Received 11 August 2020; accepted 15 August 2020. Date of publication 21 August 2020; date of current version 7 September 2020.

Digital Object Identifier 10.1109/OJCOMS.2020.3018197

A Fast Failure Recovery Scheme for Fibbing Networks

STEVEN S. W. LEE (Member, IEEE), KWAN-YEE CHAN (Member, IEEE), TING-SHAN WONG, AND BO-XIAN XIAO

Department of Communications Engineering, National Chung Cheng University, Chiayi 621, Taiwan

CORRESPONDING AUTHOR: S. S. W. LEE (e-mail: ieeswl@ccu.edu.tw)

This work was supported by the Ministry of Science and Technology of Taiwan under Grant MOST 107-2221-E-194-023-MY2.

ABSTRACT Fibbing is a network technology that can provide flexible routing in IP networks. In a Fibbing network, the network controller cleverly broadcasts link state advertisements (LSAs) of the open shortest path first (OSPF) protocol to generate fake nodes. These fake nodes enlarge the physical topology and turn the network into a virtual network. Although the routers still follow the shortest path routing in the virtual network, the flows are steered along the desired paths in the physical network. Even though a Fibbing network is capable of flexible routing, methods to achieve fast failure recovery have not been well studied. Conventionally, an IP network running the OSPF protocol takes a long time to converge after a failure occurs. The IETF proposed loop-free alternate (LFA) IP fast reroute technology to speed up failovers. However, we discovered that LFA technology is incompatible with Fibbing. The direct application of LFA technology in a Fibbing network will generate traffic loops. In this work, we propose a novel monitoring-cycle-based approach for fast failure recovery. By examining the liveness of the monitoring cycles, the system controller can promptly identify a failed link or a failed node and then perform traffic rerouting. Due to the properties of IP packet forwarding, a monitoring cycle in an IP network must be a simple cycle. We prove that any network that is both 2-vertex- and 3-edge-connected (2V3E) possesses a set of monitoring cycles that can be used to detect and identify any single link failure. In conjunction with the status pattern provided by the monitoring cycles, any single node failure can be detected by introducing an additional probe message. We propose an algorithm to obtain these monitoring cycles and design and implement a Fibbing controller for the network. To evaluate the performance of the proposed approach, we construct a physical testbed and an emulation network. The experimental results show that the proposed system works stably and can achieve a fast failover within a short period of time.

INDEX TERMS IP fast failure recovery, link failure detection, node failure detection, monitoring cycles, fibbing controller design.

I. INTRODUCTION

IN IP networks, routers follow only the shortest paths with respect to link weight metrics to forward packets. Although shortest path routing successfully eliminates traffic loops in a distributed network, it sometimes results in congestion in some links. Due to its inflexible routing, a conventional IP network does not have the ability to support traffic engineering.

Fibbing [1] is a new IP network technology that can steer flows to the desired routing paths. Fibbing was developed to achieve centralized control over a distributed routing paradigm. A Fibbing network has a system controller responsible for introducing fake messages, which generate fake nodes in the network. In the example in Fig. 1, four routers (A, B, C, and D) and one fake node V are shown. Without the fake node, the traffic between hosts H1 and H2 must follow the shortest path H1→C→D→H2. With the introduction of the fake node V and the claim that the path cost from V to destination H2 is 1, this fake node V becomes the next-hop node on the shortest path of traffic forwarded from node C

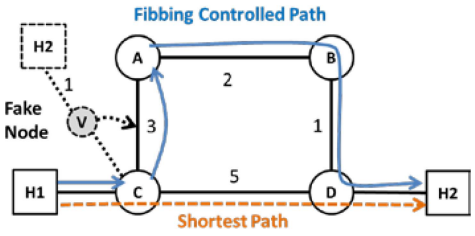
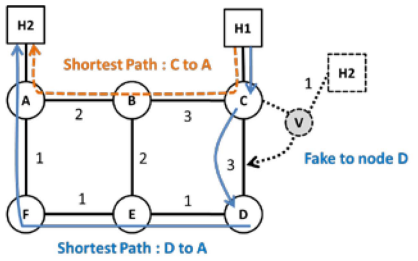
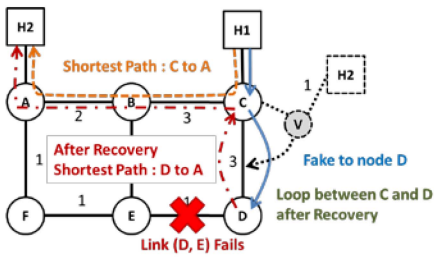


FIGURE 1. Example of traffic steering in a Fibbing network.



(a) Fibbing takes C→D to steer traffic toward H2



(b) Node D forwards the traffic back to node C when link (D, E) fails

FIGURE 2. Fibbing generates traffic loops in a network after link failure occurs.

to H2. Because the interface toward fake node V is actually the same as the interface toward node A, a packet with a destination of H2 is eventually forwarded to node A after leaving node C. As a result, the end-to-end path becomes H1→C→A→B→D→H2.

The Fibbing controller cleverly broadcasts additional open shortest path first (OSPF) link state advertisements (LSAs), which are called fake messages in the following, to generate a virtual topology. The fake messages used in Fibbing include type 1, type 2, and type 5 LSAs. The type 5 LSA is the key to achieving traffic steering. In a network running the OSPF protocol, an AS border router (ASBR) broadcasts a type 5 LSA to announce an external route to the network. In addition to including the ASBR itself in the type 5 LSA, the LSA also includes two major pieces of information. First, the LSA indicates a forwarding address, with which a specific address space can be reached. Second, the LSA specifies the path cost from the forwarding address to the specific address space.

In the example shown in Fig. 1, the type 5 LSA indicates that a forwarding address (i.e., the fake node V) can reach the address space (i.e., a subnet space containing the destination H2) with a cost of 1. For node C, fake node V is the next-hop node. The packets with destination H2 are actually sent to node A. For node A, node B is the next hop on the shortest path. As a result, although the routers inside the network follow only the shortest path, flows can be steered to the desired paths in a physical network

We do not need to use real ASBRs to broadcast fake type 5 LSA messages. Instead, we can generate virtual ASBRs for this task. In the implementation, the Fibbing controller broadcasts type 1 and type 2 LSAs to realize virtualization of the ASBRs. As a result, the real routers inside the OSPF network consider these type 5 LSAs as having been announced by real ASBRs.

In [4], a Fibbing network can use two kinds of fake nodes, each of which generates different scopes of lies in an OSPF network. The forwarding address generated by a locally scoped fake node can be seen only by the router directly connected to the fake node. In contrast, the forwarding address generated by a globally scoped fake node can be seen by all of the routers in the network. In the example shown in Fig. 1, if *V* is a locally scoped fake node, only node C can see the fake node. However, if *V* is a globally scoped fake node, every router in the network can see *V*. In this example, regardless of whether V is a locally scoped or globally scoped fake node, the flow can be steered to the desired path.

Although Fibbing can steer flows to desired paths, methods to achieve fast failure recovery have not been well studied. The existence of fake nodes will result in inconsistencies among routers after a network failure that can generate traffic loops, resulting in network crash.

Fig. 2(a) shows an example. Without Fibbing, the shortest path for the host pair (H1, H2) is C→B→A. The path cost between node C and node A is 5. By introducing fake node V, the new path cost becomes 4. Thus, as shown in Fig. 2(a), the flow is sent to node D instead. Moreover, following the shortest path routing, node D forwards the packets of the flow to node A. Accordingly, the end-to-end routing path becomes C→D→E→F→A.

When link (D, E) fails, the network converges to the new topology shown in Fig. 2(b). In the new topology, node D changes its next-hop node from node E to node C for traffic whose destination router is node A. However, due to the existence of fake node V, node C still uses node D as its next-hop node, generating a loop between node C and node D.

The example depicted in Fig. 2 shows that when Fibbing is applied, an IP network running OSPF will generate traffic loops as a link failure. A straightforward solution is to have the Fibbing controller constantly monitor the network state by examining all the LSAs broadcast inside the network. As the topology changes, the controller must determine a new fake node assignment to avoid traffic loops. Due to the slow convergence of the OSPF network [2], [3], the Fibbing controller takes a long time to detect a failure.

A failure recovery process includes the time spent on failure detection, new fake node deployment, and OSPF protocol convergence. The default period for an OSPF network to

exchange hello messages is 10 s. The dead interval is 40 s. When no receipt of any hello messages occurs within a dead interval from an adjacent neighbor node, the router that detects this situation initiates actions to announce a topology change. Although the above intervals can be reconfigured to reduce them to 1 s, the interval is still too long for time-sensitive applications.

The authors of [4] proposed using bidirectional forwarding detection (BFD) to reduce the failure detection time. The minimum time required to issue a BFD message in a Cisco router is 50 ms. If three consecutive BFD messages are lost, the OSPF network can initiate actions to announce a topology change, which results in a minimum failure detection time of 150 ms in a network using Cisco routers.

Even if BFD is applied, identifying the failed link is one thing; how to perform traffic rerouting is another. The authors of [5] applied the loop-free alternate (LFA) technique [6] to achieve fast rerouting in a Fibbing network. In a conventional IP network, the LFA technique enables a router to bypass a failed link by rerouting the flows interrupted by the failure to its neighboring nodes. More specifically, for a flow with a destination d, when (1) is satisfied, a router x will forward the flow to its neighboring router y.

$$Distance(y, x) + Distance(x, d) > Distance(y, d) \quad (1)$$

Although (1) guarantees loop-free IP traffic rerouting, it does not work well in a Fibbing network. We use the same network shown in Fig. 2 to demonstrate that directly combining Fibbing and LFA technology can generate traffic loops. Because distance(C, D) + distance(D, A) > distance(C, A), when link (D, E) fails, node D can adopt node C as an alternative node to recover the flow toward destination H2. However, due to the existence of fake node V, node D is the actual next-hop node for node C. Consequently, a loop occurs between node C and node D.

NotVia [7] is another approach for IP fast reroute. NotVia is a tunnel-based failure recovery scheme. In a network running NotVia, two types of IP addresses exist: ordinary addresses and NotVia addresses. A NotVia address is used to avoid packets passing through a specific node or link to bypass a failed device. When an IP network is operating in its normal state, only ordinary IP addresses are used. When a node detects a failure, it uses NotVia addresses to perform tunnel-based rerouting. The routing of a recovery tunnel is obtained by running the shortest path algorithm on the network in which the failed device is removed. Consequently, similar to the LFA, NotVia is not applicable in a Fibbing network.

Although routers operate in the electronic domain and can apply BFD for fast failure detection, we have shown that conventional distributed IP fast reroute schemes such as LFA and NotVia are not applicable in Fibbing networks. In addition, even if routers try to use SNMP to notify the Fibbing controller of failure, this event might not be sent to the controller if the failed device is on the route of the notification message being sent to the controller.

In this work, we propose a novel fast failover scheme for Fibbing networks. Routers in the network need to follow only the standard OSPF protocol; no additional failure detection and/or protocols are needed. The proposed scheme is based on deploying multiple monitoring cycles in a network. The Fibbing controller is responsible for periodically generating a monitoring packet for each cycle. If the monitoring packet successfully returns to the controller, all of the devices on the cycle function well. Otherwise, no monitoring packet returns over a period of time, indicating that a fault may exist somewhere in the corresponding cycle. If the cycle status is monitored, the failure link can be identified within a short time.

In graph theory, a connected graph G is said to be k-vertex-connected (or simply k-connected) if it has more than k vertices and remains connected whenever fewer than k vertices are removed. Similarly, a connected graph is k-edge-connected if it remains connected whenever fewer than k edges are removed. In this article, we show the minimum connectivity requirement for a network to guarantee the existence of monitoring cycles to detect any single link failure and any single node failure.

The major contributions of this work are as follows.

- We first present the minimum connectivity degree requirement for a Fibbing network to be able to deploy monitoring cycles. We show that a Fibbing network must be at least a 2-vertex- and 3-edge-connected (2V3E) network to fulfill the requirement.
- We present an algorithm to obtain a set of monitoring cycles that can be used to detect any single link failure in a network.
- Based on the status of the monitoring cycles, we further propose a scheme to detect any single node failure in a network.
- We introduce a method to realize these monitoring cycles, where the Fibbing controller is the common starting point used to generate the monitoring packets.
- We implement the whole system in a testbed and an emulation network and conduct experiments to validate the proposed approach. The experimental results show that the system can perform fast failure recovery in a very short time.

The remainder of this article is organized as follows. In Section II, we describe the basic principles of the proposed monitoring-cycle-based failure recovery scheme. In Section III, we present and prove the requirement of the minimum connectivity degree for a Fibbing network to be eligible for deploying monitoring cycles to detect any single link failure. In Section IV, we provide an algorithm to obtain a set of monitoring cycles. In Section V, we present the design of the Fibbing controller. We analyze the failure recovery times in Section VI. In Section VII, we show how to detect a single node failure. Section VIII shows the experimental results. Finally, we make concluding remarks and discuss future research directions in Section IX.

II. BASIC PRINCIPLES OF THE MONITORING-CYCLE-BASED FAILURE RECOVERY SCHEME

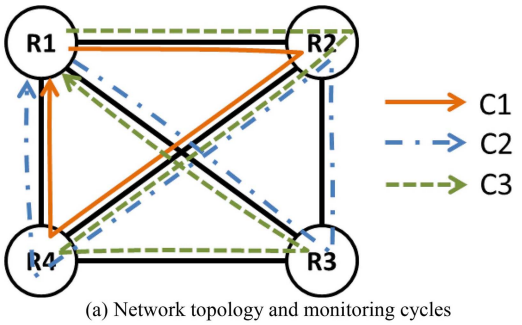
Monitoring-cycle-based network protection was previously applied in all-optical [8], [9], [10] and OpenFlow [11], [12] networks. The reason for applying monitoring cycles in all-optical networks is the lack of an electronic processing capability in optical switches. A monitoring cycle is a lightpath that starts from a monitoring point. By deploying monitoring lightpaths in an optical network, the monitoring points inject optical signals into the monitoring cycles. The monitoring points can determine the cycle status by examining the received optical signals.

Monitoring cycles are used in OpenFlow networks to reduce the failure detection time. Although an OpenFlow switch can use "group action" to automatically react to a failure, the detection time is still too long. As shown in [13], some commercial switches take 2 s to detect a failure. To reduce the failure detection time, monitoring cycles can be deployed in an OpenFlow network. Accordingly, the OpenFlow controller can monitor these cycles to achieve a fast failover.

Unlike all-optical and OpenFlow networks, deploying monitoring cycles in an IP network is subject to an additional constraint: IP routers must follow the destination routing rule. Except for ECMP, in a router, only one outgoing interface exists for any destination address in the routing table. Thus, regardless of the source address, for a packet with destination address d, the router always forwards this packet to the next-hop node using its single outgoing interface for d. This unique constraint prevents the deployment of a monitoring cycle that includes more than one outgoing link at any router. In other words, any monitoring cycle in an IP network has to be a simple cycle in which the only repeated vertex along the path in the cycle is the controller, which is both the first and the last node.

The first question that needs to be clarified is as follows: what kind of topology is feasible for deploying a set of simple cycles that can detect any single link failure? In addition, given a set of monitoring cycles, how can they be realized in a Fibbing network? In this work, we present and prove the minimum degree of connectivity for a network to be able to generate a set of simple cycles for a Fibbing network. We propose an algorithm to produce these cycles and present the design of the controller used to control and manage the whole network.

Before presenting the proposed fast failure recovery scheme, we first review how monitoring cycles are used to identify a failed link in an all-optical network [8]. To detect any single link failure, each link on the network clearly must be covered by at least one monitoring cycle. The idea behind fast failure detection is to deploy a set of cycles such that the outcome of the liveness of these monitoring cycles presents a unique pattern for each single link failure. In other words, given any two distinct links, k and k', in the network, the status pattern of the monitoring cycles when link k fails must be different from the status pattern of the monitoring cycles when link k' fails. Then, the failure location can be uniquely identified based on the status pattern of the cycles.



(a) Network topology and monitoring cycles

Link \ Cycle	C1	C2	C3
(1, 2)	X	O	X
(1, 3)	O	X	X
(1, 4)	X	X	O
(2, 3)	O	X	O
(2, 4)	X	X	X
(3, 4)	O	O	X

(b) Status patterns of these three monitoring cycles when a specific link fails

FIGURE 3. Example of failed link detection by the status pattern of the monitoring cycles.

An example is shown in Fig. 3, where three monitoring cycles can be observed. The routing for cycles C1–C3 is R1→R2→R4→R1, R1→R3→R2→R4→R1, and R1→R2→R4→R3→R1, respectively. Because C1 and C3 pass through link (1, 2), the failure of this link results in no signals being returned from C1 and C3. As shown in Fig. 3(b), any failed link results in a unique cycle status pattern. Thus, by examining the status pattern, the failure location can be identified without ambiguity.

Given two links k and k', to obtain distinct status patterns for the links, at least two of the following three types of monitoring cycles must exist [8]:

- T1: a T1 cycle includes link k but does not include link k' in the cycle;
- T2: a T2 cycle includes link k' but does not include link k in the cycle;
- T3: a T3 cycle includes both link k and link k' in the cycle.

For link (2, 3) and link (3, 4) in Fig. 3, C2 is a T1 cycle, and C3 is a T2 cycle; thus, the status patterns for these two links are different. For link (1, 2) and link (2, 4), C1 is a T3 cycle, and C2 is a T2 cycle; thus, the status patterns of these two links are also different.

In this work, we need a set of simple cycles that contain at least two of the three types of cycles for any pair of links in the network. However, the cycles provided by the algorithm proposed in [8] may not be simple cycles. Therefore, the algorithm in [8] is not applicable to a Fibbing network.

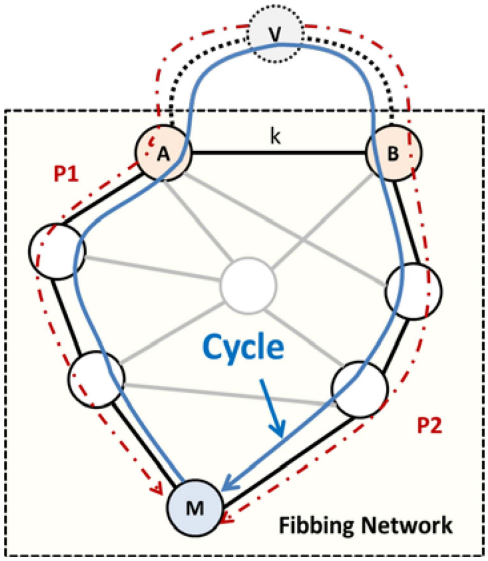


FIGURE 4. Example of generating a simple cycle covering link *k*.

As a result, we propose a new monitoring-cycle-based fast failure recovery scheme for a Fibbing network.

III. NETWORK CONNECTIVITY REQUIREMENT FOR DEPLOYING MONITORING CYCLES IN A FIBBING NETWORK

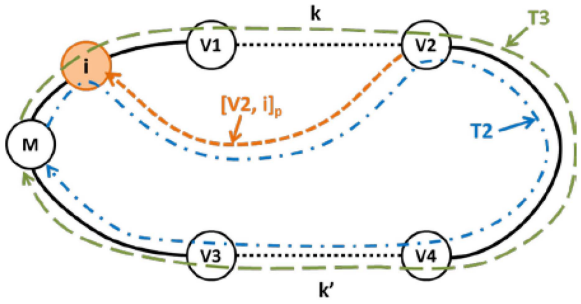
We mentioned that a monitoring cycle is eligible only if it is a simple cycle due to the constraint of IP routing. If a network is 1-vertex-connected, at least one node V exists such that removing it partitions the network into two disconnected subgraphs S1 and S2. If the controller is located in S1, a cycle going through any link in S2 must traverse node V twice. Clearly, this cycle is not a simple cycle; therefore, the network has to be at least 2-vertex-connected.

Ensuring that each link is covered by at least one monitoring cycle is the basic requirement for detecting the status of a link. In the next section, we first present how to obtain a simple cycle that contains a given link. Then, we prove that 2V3E connectivity is the minimum requirement for a network to be able to implement a set of monitoring cycles that can uniquely identify any single link failure.

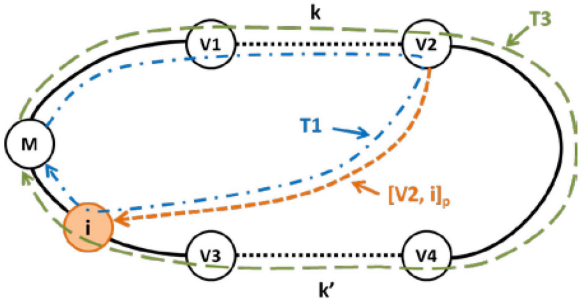
A. FIND A SIMPLE CYCLE TO CONTAIN A GIVEN LINK

Given a link k, Fig. 4 shows how to obtain a simple cycle to include the link. We denote the two end nodes of link k as nodes A and B. We first generate an artificial node V and two artificial links (V, A) and (V, B) that connect V to the two end nodes of link k.

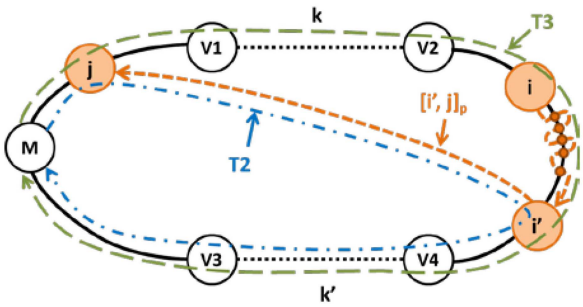
If the network is 2-vertex-connected, it is still 2-vertex-connected after inserting the artificial node V and the artificial links (V, A) and (V, B). Based on Menger's theorem [15], two node-disjoint paths $p1$ and $p2$ exist between node V and node M, where M is the Fibbing controller. By including link k and removing link (V, A) and link (V, B) from the two disjoint paths $p1$ and $p2$, a simple cycle can be obtained. Please refer to [14] for the algorithm used to find two shortest node-disjoint paths between a pair of nodes.



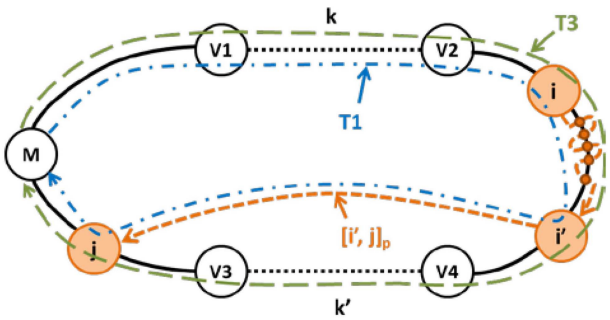
(a) Case 1: path from node $V2$ to node M is first incident to a node on the segment $[M, V1]_c$ of cycle C



(b) Case 2: path from node $V2$ to node M is first incident to a node on the segment $[V3, M]_c$ of cycle C



(c) Case 3: path from node $V2$ to node M is first incident to node i on the segment $[V2, V4]_c$ and then incident to node j on the segment $[M, V1]_c$ of cycle C



(d) Case 3: path from node $V2$ to node M is first incident to node i on the segment $[V2, V4]_c$ and then incident to node j on the segment $[V3, M]_c$ of cycle C

FIGURE 5. Given a T3 simple cycle, a T1 or T2 simple cycle exists in a 2V3E network.

B. MINIMUM CONNECTIVITY DEGREE REQUIREMENT FOR THE FEASIBILITY OF MONITORING CYCLE DEPLOYMENT

In Section III-A, we showed how to find a simple cycle to cover a given link in a 2-vertex-connected network. However, finding a set of monitoring cycles that cover all the links in a network is insufficient. These monitoring cycles have to provide a unique status pattern such that the controller can identify which link has failed. In other words, given any pair of links, the selected monitoring cycles must contain at least two of the T1, T2, and T3 cycles.

We prove that if a network including the controller is a 2V3E-connected network, a set of monitoring cycles exists that fulfils our requirement. For ease of presentation, we define the notation $[x, y]_c$ as the segment that starts from node x, follows a clockwise direction in cycle C, and ends at node y. Similarly, we use $[x, y]_p$ to indicate the segment between node x and node y on path p. In the degenerate case, $[x, x]_p$ is node x itself.

Given two links k and k', we can find a simple cycle C containing link k and a cycle C' containing link k'. Clearly, for links k and k', C is either a T1 or a T3 cycle, and C' is either a T2 or T3 cycle. We categorize the problem into the following cases and show that at least two T1, T2, and T3 cycles exist for links k and k' in the network.

1) If C is a T1 cycle, then regardless of whether C' is a T2 or T3 cycle, we will have the two different types of cycles required to differentiate between link k and link k'.
2) If C is a T3 cycle and C' is a T2 cycle, the requirement is also satisfied.
3) If both cycles are T3 cycles, we use Fig. 5 to show that at least one T1 or T2 cycle exists in this network. We denote the two end nodes of link k and link k' as $(V1, V2)$ and $(V3, V4)$, respectively. Because the network is 3-edge-connected, it will still be connected after removing links k and k'. Thus, after removing links k and k', a path exists from node $V2$ to controller M. We call this path p for short. Because controller M is already in cycle C, path p must be incident to cycle C by at least one node. We denote the first node such that path p is incident to C as node i. According to the location of node i, the problem can be classified into one of the following cases.
   (a) As shown in Fig. 5(a), node i is within the segment $[M, V1]_c$. In this case, the path $[V2, M]_p$ can be decomposed into two segments: $[V2, i]_p$ and $[i, M]_p$. A T2 cycle can be obtained by concatenating the following path segments: $[M, i]_c$, $[i, V2]_p$, and $[V2, M]_c$. Please note that we do not preclude the case where $i = M$. Clearly, a T2 cycle can be obtained by concatenating the segments $[M, V2]_p$ and $[V2, M]_c$.
   (b) Node i is within the segment $[V3, M]_c$. In this case, a T1 cycle can be obtained by concatenating the following path segments: $[M, V2]_c$, $[V2, i]_p$, and $[i, M]_c$. This case is depicted in Fig. 5(b).
   (c) If node i is within the segment $[V2, V4]_c$,we denote the last node such that $[i, M]_p$ is incident to $[V2, V4]_c$ as node i'. We denote node j as the first node such that $[i', M]_p$ is incident to $[V3, V1]_c$. As shown in Fig. 5(c), if node j is within the segment $[M, V1]_c$, a T2 cycle can be obtained by concatenating $[M, j]_c$, $[j, i']_p$, and $[i', M]_c$.
   (d) Otherwise, as shown in Fig. 5(d), if node j is within the segment $[M, V3]_c$, a T1 cycle can be obtained by concatenating $[M, i']_c$, $[i', j]_p$, and $[j, M]_c$.

Combining all of the above cases, we conclude that given a T3 cycle for a pair of links, another T1 or T2 cycle exists in the network. Thus, at least two of the T1, T2, and T3 cycles needed to distinguish the status of a pair of links in a 2V3E network exist.

IV. ALGORITHM FOR OBTAINING THE MONITORING CYCLES

In this section, we present the algorithm used to obtain the monitoring cycles. The input of the algorithm includes a graph $G(N, L)$ with a connectivity degree larger than or equal to 2V3E. We denote node M as the controller in graph G. The output of this algorithm is a set of monitoring cycles $'\Omega$.

Fig. 6 presents the algorithm. $'\Omega[k]$ denotes a set of cycles passing through link k. Initially, $'\Omega$ is an empty set. Γ denotes the set of links that have been covered by at least one of the monitoring cycles in $'\Omega$. When any link r exists that is not included in any monitoring cycles in $'\Omega$, we apply the scheme shown in Section III-A to obtain a simple cycle to cover that link. In the algorithm, $\alpha(r)$ and $\beta(r)$ are the two end nodes of link r, and $(V, \alpha(r))$ and $(V, \beta(r))$ are the two artificial links connecting the two end nodes of r to artificial node V. We apply Suurballe's disjoint path algorithm [14] to find two node-disjointed paths from controller M to artificial node V. By removing the two artificial links $(V, \alpha(r))$ and $(V, \beta(r))$ and including link r in the two node-disjointed paths, this new cycle is added to the monitoring cycle set $'\Omega$. To reduce the number of selected cycles, when a link is included in a cycle, its weight is added with a large number η to prevent that link from being included in future cycles.

After obtaining a set of monitoring cycles that cover all the links, we examine whether the cycles that cover link k and link k' contain at least two of the T1, T2, and T3 cycles. If the two sets $'\Omega[k]$ and $'\Omega[k']$ are exactly the same, then every cycle that passes through link k must also go through link k'. Evidently, these cycles are T3 cycles for the link pair (k, k'); thus, we try to obtain a T1 cycle by removing link k' from the network and then find a cycle to cover link k. If no T1 cycle exists, a T2 cycle must exist based on the proof provided in Section III-B.

The algorithm outputs the final set of monitoring cycles $'\Omega$. The controller monitors the liveness of these cycles and performs a fast failure recovery when a failure is identified.

In the algorithm, we use Suurballe's disjoint path algorithm in our subroutine *two_node_disjoint_path*. When

Fibonacci heap is applied to implement Suurballe's algorithm, its time complexity is $O(L+N\log N)$, where N and L are the numbers of nodes and links, respectively. As a result, the time complexity for our algorithm is $O(L^3+L^2N\log N)$.

To evaluate the computation time of the algorithm, we implement our algorithm in various networks. The program is run on a PC with an Intel i7-4770 core and 10 GB of RAM. We randomly generate 2V3E networks with a number of nodes ranging from 10 to 100 (increasing by 2) and from degree 4 to 9 (increasing by 1). A total of 230 topologies are evaluated.

The computational results are shown in Fig. 7. Fig. 7(a) shows the hop count number of the cycle that has the maximum hop count. Fig. 7(b) shows the number of monitoring cycles produced by the proposed algorithm. The computation times are plotted in Fig. 7(c). As the number of nodes and degrees increase, the algorithm takes longer to come up with a solution. However, even for a network with 100 nodes and degree 9, the program can obtain the monitoring cycles within 0.4 s. Because the monitoring cycle planning problem is not a real-time problem, it is computed offline; the results indicate that the proposed algorithm can meet the time requirement for application in a Fibbing network.

Before closing this section, we include a remark regarding building a 2V3E network. In graph theory, some research works address the design of the minimum cost k-vertex-connectivity network and the minimum cost k-edge-connectivity network. A reader can refer to the book [16], which shows a survey of the connectivity design problems. Although both problems are NP-hard, some approximation algorithms exist in the literature. To the best of our knowledge, no algorithm is available to design a minimum cost 2V3E network. However, a 2V3E network can be obtained by serially applying the two kinds of algorithms. For example, we could first apply the k-edge connectivity algorithm to obtain a 3E network and then use the k-vertex connectivity algorithm to extend a 3E network to a 2V network.

V. SYSTEM ARCHITECTURE AND OPERATIONS

We implement a Fibbing controller for the network. The controller is responsible for computing and setting up the monitoring cycles. Monitoring cycles are realized by introducing fake nodes in the network. By periodically sending monitoring packets to the monitoring cycles and examining the status patterns of the monitoring cycles, the controller can determine the network state and identify a failed link or a failure node.

To reduce the failure recovery time, the controller predetermines the backup paths for each link. Thus, as a failure location is determined, the controller can reconfigure the routing paths within a short period of time. Accordingly, a fast failure recovery can be achieved by generating new fake nodes in the network to reroute the failed flows.

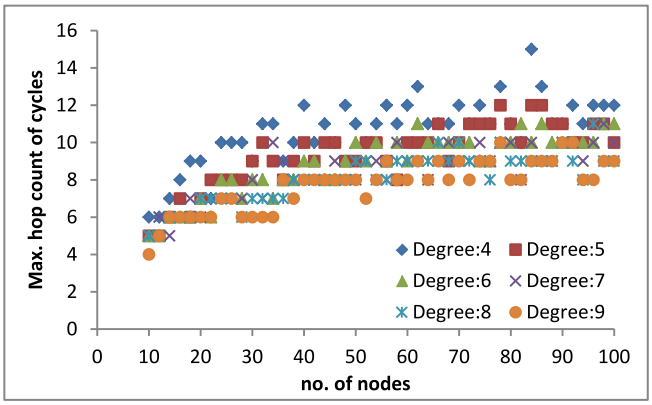
Fig. 8 depicts the architecture of the Fibbing controller. The controller includes four modules: a monitoring cycle placement (MCP) module, a route provisioning (RP) module, a virtual ASBR (vASBR) module, and a failure detection and recovery (FDR) module.

```
Monitoring_Cycles_Planning(G(N, L), M)
// input: G is a 2V3E network, and node M is the controller
begin
    for each link k ∈ L
        Ω[k]: = ϕ; // the initial monitoring cycle set is empty
        w[k]: = 1; // assign the initial link weight
    end for
    Γ: = ϕ; // any member link in Γ is covered by at least one cycle
    while (|Γ|! = |L|)
        // every link must be covered by at least one cycle
        r: = randomly_select_link(L \ Γ);
        N′: = N ∪ {V}; // V is the artificial node
        L′: = L ∪ { (V, α(r)), (V, β(r)) } \ r;
        (P1, P2): = two_node_disjoint_path(G(N′, L′), w, M, V);
        C: = P1 ∪ P2 ∪ {r}\ {(V, α(r)), (V, β(r))};
        for each link k ∈ C
            Ω[k]: = Ω[k] ∪ C; //link k is included in cycle C
            w[k]: = w[k] + η; //add a large weight η to the link metric
        end for
        Γ: = Γ ∪ C; // include cycle C in Γ
    end while
    for each link pair (k, k′) in the network
        if (Ω[k] = Ω[k′]) // for link pair (k, k′), all cycles are T3
            N′: = N ∪ {V};
            L′: = L ∪ { (V, α(k)), (V, β(k)) } \ k \ k′;
            (P1, P2): = two_node_ disjoint_path(G(N′, L′), w, M, V);
            if two disjoint paths (P1, P2) exist
                C: = P1 ∪ P2 ∪ {k}\ {(V, α(k)), (V, β(k))};
                // C is a T1 cycle
            else
                L′: = L ∪ { (V, α(k′)), (V, β(k′)) } \ k \ k′;
                (P1, P2): = disjoint_path(G(N′, L′), w, M, V);
                C: = P1 ∪ P2 ∪ {k′}\ {(V, α(k′)), (V, β(k′))};
                // C is a T2 cycle
            end if
        end if
        for each link k ∈ C
            Ω[k]: = Ω[k] ∪ C; // link k is included in cycle C
            w[k]: = w[k] + η; //add a large weight η to the link metric
        end for
    end for
    return Ω;
end
```

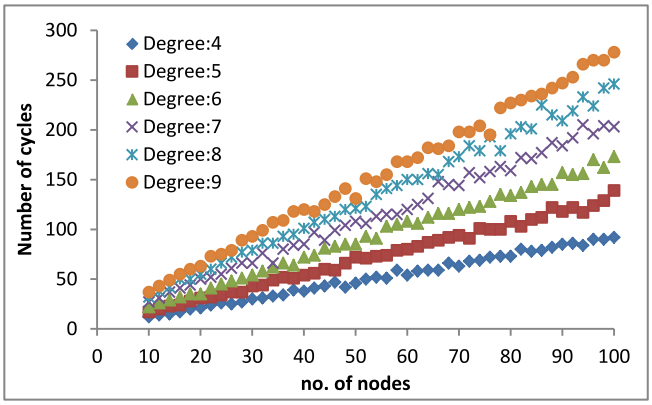
FIGURE 6. Algorithm used to obtain a set of monitoring cycles.

Given the network topology and the control node, the MCP module executes the algorithm shown in Fig. 6 to generate a set of monitoring cycles. The Fibbing controller is the common starting node of the cycles. The obtained monitoring cycles are sent to the RP module for deployment.

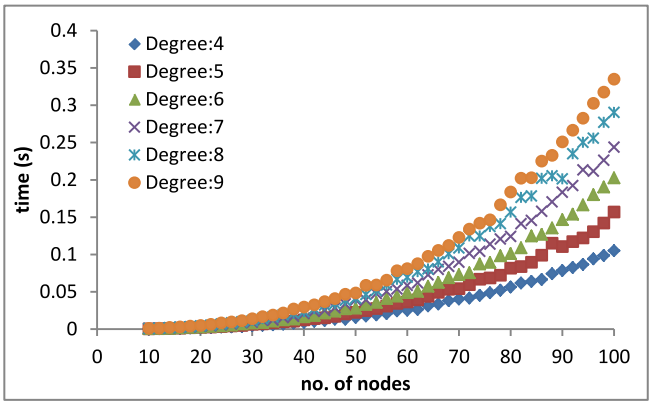
The RP module is responsible for providing routing paths for both data flows and the monitoring cycles determined by the MCP module. The routing paths are provided by introducing fake nodes through broadcasting type 5 LSAs on behalf of the virtual ASBRs into the network.



(a) Maximum hop count of the monitoring cycles



(b) Number of monitoring cycles



(c) Time needed to compute monitoring cycles

FIGURE 7. Analysis results of the algorithm used to find the monitoring cycles.

The vASBR module is responsible for generating virtual ASBRs using type 1 and type 2 LSAs. As shown in Fig. 8, the real routers in the network conclude that four vASBRs exist in the network. However, these ASBRs do not actually exist in the network. The RP module broadcasts type 5 LSAs on behalf of the vASBRs to deploy fake nodes. In our system, each vASBR is implemented by a virtual machine inside the PC. We use Quagga [17] to implement the vASBRs.

The FDR module periodically generates monitoring packets and sends them to the monitoring cycles. When no failure occurs in its cycle, a monitoring packet will return to the controller. To simplify the implementation, time is slotted with

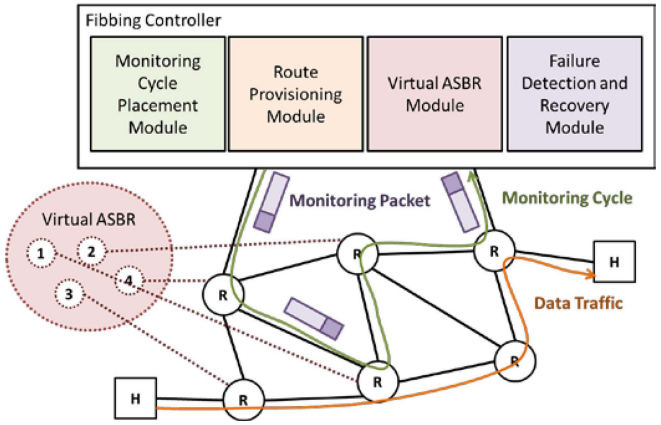


FIGURE 8. Architecture of the system controller.

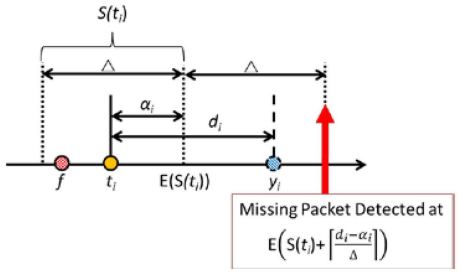
a duration of Δ in the FDR module. The controller sends the monitoring packets and examines the status of each cycle only at the boundary of each slot. To reduce the monitoring packet queuing delay, the differentiated services code point (DSCP) in a monitoring packet is set to have the highest priority. Consequently, monitoring packets cut through congested links without being blocked by other network traffic.

The system is operated in two states: the normal state and the failure recovery state. As soon as a monitoring cycle is discovered that does not send back any monitoring packets within a failure detection window (FDW), the monitoring cycle is declared to be down, and the system enters the failure recovery state. The controller waits an additional traversal delay window (TDW). As the TDW expires, the controller, based on the pattern of all of the monitoring cycles, reroutes the traffic to bypass the failure device.

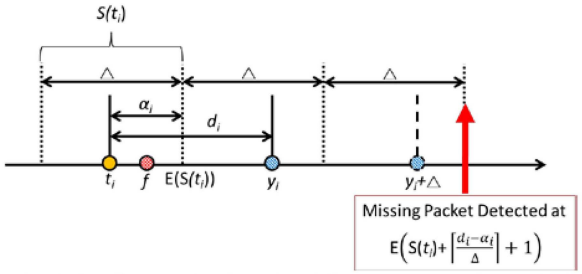
The FDW is used to prevent false alarms. The system uses the FDW to tolerate delay variations and the loss of monitoring packets. In our system, the FDR module does not change the status of a cycle from normal to down unless it fails to receive a returning monitoring packet from the cycle beyond the FDW. The FDW is set to $m\Delta$, with the constant m set to 3 in our current implementation.

Because different monitoring cycles have different lengths and different hops to traverse, the failure device cannot be determined based only on the first down cycle. The controller has to wait to make sure it has the correct status for each monitoring cycle. The TDW serves this purpose. In our implementation, after the first cycle is confirmed to be down, the controller waits an extra TDW $= n\Delta$ to guarantee that all of the cycles present their correct up/down status. The detailed timing and the requirement for the TDW are analyzed in the next section.

In Section III, we proved that a network—including the control node—must be 2V3E-connected to obtain a set of monitoring cycles. This requirement implies that there must be at least three access links connecting the controller to the IP network. However, we find that using only two access



(a) Timing diagram showing when failure occurs before the monitoring packet passes through the failed link



(b) Timing diagram showing when failure occurs after the monitoring packet passes through the failed link

FIGURE 9. Timing diagram showing a monitoring cycle corrupted by a failure.

links to connect the controller to the network is feasible with additional work.

When only two access links are available to connect the controller to two distinct routers in the network, we first introduce an additional artificial access link for the controller, making the network meet the 2V3E connectivity requirement. More specifically, assume that (M, R) is an access link, where R is one of the two neighboring routers of controller M. We introduce an additional artificial link to connect M and R. Consequently, one real link and one artificial link exist between nodes M and R. After applying the proposed algorithm in Fig. 6 to the networks, any monitoring cycle taking the artificial link is replaced by the real link (M, R). Thus, any nonaccess link failures can be identified by the set of monitoring cycles. If any access link fails, all the monitoring cycles go down. Thus, the controller can clearly identify the network status based on the failure pattern.

Please note that the set of cycles does not prevent a nonaccess link x from having all its cycles down as its failure pattern. In fact, Fig. 3 provides an example in which the failure of link (4, 2) results in all the cycles going down. Thus, to apply the algorithm shown in Fig. 6 to a two-access-linked network, we must ensure that no nonaccess link takes all cycles down as its failure pattern. Otherwise, link x and the two access links will be indistinguishable when a failure occurs.

Here, we introduce two solutions to resolve this indistinguishable situation. One solution is to apply BFD to monitor the status of the two access links. When all the monitoring cycles are down but both access links are still working, the controller knows that the failure must have occurred in a nonaccess link.

Another approach is to introduce an additional cycle to prevent any nonaccess links from having a failure pattern with all cycles down. This cycle can be easily obtained by applying Suurballe's algorithm [14] to the network after removing the ambiguous link x in the network. Doing so will prevent any link except the access links from having all cycles down as a failure pattern.

VI. ANALYSIS OF THE FAILURE RECOVERY TIME

As soon as the controller does not receive monitoring packets from a cycle within the FDW (i.e., 3Δ), the system enters the fault identification state, and the controller initiates a TDW window. At the end of the TDW, the controller performs failure recovery by rerouting all disrupted traffic based on the collected pattern of the monitoring cycles.

In this section, we analyze the failure recovery time. The failure recovery time includes how long the controller takes to detect a failure and how long the protocol takes to converge after the RP module in the controller issues new type 5 LSAs to configure the recovery paths. We discuss both factors below.

A. FAILURE DETECTION

We first examine how long the controller takes to detect a cycle down when link l fails. Fig. 9 depicts the timing diagram from the viewpoint of the controller. In this figure, cycle i passes through link l. We denote t_i as the time at which the monitoring packet of cycle i passes through link l and y_i as the time at which the packet returns to the controller if the network is in the normal state. The duration between y_i and t_i is defined as d_i. $\mathrm{S}(t_i)$ is the time slot containing t_i, and $\mathrm{E}(s)$ is the end time of slot s. We denote $\alpha_i = \mathrm{E}(\mathrm{S}(t_i)) - t_i.$, where α_i is the time difference between the end of slot $\mathrm{S}(t_i)$ and t_i.

Without loss of generality, let us assume the failure occurs at f, which is within the time slot containing t_i. Depending on whether f is before or after t_i, a single time slot difference exists for the controller to detect that cycle i is down.

(a) $f < t_i$: In this case (shown in Fig. 9(a)), failure occurs before the monitoring packet of cycle i arrives at link l. As a result, the controller will not receive this monitoring packet. This missing packet will be detected at the end of slot $\mathrm{S}(t_i) + \lceil \frac{d_i - \alpha_i}{\Delta} \rceil$.

(b) $f \geq t_i$: In this case (shown in Fig. 9(b)), the monitoring packet in slot $\mathrm{S}(t_i)$ can successfully return to the controller. However, the controller will not receive the next monitoring packet. Consequently, the controller will detect the missing monitoring packet at the end of slot $\mathrm{S}(t_i) + \lceil \frac{d_i - \alpha_i}{\Delta} \rceil + 1$.

Please note that the above detection times shown in case (a) and case (b) are correct even if d_i is small. In that case, y_i and t_i fall in the same time slot $\mathrm{S}(t_i)$.

In our system, as the first cycle down is detected, the controller waits for the duration of the TDW. After the TDW timer expires, the controller starts to perform traffic rerouting. The TDW needs to be large enough to ensure that the controller has the correct status for each cycle. For

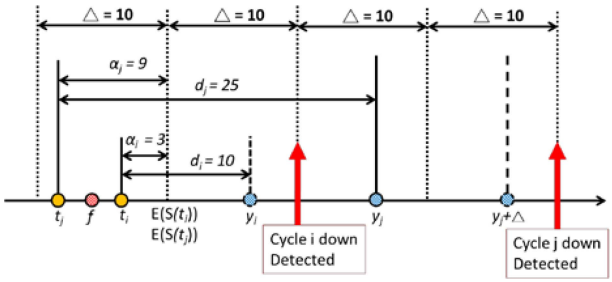


FIGURE 10. Example demonstrating the required slot time between the first and last cycles to be detected as down by the controller.

link l, the minimum value for the TDW should be the difference between the longest detection cycle and the shortest detection cycle.

$$TDW_l = \max_{\forall i,j \in C_l} \left\{ \left(\left\lceil \frac{d_j - \alpha_j}{\Delta} \right\rceil + 1 \right) - \left(\left\lceil \frac{d_i - \alpha_i}{\Delta} \right\rceil \right) \right\} \quad (2)$$

In (2), C_l is the set containing all cycles passing through link l. Fig. 10 depicts an example of assigning TDW_l. In this example, cycle i is the cycle with the smallest $\lceil \frac{d-\alpha}{\Delta} \rceil$ value. It is the first cycle whose monitoring packet cannot be received by the controller due to failure. Because the failure occurs before t_i, the monitoring packet of cycle i that arrives at link l at t_i is blocked by the failure. Thus, this event will be detected in one slot (i.e., $\lceil \frac{10-3}{10} \rceil$) after the slot containing f.

Cycle j is the cycle that has the largest $\lceil \frac{d-\alpha}{\Delta} \rceil$ value. Because the failure occurs at f, which is after the monitoring packet has already passed through link l at t_j, it requires 3 slots (i.e., $\lceil \frac{25-9}{10} \rceil$ +1) to be detected by the controller.

As a special case, the traversal time for each monitoring cycle is within a single time slot Δ. It results in the TDW_l in (2) becoming 1. In this case, the controller needs to wait for only an extra Δ after the first cycle down is detected before it starts to activate the traffic rerouting processes.

In the above analysis, we take only link l into consideration. To determine the TDW for the whole network, we need to consider all of the links in the network. We can set the TDW to the following value.

$$\text{TDW} = \max_{l \in L} \ TDW_l \quad (3)$$

To determine the TDW in (3), we need the exact timing of each cycle in (2) to determine the TDW_l. α_i is difficult to measure for each cycle. Because the traversal time of each cycle is easier to measure, using only cycle traversal times to determine the TDW is a feasible approach.

Toward this target, we use an upper bound as our TDW.

$$\begin{aligned} \max_{l \in L} \ TDW_l &= \max_{l \in L} \left\{ \max_{\forall i,j \in C_l} \left\{ \left(\left\lceil \frac{d_j - \alpha_j}{\Delta} \right\rceil + 1 \right) - \left(\left\lceil \frac{d_i - \alpha_i}{\Delta} \right\rceil \right) \right\} \right\} \\ &\leq \max_{\forall x,y \in C} \left\{ \left(\left\lceil \frac{d_x - \alpha_x}{\Delta} \right\rceil + 1 \right) - \left(\left\lceil \frac{d_y - \alpha_y}{\Delta} \right\rceil \right) \right\} \\ &\leq \left\lceil \frac{T_{max}}{\Delta} \right\rceil + 1 \end{aligned} \quad (4)$$

In (4), C is the set of all monitoring cycles in the network. The value of T_{max} is the traversal time of the longest cycle in the network. It is the time needed for a monitoring packet to traverse the entire longest cycle. Thus, we can set the TDW as $\lceil \frac{T_{max}}{\Delta} \rceil + 1$, which is guaranteed to be long enough for any single link failure cases.

Please note that Eq. (4) holds even if every monitoring cycle has a short traversal time. If the traversal time for each monitoring cycle is within a single time slot Δ, only a single time slot is enough for the TDW.

B. PROTOCOL CONVERGENCE TIME

The protocol convergence time depends on the flood pacing time (FPT) of the routers in the network. Because the controller identifies the exact location of a failed link, new type 5 LSAs are broadcast to reconfigure the recovery paths. However, when a router receives LSAs, it does not relay them to its neighboring nodes immediately; instead, it accumulates the received LSAs and packs them into a single packet. The router will not send that packet until the FPT timer expires. Because the failure recovery time depends on the FPT, in our system, we set the FPT to 5 ms to reduce the network reconfiguration time.

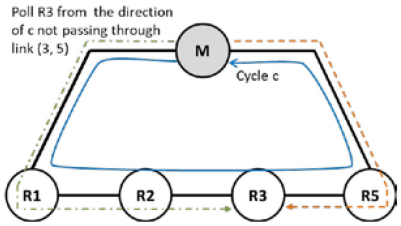
VII. SINGLE NODE FAILURE DETECTION

In this section, we describe how to extend the proposed monitoring-cycle-based approach to achieve fast recovery against single node failure. A single link failure and a single node failure may share the same status pattern of the monitoring cycles. For example, in Fig. 3, when R2 fails, the controller is not able to receive any monitoring packets from cycles C1, C2, and C3. This pattern is the same when link (2, 4) fails. As a result, extra polling is required to determine the status of each node.

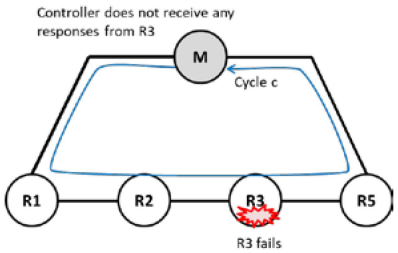
For a node with a unique status pattern, the controller can determine the failed node without ambiguity. The extra status detection is applied only to the nodes that share the failure pattern with a single link failure. We denote the collection of these nodes as Λ.

For a failure node n that shares the same pattern with link l, the controller follows a monitoring cycle c to poll node n. Because cycle c contains link l, the controller M can reach n from a clockwise path or counterclockwise path along cycle c. However, only one of the two paths does not pass through link l. We denote the path that does not go through link l as q_n, which is the path used to determine the liveness of node n. By sending a ping packet along q_n to node n, the controller can detect whether node n is active or not.

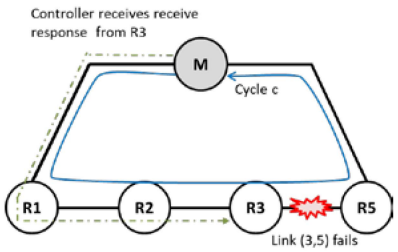
Fig. 11 demonstrates an example. Assume R3 in Fig. 11(a) is a node sharing the same failure pattern as that of link (3, 5). Cycle c is one of the monitoring cycles containing link (3, 5). In this example, following the counterclockwise path $q_{R3} = \text{M} \rightarrow \text{R1} \rightarrow \text{R2} \rightarrow \text{R3}$ on c, the controller can reach R3 without passing through link (3, 5).



(a) Controller ping R3 following P_{R3}=M→R1→R2→R3



(b) R3 fails if no ping response occurs



(c) Link (3, 5) fails if the controller receives the ping response

FIGURE 11. Example of node failure detection.

Please note that we do not preclude that two nodes *n1* and *n2* have the same failure pattern on the monitoring cycles. However, by examining the response of the polling packets, the controller can successfully differentiate the failure node.

To reduce the failure detection time, the node status polling is not started after the controller detects a failure with a status pattern shared by a single link and a single node failure. Instead, the controller can probe every node in Λ at every time slot. Therefore, the node status probing time and cycle status detection time overlap. Similar to the monitoring packets used in the monitor cycles, we take fault torrent into consideration on the ping packets. The controller declares that a node fails only when it does not receive any ping packets from a node in at least three consecutive slots. As a result, the controller determines the failure location when the TDW expires.

VIII. EXPERIMENTAL AND EMULATION RESULTS

We first implemented our system on a testbed with ten Cisco routers. We examine the failover times for both single link failure and single node failure. To further evaluate the recovery time of the proposed system in larger networks, we implement an emulator based on GNS3 [19]. The number of nodes in the emulation networks ranges from 10 to 20.

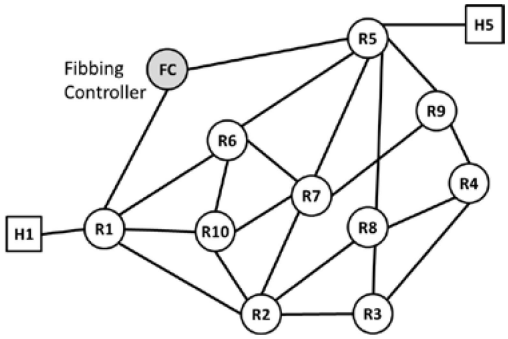


FIGURE 12. Testbed network topology.

A. EXPERIMENTAL RESULTS OF THE 10-NODE TESTBED NETWORK

Fig. 12 depicts the topology of the testbed network. The routers are Cisco 3750 with a 100 Mbps link rate. The weights for each physical link and Fibbing virtual link are 10 and 1, respectively. The FPT is set to 5 ms in each router. The Fibbing controller is implemented on a PC with an Intel i7 8-core CPU running at 3.6 GHz. We set up 11 monitoring cycles to cover all the links in the network. The routing of each cycle is listed below.

C1: R1 → R2 → R7 → R5
C2: R1 → R6 → R5
C3: R1 → R10 → R2 → R8 → R5
C4: R1 → R2 → R3 → R4 → R9 → R5
C5: R1 → R2 → R3 → R8 → R5
C6: R1 → R6 → R7 → R9 → R4 → R8 → R5
C7: R1 → R10 → R6 → R5
C8: R1 → R10 → R7 → R5
C9: R1 → R2 → R8 → R5
C10: R1 → R2 → R3 → R4 → R8 → R5
C11: R1 → R6 → R7 → R5

In the experiments, we observe the recovery time of a particular flow between hosts H1 and H5. The working path follows R1→R2→R3→R4→R9→R5. The source host periodically sends a packet to the working path with a constant interval of 0.7 ms. A failure on the working path will interrupt packet forwarding, and the receiver host will not receive any packets until the connection is restored by the controller. The failure recovery time is measured by the receiver host by examining the time difference between the last packet received from the working path and the first packet received from the recovered path.

We use different slot times Δ in the experiments. Because the processing and queuing delays in a Cisco router are small, the worst traversal time of these monitoring cycles is within 10 ms. Thus, the minimum slot time Δ is set as 10 ms. To evaluate the impact of Δ on the recovery time, in addition to 10 ms, we also take 20, 30, 40, and 50 ms into consideration.

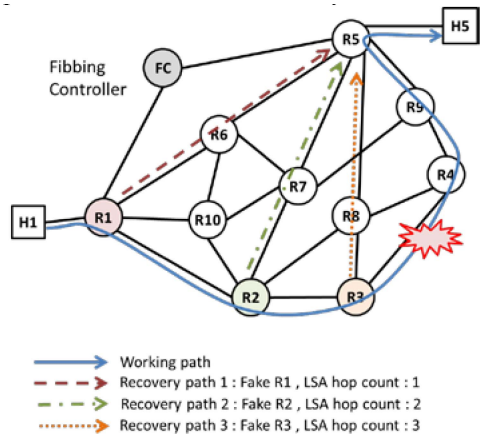


FIGURE 13. Experiments with different recovery path lengths.

TABLE 1. Failure recovery time of the testbed network.

	Recovery Path 1	Recovery Path 2	Recovery Path 3
Max	57.72 ms	61.93 ms	66.88 ms
Min	42.90 ms	44.94 ms	46.46 ms
Average	52.42 ms	54.75 ms	56.10 ms

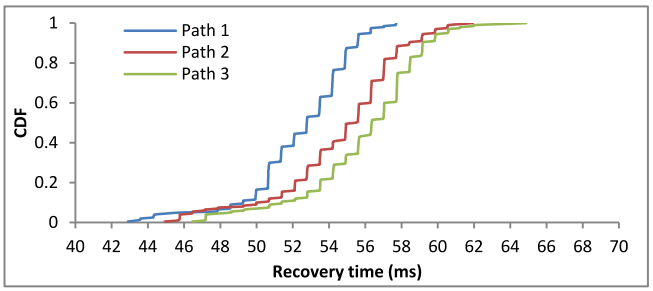


FIGURE 14. CDFs of the recovery times for the three recovery paths.

In the system, the FDW is set to 3Δ, and the TDW is set to a single Δ. If the controller does not receive returning monitoring packets from any of the monitoring cycles in three consecutive Δ, it will start the TDW. After the end of the TDW, the controller determines the failure location based on the status of the monitoring cycles and then starts the traffic rerouting processes.

The controller predetermines the backup paths for each link and each node. When a failure location is detected, the controller reconfigures the routing paths with no further computation needed.

1) THE IMPACT OF THE PRIORITY OF THE MONITORING PACKETS

We first examine the importance of setting the highest priority to the monitoring packets on the stability of the whole system. During the experiments, the network operates normally, and all of the links work without failure. We use iPerf [18] to generate heavy background traffic to congest the links along the routing path R1→R2→R3→R4→R9→R5 between hosts H1 and H5. When the monitoring packets are assigned the same priority as that of the background traffic, the heavy background traffic results in the network dropping the monitoring packets and/or introducing extra delays to the monitoring packets. Because some monitoring packets are unable to return to the controller, the controller may not receive any responses from a particular cycle after the FDW window expires. The Fibbing controller incorrectly concludes that a failure exists and triggers a reconfiguration of the data paths to reroute the network traffic.

We make comparisons between assigning the highest priority and assigning the normal priority to the monitoring packets. The experiments are performed 100 times independently. We make observations for 8 s in each experiment. The results reveal that all 100 experiments generated false alarms within 8 s if the monitoring packets are assigned the same priority as the background traffic. However, when the priority of the monitoring packets is assigned the highest priority, no false alarm is generated.

This experiment shows that assigning the highest DSCP priority to the monitoring packets can successfully prevent the monitoring cycles from being blocked by the data traffic, which is important to the stability of the network.

2) IMPACT OF THE DISTANCE BETWEEN THE CONTROLLER AND THE NODES RESPONSIBLE FOR PERFORMING THE FAILURE RECOVERY

In the second set of experiments, we examine the failure recovery time when link (3, 4) fails. Three different recovery paths are shown as dotted lines in Fig. 13. To examine the impact of the distance that the fake message travels on the failure recovery time, the link metrics of (6, 5), (7, 5), and (8, 5) are set to very small values. As a consequence, during the recovery processes, we do not need to generate fake messages to replace the routing at nodes 6, 7, and 8. More specifically, to activate the first recovery path, the controller needs to generate only a fake message to make a route change at R1. Similarly, to take the second and third recovery paths, the fake messages are applied only to R2 and R3, respectively.

We perform 200 individual experiments for each scenario. In this set of experiments, the time slot is 10 ms. The FDW and TDW are three slots and one slot, respectively. Table 1 shows the average, maximum, and minimum recovery times. The cumulative distribution functions (CDFs) of the results are plotted in Fig. 14.

For the first recovery path, the LSA that is used to generate a fake node at R1 traverses only one hop after leaving the controller. For the second and third recovery paths, the LSA messages need to travel two and three hops, respectively. The experimental results indicate that a longer path requires a longer recovery time. However, even under the worst case in all 200 experiments, the failure recovery time is within 70 ms.

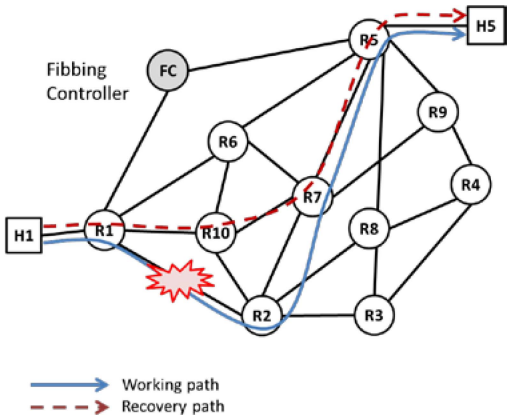


FIGURE 15. Experiments under various settings of Δ and the TDW.

3) DIFFERENT DURATIONS OF THE TIME SLOT AND TDW

Although the delay inside a Cisco router is quite small when a monitoring packet is assigned the highest priority, we do not preclude a network using routers that need longer processing times and have longer delays. In fact, this situation occurs in our GNS3-based emulator shown in the next section. In addition, the routers could be deployed in a wide area network where the nodes are separated by long distances. In addition, when the system is applied in a wide area network, the difference traversal time between a short cycle and a long cycle will exceed a single slot time, resulting in multiple slots being needed for the TDW.

In the third set of experiments, shown in Fig. 15, we evaluate the failure recovery times under different slot times and different durations of the TDW. The working path follows R1→R2→R7→R5. We examine the times needed to recover link (1, 2). Fig. 16 shows the results of the experiments.

The curves of the results are quite linear. Given a fixed slot time, the average recovery time is almost linearly proportional to the length of the TDW. Furthermore, given a fixed TDW, the recovery time also linearly increases with the slot size. The detailed CDF results are shown in Fig. 17. The recovery times under different slot sizes and TDWs are uniformly distributed. Clearly, when the system is applied in a network with a long node processing time or the system is used to control a wide area network, adopting an adequate slot size and TDW can still meet the need to realize fast failure recovery.

4) NODE FAILURE RECOVERY

In the final set of experiments in the testbed, we examine the recovery time after a node fails. Failure is detected by the scheme proposed in Section VII. The experimental network is shown in Fig. 18, and the status patterns of the monitoring cycles that are related to the considered failure event are included in Table 2. The status pattern of the monitoring cycles is the same for link (1, 10) and R10. Link (2, 3) and R3 and link (4, 9) and R9 also share the same status patterns.

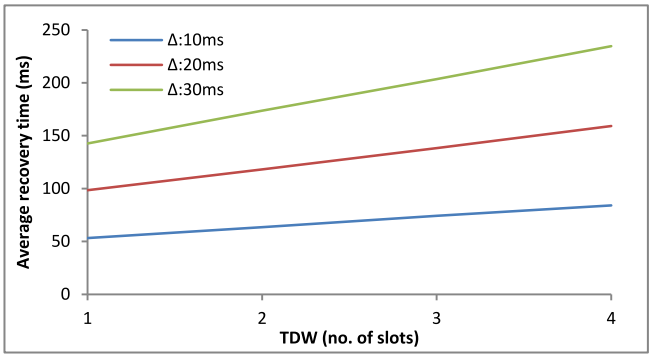


FIGURE 16. Average failure recovery time under different Δ and TDW durations.

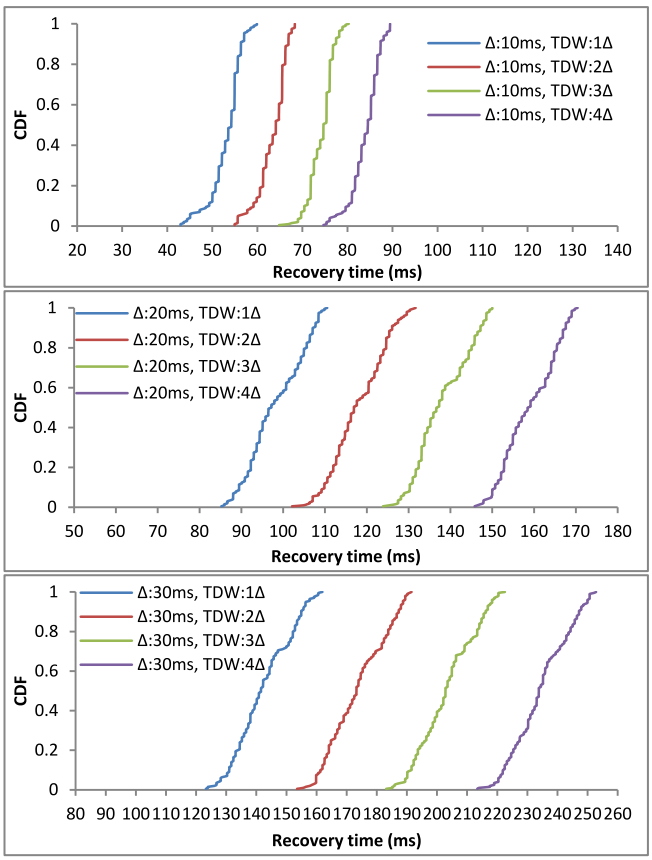
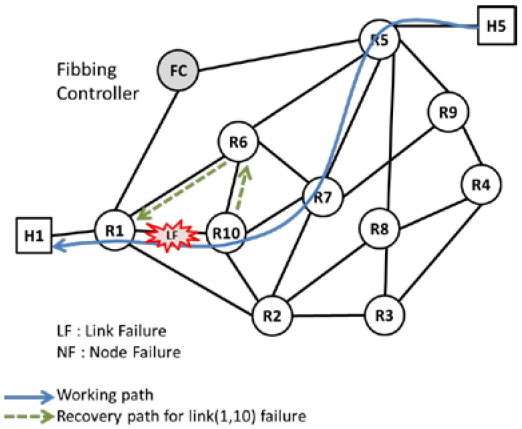


FIGURE 17. CDFs of the recovery times when different Δ and TDW durations are applied.

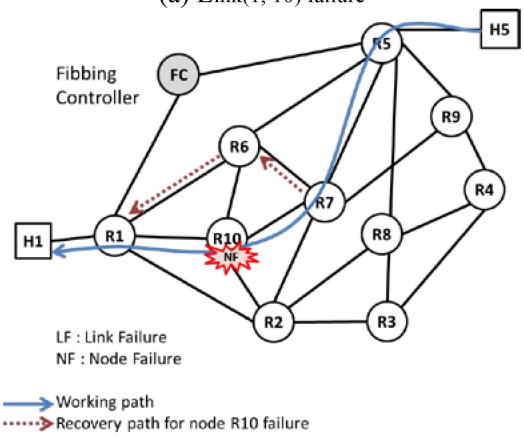
We make comparisons of the recovery times when link (1, 10) and R10 fail. Table 3 summarizes the experimental results, which show that the proposed monitoring-cycle-based scheme can successfully recover from a single link failure and a single node failure. The failure recovery times for link (1, 10) and node R10 are almost the same because the time to detect node failure overlaps with the time to detect the liveness of the monitoring cycles.

B. GNS3 EMULATION RESULTS

We use GNS3 [19] to set up larger experimental networks. The networks are randomly generated with a variety of nodes



(a) Link(1, 10) failure



(b) Node R10 failure

FIGURE 18. Experiments to distinguish link (1, 10) failure or node R10 failure.

TABLE 2. Monitoring cycles and status patterns.

	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11
(1, 10)	O	O	X	O	O	O	X	X	O	O	O
R10	O	O	X	O	O	O	X	X	O	O	O

TABLE 3. Experimental results for link (1, 10) failure and node R10 failure.

	Link (1, 10) Failure	Node 10 Failure
Max	65.51 ms	65.54 ms
Min	47.97 ms	47.16 ms
Average	56.17 ms	56.75 ms

ranging from 10 to 20. The weights for each physical link and virtual link are 10 and 1, respectively. These experiments involve six personal computers, each with an Intel i7 CPU inside. The detailed specifications of these PCs are listed in Table 4. To equally spread the workloads on these PCs, routers are allocated to the PCs sequentially based on their IDs. As a result, a router with ID i is assigned to the PC that has ID i mod 6. The controller is implemented on another PC with an Intel i5 CPU. The source codes of the system and the configurations of the emulators are available in our GitHub repository [21].

TABLE 4. PCs used in the GNS3 emulator.

Fibbing Controller	CPU: Intel(R) Core(TM) i7-4770 CPU @ 3.40 GHz RAM: 10 GB DDR3 1333 MHz
GNS3 Node 1	CPU: Intel(R) Core(TM) i7 CPU 920 @ 2.67 GHz RAM: 16 GB DDR3 1066 MH
GNS3 Node 2	CPU: Intel(R) Core(TM) i7-4790 CPU @ 3.60 GHz RAM: 16 GB DDR3 1333 MHz
GNS3 Node 3	CPU: Intel(R) Core(TM) i7-2600 CPU @ 3.40 GHz RAM 16 GB DDR3 1333 MHz
GNS3 Node 4	CPU: Intel(R) Core(TM) i7-4770 CPU @ 3.40 GHz RAM 16 GB DDR3 1600 MHz
GNS3 Node 5	CPU: Intel(R) Core(TM) i7 CPU 870 @ 2.93 GHz RAM 16 GB DDR3 1333 MHz
GNS3 Node 6	CPU: Intel(R) Core(TM) i7 CPU 870 @ 2.93 GHz RAM 16 GB DDR3 1333 MHz

TABLE 5. Average failure recovery time (in ms).

Node / Degree	10	12	14	16	18	20
4	113.23	119.84	124.41	127.17	131.08	138.20
5	111.10	114.54	118.54	122.10	124.61	126.53
6	109.26	111.67	112.75	116.86	118.19	119.95

1) NETWORK WITH DIFFERENT NUMBERS OF NODES AND MEAN DEGREES

The hardware requirement of GNS3 is described in [20]. In our emulator, the router model is Cisco CSR1000V. The processing delay for each packet is approximately 0.3 ms in a CSR1000V router. For the link delay, we apply the default value. We do not introduce extra delays for the links.

The slot time Δ is set to 20 ms. The FDW and TDW are 3Δ and one Δ, respectively. The routers in the emulator have an FPT of 5 ms, which is the same as that used in the 10-node Cisco testbed network. We apply the same technique used in the 10-node testbed to measure the failure recovery time. The source host periodically sends a packet to the working path with a constant interval of 3 ms. The failure recovery time is measured by the receiver host by examining the time difference between the last packet received from the working path and the first packet received from the recovered path.

In this set of experiments, we randomly generate networks with 2V3E connectivity. Due to the computation capability of GNS3, the largest network is limited to 20 nodes.

Table 5 shows the average recovery times for networks with mean degrees of 4, 5, and 6. We performed 200 independent experiments for each setting. Because the slot time used in GNS3 is two times that used in the Cisco testbed, for the network with 10 nodes, the recovery time is larger than the recovery time observed in the 10-node Cisco testbed network.

The results shown in Table 5 and Fig. 19 reveal that a larger network requires a longer recovery time because the recovery messages need to traverse longer distances on average to reach the target routers. We also observe that networks with a higher degree take a shorter time to recover

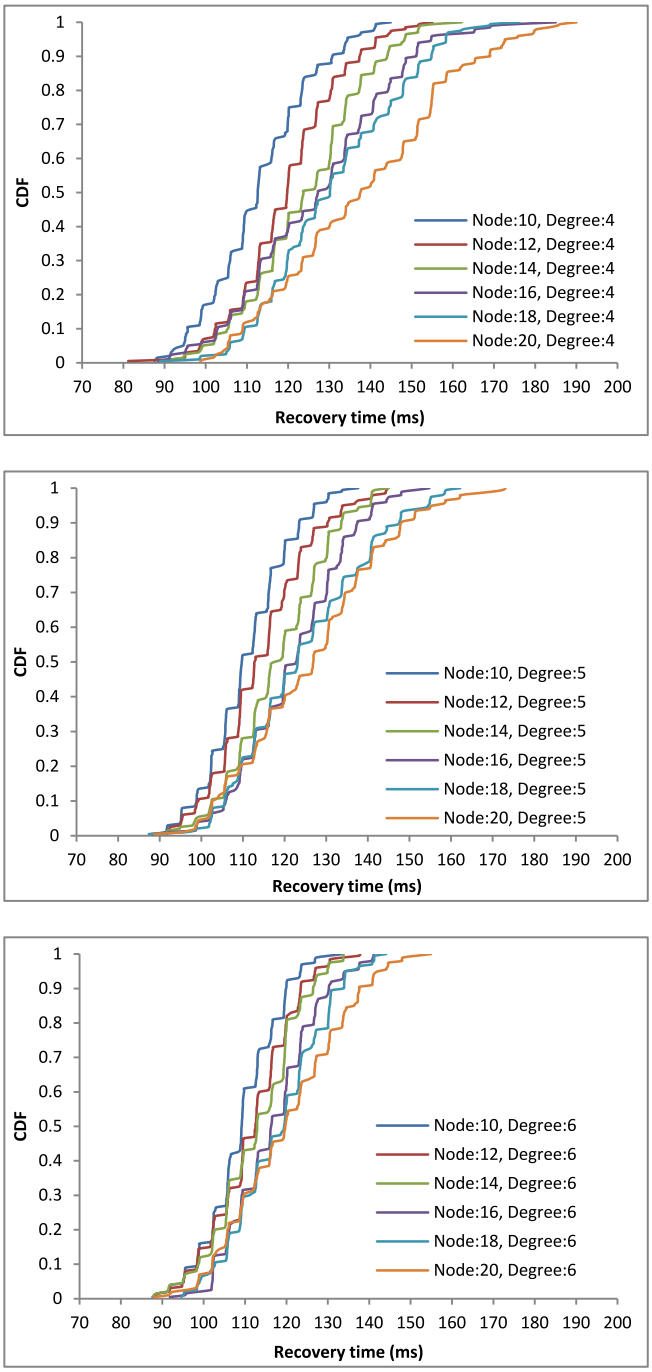


FIGURE 19. CDFs of the failure recovery times in GNS3 networks.

because a higher node degree of connectivity provides a better chance of allocating shorter paths to reroute the affected traffic.

2) EMULATION RESULTS FOR AN IP NETWORK WITHOUT USING THE PROPOSED FAST FAILURE RECOVERY SCHEME

We further set up emulations to observe the failure recovery times in Fibbing-controlled IP networks. The networks do not apply the proposed monitoring-cycle-based failure recovery scheme. As a network failure occurs, the interrupted traffic does not start to perform failure recovery until the controller receives OSPF messages that indicate a topology change.

TABLE 6. Failure recovery time (in s) of Fibbing-controlled IP networks without using the proposed fast failure recovery scheme.

Degree	N: 10	N: 12	N: 14	N: 16	N: 18	N: 20
4	42.24	42.19	42.41	42.21	42.44	42.77
5	42.41	42.61	42.34	42.50	42.28	42.40
6	42.51	42.45	42.23	42.39	42.25	42.68

TABLE 7. Failure recovery time (in ms) of Fibbing-controlled IP networks with the fast hello scheme.

Degree	N: 10	N: 12	N: 14	N: 16	N: 18	N: 20
4	1043.88	1049.62	1043.43	1050.76	1054.20	1054.03
5	1039.86	1049.95	1046.72	1051.92	1050.04	1066.15
6	1059.31	1040.91	1057.50	1058.68	1058.44	1059.41

TABLE 8. Failure recovery time (in ms) of Fibbing-controlled IP networks with the BFD scheme.

Degree	N: 10	N: 12	N: 14	N: 16	N: 18	N: 20
4	266.34	268.81	272.14	274.92	273.28	278.40
5	267.80	272.81	271.86	274.92	275.19	278.39
6	272.64	274.01	273.31	275.66	278.80	282.11

We evaluate the failure recovery time in three kinds of IP networks. In the first one, the IP network follows a typical setting. The hello interval and the dead interval of the routers are 5 s and 40 s, respectively. In the second one, we enable the OSPF fast hello function of the CSR1000V routers. The dead interval is 1 s, which is the minimum allowed value for CSR1000V. In the third set of experiments, we turn on the BFD functions of the routers. The interval is 50 ms, and the multiplier number is three. These are the smallest values allowable for the CSR1000V. Each router sends a BFD packet for its link every 50 ms. If three consecutive BFD packets are lost, the link is claimed to have failed. Please note that not all of the commercial routers support fast hello and BFD. For example, Cisco 3750 routes that used in our 10-node testbed do not have both capabilities.

Table 6 shows the average recovery times on a Fibbing-controlled IP network that follows the first kind of network configuration. The networks take longer than 40 s to recover from a single link failure.

Table 7 shows the results when OSPF fast hello is applied. The networks take longer than one second to recover from a single link failure. Even if OSPF fast hello is applied, a pure Fibbing-controlled OSPF network cannot fulfill the fast recovery requirement for time-sensitive network services.

Table 8 shows the results when BFD is applied. The failure recovery times are larger than 250 ms. Although BFD can improve the failure recovery time, the average recovery time

is still larger than twice that of the proposed cycle-based recovery scheme.

IX. CONCLUSION

Fibbing technology provides a new paradigm that endows an IP network with the flexible routing capability. In a Fibbing network, the system controller can provide paths for traffic engineering in an IP network. However, Fibbing has a long failure recovery time due to the slow convergence of the OSPF routing protocol. Although an LFA is a standard approach employed to achieve fast failure recovery in an IP network, directly applying an LFA in a Fibbing network can generate traffic loops after a failure occurs. In this article, we propose a lightweight cycle-based monitoring scheme that can detect and locate any single link failure within a short period of time. By introducing a few probe messages, our system can also detect any single node failure in a Fibbing network.

The major contributions of this work are 3-fold. First, we proved that any network topology with 2V3E connectivity possesses a set of monitoring cycles that can realize fast failure detection. Second, we presented an algorithm for obtaining a set of monitoring cycles in a Fibbing network. Third, we designed and implemented a Fibbing controller for the system and introduced the architecture and modules of the controller.

The experimental results in a 10-node testbed reveal that our system can successfully recover from any single link failure or any single node failure within 70 ms when a 10 ms slot interval is applied. Because a larger network needs more time to perform failure recovery, the experimental results naturally indicate that employing multiple controllers and enforcing a hop count constraint for the monitoring cycles and recovery paths will reduce the failure recovery time. Possible future research directions include how to synchronize multiple controllers and the design of monitoring cycles in a multicontroller network.

ACKNOWLEDGMENT

The authors thank the anonymous reviewers for their helpful and constructive comments, which greatly contributed to our efforts to improve the quality of this article. The authors would also like to thank Mr. Saeed Barkabi Zanjani for his technical assistance in the early stage of establishing the experimental network using Cisco routers.

REFERENCES

[1] *Fibbing*. Accessed: Jan. 15, 2019. [Online]. Available: http://fibbing.net/
[2] Y. Tsegaye and T. Geberehana. (2012). *OSPF Convergence Times*. [Online]. Available: https://pdfs.semanticscholar.org/200f/c654519d0be791b5a4523fc0226043bb4dfa.pdf
[3] M. Goyal, K. K. Ramakrishnan, and W. C. Feng, "Achieving faster failure detection in OSPF networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, vol. 1, May 2003, pp. 296–300.
[4] S. Vissicchio, O. Tilmans, L. Vanbever, and J. Rexford, "Central control over distributed routing," in *Proc. ACM SIGCOMM*, 2015, pp. 43–56.
[5] S. Vissicchio, L. Vanbever, and J. Rexford, "Sweet little lies: Fake topologies for flexible routing," in *Proc. ACM HotNets*, 2014, pp. 1–7.
[6] A. Atlas and A. Zinin, "Basic specification for IP fast reroute: Loop-free alternates," IETF, RFC 5286, Sep. 2008.
[7] S. Bryant, S. Previdi, and M. Shand, "A framework for IP and MPLS fast reroute using not-via addresses," IETF, RFC 6981, Aug. 2013.
[8] S. Ahuja, S. Ramasubramanian, and M. Krunz, "Single link failure detection in all-optical networks using monitoring cycles and paths," *IEEE/ACM Trans. Netw.*, vol. 17, no. 4, pp. 1080–1093, Aug. 2009.
[9] B. Wu and K. L. Yeung, "Monitoring cycle design for fast link failure detection in all-optical networks," in *Proc. IEEE GLOBECOM*, 2007, pp. 2315–2319.
[10] M. Mao and K. L. Yeung, "Super monitor design for fast link failure localization in all-optical networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2011, pp. 1–5.
[11] 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 *Proc. IEEE RNDM*, 2015, pp. 194–200.
[12] 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 *Proc. ICTC*, 2018, pp. 396–401.
[13] 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 *Proc. IEEE/IFIP DRCN*, 2014, pp. 1–8.
[14] J. W. Suurballe, "Disjoint paths in a network," *Networks*, vol. 4, no. 2, pp. 125–145, 1974.
[15] K. Menger, "Zur allgemeinen Kurventheorie," *Fund. Math.* vol. 10, no. 1, pp. 96–115, 1927
[16] T. F. Gonzalez, *Handbook of Approximation Algorithms and Metaheuristics*. London, U.K.: Chapman and Hall, 2018.
[17] *Quagga*. Accessed: Nov. 20, 2019. [Online]. Available: https://www.nongnu.org/quagga/docs/quagga.html
[18] *Iperf*. Accessed: Nov. 20, 2019. [Online]. Available: https://iperf.fr/
[19] *Software:GNS3*. Accessed: Nov. 20, 2019. [Online]. Available: https://www.gns3.com/software
[20] *GNS3 Hardware Requirements*. Accessed: Nov. 20, 2019. [Online]. Available: https://docs.gns3.com/1PvtRW5eAb8RJZ11maEYD9_aLY8kkdhgaMB0wPCz8a38/index.html
[21] (2020). *The Monitoring Cycle-Based Fast Failure Recovery Scheme for Fibbing Networks*. [Online]. Available: https://github.com/commlab513/MC-based-FFR-Scheme-for-Fibbing-Networks

STEVEN S. W. LEE (Member, IEEE) received the Ph.D. degree in electrical engineering from National Chung Cheng University, Taiwan, in 1999. From 1999 to 2008, he was with Industrial Technology Research Institute, Taiwan, where he was the Leader of Intelligent Optical Networking project and a Section Manager of Optical Communications Department. He was also a Research Associate Professor with National Chiao Tung University, Taiwan, from 2004 to 2007. In 2008, he joined National Chung Cheng University, where he is currently a Professor with the Department of Communications Engineering. He has published over 80 papers in international journals and conferences and has coauthored 20 international patents in the areas of broadband communications. His research interests include optical and broadband networking, network optimization, green network, and software defined networking.

KWAN-YEE CHAN (Member, IEEE) received the B.S. degree in computer science and information engineering from Nanhua University, Taiwan, in 2012, and the Ph.D. degree in communications engineering from National Chung Cheng University, Taiwan, in 2020. Her research interests include survivable network design and software-defined networking. She received the Best Student Paper Award at IEEE CloudNet 2015. In 2017, she and her team members won the First Prize on the SDN/NFV Design Contest hosted by Industrial Development Bureau, Ministry of Economic Affairs, Taiwan.

TING-SHAN WONG received the B.S. degree in communications engineering from National Chung Cheng University, Chiayi, Taiwan, in 2015, where he is currently pursuing the Ph.D. degree with the department of electrical engineering. His main research interests focus on the area of software-defined networking.

BO-XIAN XIAO received the B.S. degree in computer science from Tunghai University, Taiwan, in 2017, and the M.S. degree in communications engineering from National Chung Cheng University, Taiwan, in 2019. In 2020, he joined DreyTek Corporation, Hsinchu, Taiwan, where he is currently a Software/Firmware Engineer. His main research interests include software-defined networking and network survivability.