, resulting in slightly better performance across all topologies. The study's analysis indicated that the average failure detection time increased as the network size grew. Nonetheless, the proposed FDCD framework consistently performed slightly better than the other existing approaches, indicating a promising direction for future research. The future direction of research in this area may include developing more efficient algorithms for failure detection and evaluation on larger, more realistic network topologies with a variety of data rates to observe the impact of traffic on failure detection performance. Currently, testing the environment in real network deployments incurs enormous network resources, limiting the study to only using Mininet as the testing environment. Overall, the findings of this study suggest that CD methods can enhance the efficiency of failure detection using active probes in SDN.

REFERENCES

[1] K. Greene, "TR10: Software-Defined Networking - MIT Technology Review," 2009. [Online]. Available: http://www2.technologyreview.com/news/412194/tr10-software-defined-networking/. [Accessed: 25-Feb-2019].

[2] MarketsandMarkets, "Software-Defined Networking Market by Component (SDN Infrastructure, Software, and Services), SDN Type (Open SDN, SDN via Overlay, and SDN via API), End User, Organization Size, Enterprise Vertical, and Region - Global Forecast to 2025," 2021.

[3] S. Xi, K. Bu, W. Mao, X. Zhang, K. Ren, and X. Ren, "RuleOut Forwarding Anomalies for SDN," *IEEE/ACM*

Trans. Netw., vol. 31, no. 1, pp. 395–407, 2023.

[4] H. Tahaei, R. Salleh, S. Khan, R. Izard, K. K. R. Choo, and N. B. Anuar, "A multi-objective software defined network traffic measurement," *Meas. J. Int. Meas. Confed.*, vol. 95, pp. 317–327, 2017.

[5] K.-Y. Chan, C.-H. Chen, Y.-H. Chen, Y.-J. Tsai, S. S. W. Lee, and C.-S. Wu, "Fast Failure Recovery for In-Band Controlled Multi-Controller OpenFlow Networks," in *2018 International Conference on Information and Communication Technology Convergence (ICTC)*, 2018, pp. 396–401.

[6] S. S. W. Lee, K.-Y. Li, K. Y. Chan, G.-H. Lai, and Y. C. Chung, "Software-based fast failure recovery for resilient OpenFlow networks," in *2015 7th International Workshop on Reliable Networks Design and Modeling (RNDM)*, 2015, pp. 194–200.

[7] M. Y. Daha, M. S. M. Zahid, B. Isyaku, and A. A. Alashhab, "CDRA: A Community Detection based Routing Algorithm for Link Failure Recovery in Software Defined Networks," *Int. J. Adv. Comput. Sci. Appl.*, vol. 12, no. 11, pp. 712–722, 2021.

[8] T. Arjannikov, S. Diemert, S. Ganti, C. Lampman, and E. C. Wiebe, "Using Markov Chains to Model Sensor Network Reliability," in *Proceedings of the 12th International Conference on Availability, Reliability and Security*, 2017.

[9] V. Tong, S. Souihi, H. A. Tran, and A. Mellouk, "Machine Learning based Root Cause Analysis for SDN Network," in *2021 IEEE Global Communications Conference, GLOBECOM 2021 - Proceedings*, 2021, pp. 1–6.

[10] T. Jafarian, "Security anomaly detection in software-defined networking based on a prediction technique," *Int. J. Commun. Syst.*, vol. 33, 2020.

[11] A. Aydeger, N. Saputro, K. Akkaya, and S. Uluagac, "SDN-enabled recovery for Smart Grid teleprotection applications in post-disaster scenarios," *J. Netw. Comput. Appl.*, vol. 138, no. June 2018, pp. 39–50, 2019.

[12] N. Dorsch, F. Kurtz, and C. Wietfeld, "Enabling hard service guarantees in Software-Defined Smart Grid infrastructures," *Comput. Networks*, vol. 147, pp. 112–131, 2018.

[13] A. Ghannami and C. Shao, "Efficient fast recovery mechanism in Software-Defined Networks: Multipath routing approach," in *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*, 2016, pp. 432–435.

[14] R. Aryan, A. Yazidi, F. Brattensborg, Ø. Kure, and P. E. Engelstad, "SDN Spotlight: A real-time OpenFlow troubleshooting framework," *Futur. Gener. Comput. Syst.*, vol. 133, pp. 364–377, 2022.

[15] S. S. W. Lee, K.-Y. Li, K.-Y. Chan, G.-H. Lai, and Y.-C. Chung, "Path layout planning and software based fast failure detection in survivable OpenFlow networks," in *2014 10th International Conference on the Design of Reliable Communication Networks (DRCN)*, 2014, pp. 1–8.

[16] S. Orlowski, M. Pióro, A. Tomaszewski, and R. Wessäly, "SNDlib 1.0-Survivable Network Design Library," in *Proceedings of the 3rd International Network Optimization Conference (INOC 2007), Spa, Belgium*, 2007.

[17] S. Emmons, S. Kobourov, M. Gallant, and K. Börner, "Analysis of network clustering algorithms and cluster quality metrics at scale," *PLoS One*, vol. 11, no. 7, pp. 1–18, 2016.

[18] H. Lu, M. Halappanavar, and A. Kalyanaraman, "Parallel heuristics for scalable community detection," *Parallel Comput.*, vol. 47, pp. 19–37, 2015.

[19] U. N. Raghavan, R. Albert, and S. Kumara, "Near linear time algorithm to detect community structures in large-scale networks," *Phys. Rev. E - Stat. Nonlinear, Soft Matter Phys.*, vol. 76, no. 3, pp. 1–12, 2007.

[20] X. Liu and T. Murata, "How Does Label Propagation Algorithm Work in Bipartite Networks?," in *2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology*, 2009, pp. 5–8.

[21] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," in *Proceedings of the National Academy of Sciences*, 2002, vol. 99, no. 12, pp. 7821–7826.

[22] M. M. Tajiki, S. H. G. Petroudi, S. Salsano, S. Uhlig, and I. Castro, "Optimal Estimation of Link Delays Based

on End-to-End Active Measurements," *IEEE Trans. Netw. Serv. Manag.*, vol. 18, no. 4, pp. 4730–4743, Dec. 2021.

[23] S. De Maesschalck *et al.*, "Pan-European Optical Transport Networks: An Availability-based Comparison," *Photonic Netw. Commun.*, vol. 5, no. 3, pp. 203–225, 2003.

[24] J. Derkacz, A. Jajszczyk, A. Lasoń, J. Rząsa, and K. Wajda, "Next generation optical networks for broadband services: The IST NOBEL approach," in *Proceedings of 2004 6th International Conference on Transparent Optical Networks*, 2004, vol. 1, pp. 69–74.

[25] V. Renganathan Raja, C.-H. Lung, A. Pandey, G. Wei, and A. Srinivasan, "A subtree-based approach to failure detection and protection for multicast in SDN," *Front. Inf. Technol. Electron. Eng.*, vol. 17, no. 7, pp. 682–700, Jul. 2016.

[26] S. Sharma, D. Colle, and M. Pickavet, "Enabling Fast Failure Recovery in OpenFlow networks using RouteFlow," in *2020 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN*, 2020, vol. 2020-July, pp. 1–6.