FLIPPER: Fault-Tolerant Distributed Network Management and Control

Subhrendu Chattopadhyay
Department of CSE
IIT Guwahati, India 781039
subhrendu@iitg.ernet.in

Niladri Sett
Department of CSE
IIT Guwahati, India 781039
niladri@iitg.ernet.in

Sukumar Nandi
Department of CSE
IIT Guwahati, India 781039
sukumar@iitg.ernet.in

Sandip Chakraborty
Department of CSE
IIT Kharagpur, India 721302
sandipc@cse.iitkgp.ac.in

***Abstract*—The current developments of software defined networking (SDN) paradigm provide a flexible architecture for network control and management, in the cost of deploying new hardwares by replacing the existing routing infrastructure. Further, the centralized controller architecture of SDN makes the network prone to single point failure and creates performance bottleneck. To avoid these issues and to support network manageability over the existing network infrastructure, we develop Flipper in this paper, that uses only software augmentation to convert existing off-the-shelf routers to network policy design and enforcement points (PDEP). We develop a distributed self-stabilized architecture for dynamic role change of network devices from routers to PDEPs, and make the architecture fault-tolerant. The performance of Flipper has been analyzed from both simulation over synthetic networks, and emulation over real network protocol stacks, and we observe that Flipper is scalable, flexible and fail-safe that can significantly boost up the manageability of existing network infrastructure.**

***Index Terms*—Network architecture, Fault-tolerance, Programmable network, software defined network**

I. Introduction

Consider the following scenario. The network administrator of an academic institute wants to dynamically update the bandwidth distribution policies based on network usage statistics. The institute network is connected with multiple network service providers, and therefore she needs to update the configuration at different edge routers and gateways. With traditional network devices, like layer 3 switches, this task is tedious, as even a minor configuration inconsistency among the edge routers and gateways may lead to severe network underutilization or bandwidth imbalance. Further, the system is also not scalable for such dynamic updates of network configuration policies.

Software defined networking (SDN) [1] is a technique, which can help in dynamic network configuration update. SDN uses centralized controller to convert policies to device configurations, and to update the targeted devices in the network with the corresponding configurations. To enforce policy to configuration conversion, SDN segregates the network control plane from the data plane. The SDN control plane takes care of all the control functionalities and update of device parameters and configurations, whereas the data plane is only responsible for data forwarding based on the configuration parameters set by the control plane.

Although SDN has revolutionized dynamic network management aspects, it requires specific hardwares that can understand the language for SDN, like OpenSwitch [2], [3], so that a SDN controller can dynamically update the configuration parameters for those hardwares. Therefore the important question is: *How much effort and cost do one need to convert an existing network infrastructure to a SDN supported one?* The existing studies in this direction talk about interoperability among SDN supported and non-SDN network devices, such that incremental deployment of SDN supported devices becomes possible [4]–[6]. However, the concern about cost-effectiveness is still there. SDN supported hardwares are much costlier than commercial off-the-shelf (COTS) network devices, and therefore requires huge operational expenditure to replace existing infrastructure by SDN supported infrastructure.

Although it is quite inevitable that the future of network management is SDN, but simultaneously we also ask this question: *Can we make our existing network more management friendly, such that dynamic network configuration becomes possible without much changing the existing infrastructure?* This paper tries to find out the answer of this question. We show that it is quite possible to use the existing COTS routers to work as network policy decision and enforcement points (PDEP), which are known as network information base (NIB). We can turn a COTS router to a NIB by installing a few additional software tools to support "network function virtualization" [7] (NFV). With the help of NFV functionalities, a COTS router can dynamically update the policy control parameters within its neighborhood [8], [9]. Accordingly, we develop a new network management architecture, which is somewhere in-between the traditional architecture and SDN based architecture, where the COTS routers dynamically change their roles from a conventional network router to a NIB, and participate in PDEP functionalities. We call this architecture *Flipper*.

Flipper has two specific advantages over SDN based network architecture, among others. First, to implement Flipper, a network administrator does not need to procure new costly hardwares, and second, Flipper avoids the controller bottleneck problem [4], [7], [10]–[12] which is much debated in the SDN research community. Flipper is a distributed architecture, where the COTS routers execute a distributed self-stabilizing algorithm to decide which nodes can work as a NIB. As

978-3-901882-89-0 @2017 IFIP

the NIBs have limited resources because they are built on top of the existing routers, a NIB can manage, control and update the network policies only among its neighborhood. Therefore, we develop a distributed self-stabilizing maximal independent set (MIS) selection mechanism, which is indeed non-trivial. To maintain consistency in policy decisions across the network, we have developed a fault-tolerant NIB selection mechanism. We analyse the closure, fault-tolerance and scalability properties of Flipper. The performance of Flipper is analysed from both simulation through a synthetic network environment, as well as through real implementation over an emulation platform using *network name-space*. Our implementation of Flipper provides a proof-of-concept support of the new architecture, while compare the performance with that of other protocols in terms of flow initiation delay.

II. Flipper Architecture

This section gives the details of Flipper architecture and its working procedure. Flipper uses a technology to convert existing COTS routers to PDEP devices through NFV. For this task to convert a COTS router to a PDEP supported device, we use the existing technology called ONIX [7] that describes how the NFV modules can be interfaced with existing router operating system to make it work as a PDEP device that can sync up network policies with other PDEP devices, converts it to network configurations and feeds up those configurations to other normal routers in the neighborhood. Although we use the existing ONIX technology for this purpose, but deploying it over an existing network is non-trivial, because of the limited processing capacity of the existing COTS routers. As a consequence, such devices introduce large delay and processing overhead if a single ONIX node works like a SDN controller. Therefore, in Flipper, we introduce a distributed dynamic PDEP selection mechanism, which is self-stabilized and fault-tolerant. The details of this architecture is discussed next.

A. What is Flipper?

Our proposed Flipper architecture consists of following components which are similar to ONIX. Although the components are similar, the functionalities and arrangement of the components are completely different in Flipper.

1. OpenFlow supported switch (OFS): An OFS is responsible for data forwarding based on forwarding rule set. OpenFlow [13], [14] is a software component that is installed in the router OS to provide NFV functionalities. However, mere OpenFlow support does not make these devices SDN complaint, as specialized hardware (like OpenSwitch) is required for this purpose. In our Flipper architecture, we install additional components only at the software level, but the COTS hardwares are used.

2. Host: End user devices connected with OFSs that hosts the applications and generates data traffic.

3. DHT-NIB: Memory based high update prone eventually-consistent *"distributed hash table"* based NIB (DHT-NIB) for storing link level information of switches. DHT-NIB also helps

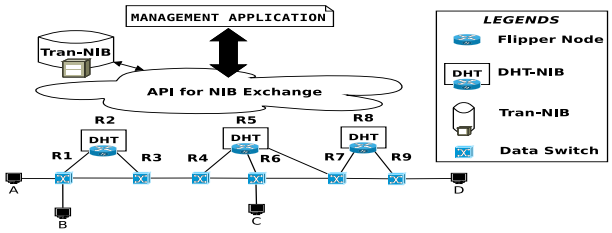


Fig. 1: FLIPPER: Architecture

in setting up forwarding rules in switches based on control application. As shown in ONIX architecture, an OFS can act as a DHT-NIB with additional functionalities.

4.tran-NIB: Strongly consistent tran-NIB is used for rarely changed network wide policy management.

A major difference between the existing SDN based architecture and Flipper is that the standard SDN components have fixed roles to play. However, in this paper we define a Flipper device as a service grade router which can dynamically choose a role of either OFS or DHT-NIB. This dynamic change of roles (*"flip"*) are possible due to use of NFV in Flipper devices. For the sake of readability we refer the Flipper architecture as *"FLIPPER"* and Flipper devices as *"flipper"*.

B. How FLIPPER works?

To understand the working principle of FLIPPER, we take help of Figure 1. The topology consists of a dedicated high performance transactional NIB, hosts (A, B, C, D) and flippers $(R1, R2, ..., R9)$. *"switch-flipper"* if a flipper that acts as a OFS. *"DHT-flipper"* are the flippers that perform DHT-NIB functionalities. Initially, flippers adjust themselves so that a switch-flippers have at least one DHT-flipper in its neighborhood. Upon receiving a flow request from switch-flipper, the distributed control plane consults the relevant DHT-flippers based on the programmable network rules in tran-NIBs and completes the flow table setup procedure in switch-flippers.

C. Fault-tolerance in FLIPPER

The use of NFV for deployment of services provides the flexibility towards FLIPPER. However, general purpose switches in a service provider network are failure prone. The failure of a DHT-flipper can significantly affect the network performance as it controls all the flows in its neighborhood. Therefore, to maintain the robustness of the architecture, FLIPPER needs to be fault-tolerant. For example, in Figure 1, let us assume that $R3, R5$ and $R8$ are acting as DHT-flippers. The associated switch-flippers of $R2, R5$ and $R8$ are $\{R1, R3\}, \{R4, R6, R7\}$ and $\{R9\}$, respectively. If $R5$ fails, $R4, R6$ can not work in the absence of DHT-flipper. For maintaining fault tolerance, we propose a distributed flipper readjustment framework. Whenever one or more switch-flippers detect unavailability of DHT-flipper in its (their) neighborhood, it (they) invokes (invoke) flipper readjustment procedure. The re-adjustment procedure provides the newly selected set of DHT-flippers and switch-flippers. After reaching a consensus, each switch-flipper notifies its adjacent DHT-flipper with its

state information. A switch-flipper having multiple DHT-flippers in its neighborhood chooses a DHT-flipper randomly. Therefore, they can initiate the distributed flipper readjustment framework. In this case, we use distributed self stabilization technique to make the flipper readjustment fault-tolerant.

III. Fault-tolerant Flipper Readjustment

To make the readjustment of switch-flippers and DHT-flippers fault-tolerant, we consider the use of *"self-stabilization"* [15] which is a popular technique to provide defense against *"transient failures"*. A transient failure is defined as irregular and unpredictable brief failure. In this work, we propose a novel flipper readjustment algorithm which expectedly converges with linear time complexity. Our proposed algorithm also satisfies the basic properties of self-stabilization which are as following.

1) **Convergence:** From any state, the system must reach a legitimate or desired state eventually.
2) **Closure:** In case of no failure, the system is guaranteed to remain in legitimate states.

We consider the network as a graph $G = \{V, E\}$, where V is the set of flippers, and E is the set of the edges representing physical connections among flippers. Each flipper periodically senses the physical medium for detecting link failure. A flipper i maintains label $Label_i = \{NIB, Swi, Wait\}$ and priority variable $Pri_i = \{0, 1, \ldots, B\}$, where B denotes the maximum degree of G. Any flipper k with $Label_k = NIB$ signify that, the flipper k is ready to act as a DHT-flipper. Similarly, a flipper l with $Label_l = Swi$ acts as an switch-flipper. We consider flipper with $Label = Wait$ as a flipper with intermediate state whose role is yet to decide. Neighborhood of flipper i is denoted by N_i. Flipper i also maintains $N_i^{NIB} = \underset{j \in N_i}{\forall} j : Label_j = NIB$ and $N_i^{Wait} = \underset{j \in N_i}{\forall} j : Label_j = Wait$. We consider the state of i as $(Label_i, Pri_i)$. Each flipper also maintains the state of its adjacent neighbor flippers. When a flipper changes its state, it pro-actively notifies its neighbors. Upon detecting a link failure, flipper removes the entry about the corresponding neighbor from its table.

We represent our proposed algorithm as a set of guarded actions, where each guarded action is termed as a *rule*. A rule R_j, uses the following representation, $(\mathbf{R_j})|< \mathbf{G_j} > \longrightarrow < \mathbf{A_j} >$, where $< \mathbf{G_j} >$ represents the condition which is required to be satisfied to execute action $< \mathbf{A_j} >$. Upon receiving an update from the neighbor, each flipper checks the guard statements of the rules. If any one of the guard is found to be true, then the corresponding action is executed.

A. flipper Readjustment in Case of Failure

Following the aforementioned model, the flipper readjustment problem is defined as follows. Given a network graph G, the objective of the flipper readjustment problem is to find the set of DHT-flippers in such a way that all switch-flippers can have at least one DHT-flipper in their neighborhoods, so that the policy updates can be done with minimal control plane delay and network overhead. The solution approach must find

Variables:

$\mathbf{Label_i} = \{\mathbf{NIB}, \mathbf{Swi}, \mathbf{Wait}\}$ $\quad$ $\mathbf{Pri_i} = \{\mathbf{0}, \mathbf{1}, \ldots, \mathbf{B}\}$

Functions:

$\mathbf{N^{NIB}(i)} = \forall \mathbf{j} \in \mathbf{N_i} : \mathbf{Label_j} = \mathbf{NIB}$ $\quad$ $\mathbf{N^{Wait}(i)} = \forall \mathbf{j} \in \mathbf{N_i} : \mathbf{Label_j} = \mathbf{Wait}$

$\mathbf{Max_W(i)} = \forall \mathbf{j} \in \mathbf{N_i^{Wait}} : \mathbf{Max(Pri_j)}$ $\quad$ $\mathbf{Trial(i)} : \mathbf{Pri_i} = \mathbf{Rand(0, 1, 2, \ldots, B)}$

Rules:

$(\mathbf{R_1})$—$(\mathbf{Label_i} = \mathbf{Swi})$ $\bigwedge(\mathbf{N^{NIB}(i)} = \emptyset)$ $\longrightarrow$ $(\mathbf{Label_i} = \mathbf{Wait})|\mathbf{Trial}$

$(\mathbf{R_2})$—$(\mathbf{Label_i} = \mathbf{NIB})$ $\bigwedge(\mathbf{N^{NIB}(i)} \neq \emptyset)$ $\longrightarrow$ $(\mathbf{Label_i} = \mathbf{Swi})$

$(\mathbf{R_3})$—$(\mathbf{Label_i} = \mathbf{Wait})$ $\bigwedge(\mathbf{N^{NIB}(i)} \neq \emptyset)$ $\longrightarrow$ $(\mathbf{Label_i} = \mathbf{Swi})$

$(\mathbf{R_{4a}})$—$(\mathbf{Label_i} = \mathbf{Wait})$ $\bigwedge(\mathbf{N^{NIB}(i)} = \emptyset)$ $\bigwedge(\mathbf{Pri_i} = \mathbf{Max_W(i)})$ $\longrightarrow$ $(\mathbf{Label_i} = \mathbf{Wait})|\mathbf{Trial}$

$(\mathbf{R_{4b}})$—$(\mathbf{Label_i} = \mathbf{Wait})$ $\bigwedge(\mathbf{N^{NIB}(i)} = \emptyset)$ $\bigwedge(\mathbf{Pri_i} > \mathbf{Max_W(i)})$ $\longrightarrow$ $(\mathbf{Label_i} = \mathbf{NIB})$

Here $\emptyset$ represents empty set.

Fig. 2: SS-MIS Protocol

an alternative DHT-flipper dynamically when any flipper or link fails, to incorporate fault-tolerance property. The flipper readjustment mechanism is similar to finding a *"maximal independent set"* (MIS) in flipper connectivity graph. We propose a novel distributed anonymous "self-stabilizing MIS" (SS-MIS) algorithm to find DHT-flippers dynamically. The reason for using anonymous algorithm is to remove the unfairness issue caused by the identifier system. Our proposed anonymous SS-MIS protocol has a step complexity [1] of $\mathcal{O}(n)$. Although there exists a linear time self-stabilizing distributed algorithm [16] for solving MIS problem, to the best of our knowledge in case of anonymous systems the best proposed solution [17] has $\mathcal{O}(n \log n)$ step complexity. In this work, we propose a linear time algorithm for anonymous systems that can significantly reduce the control plane overhead in FLIPPER.

B. SS-MIS Algorithm for flipper Readjustment

The proposed SS-MIS protocol selects switch-flippers and DHT-flippers in terms of assigning $Label = Swi$ and $Label = NIB$ respectively. According to MIS properties, no two DHT-flipper can be adjacent and, each switch-flipper should have at least one DHT-flipper in its adjacency list. The proposed protocol is described in Figure 2.

A flipper i which has $Label_i = Wait$ or $Label_i = NIB$, violates the independence property if any of its neighbor is in NIB state. Hence, it must execute (R_2, R_3), and must go to a state having $Label_i = Swi$. If two adjacent flipper have $Label = Wait$, and no other neighbor of them are in $Label = NIB$ state, then both the adjacent flippers will try to enter in a state with $Label = NIB$ state which requires a tie breaking mechanism. Although, the tie breaking can be done using an identifier (ID) of the flipper, in this work ID based tie breaking is not used. ID based tie breaking introduces unfairness problem, because a flipper with higher ID always gets a priority. Therefore, to break this tie, a randomized trial is performed. The proposed random trial is designed in the following way. Each node in $Wait$ state generates a random number in the range $\{0, 1, 2, \ldots, B\}$, and assigns to Pri. A

[1]Execution of an action is called a step. Step complexity of a distributed system is defined as the number of steps executed by the system. Throughout this work, the terms step and move are used invariably.

"Winner" is decided based on the unique maximum Pri value in a closed neighborhood. If no winner is found in a single experiment, it is repeated until there is a winner. The winner gets the privilege to enter into the NIB state.

IV. Properties of Flipper Architecture

In this section we discuss about the properties of proposed flipper architecture. Let the global state of the system be denoted as $\mathcal{S}$; and *legitimate state* is defined as the global configuration where no further rule may be applied at any flipper. We claim that the proposed scheme is self-stabilizing. A proof of self-stabilization requires the proof of **Closure property** and **Convergence property**.

A. FLIPPER Supports Closure Property

Theorem 1: If any flipper in the system is in intermediate state then there is at least one rule which can be executed.

Proof: Assume the state of an intermediate flipper u is $Wait$. Now there can be following scenarios.

Case 1: $\exists v \in N_u : (Label_v = NIB)$. In this case, R_3 will be applicable.

Case 2: $\forall v \in N_u : (Label_v = Wait)$ and $(Pri_v < Pri_u)$. In this case flipper u has unique maximum priority. R_{4b} will be applicable on flipper u and it will act as DHT-flipper.

Case 3: $\exists v \in N_u : (Label_v = Wait)$ and $(Pri_v = Pri_u)$ where Pri_v and Pri_u are maximum in their neighborhood. In this case flipper u and v must apply rule R_{4a} and retrial for a new priority value.

Case 4: $\exists v \in N_u : (Label_v = Wait)$ and $(Pri_v > Pri_u)$. Also $\exists w \in \mathcal{N}(v) : (Label_w = Wait)$ and $(Pri_w > Pri_v)$. From this statement it can be concluded that $(Pri_w > Pri_u)$. Hence priority of these forms a non-increasing function. Also number of flippers are bounded by N. Hence, at least one flipper will have highest priority which will be able to execute rule R_{4b} or R_{4a}. ∎

Corollary 1.1: **(Closure property)** If the system is in a state where flippers with DHT-flippers form a MIS, it will remain in that state forever, provided no further fault occurs.

Corollary 1.1 also suggests the correctness of the proposed scheme. Detailed proof of this statement is omitted due to space restriction of the paper.

B. FLIPPER Converges If a Failure Occurs and It is Scalable

A self-stabilizing system always converges in case of a failure. We analyze the algorithm and prove that, the expected time required to converge is linearly dependent on the number of flipper used.

Theorem 2: Let N denote the cardinality of the closed neighborhood of any arbitrary flipper v. Let $P(N, B)$ denote the probability of finding an unique maximum in the closed neighborhood of v. Then

$$P(N,B) = \frac{(N \times \sum_{i=1}^{B} i^{(N-1)})}{(B+1)^N}$$

Proof: Let i be the highest priority in a configuration $\mathcal{S}$ after one round, where each round corresponds to the event of generating the priority by at most each flipper in the closed neighborhood of v. To satisfy unique maximum property, i can be assigned to any one of the N flippers and the rest of the flippers can have a priority value ranging from 0 to $i-1$. So there will be $N \times i^{(N-1)}$ different possibilities. The value of i can vary from 1 to B. The sample space is $(B+1)^N$ as each node in the closed neighborhood of v can take values from 0 to B independently. Hence the total probability:

$$P(N,B) = \frac{(N \times \sum_{i=1}^{B} i^{(N-1)})}{(B+1)^N}$$

∎

Consider N flippers in the closed neighborhood of v are executing R_{4a} and R_{4b}. To find the expected number of rounds for one of the intermediate flipper to move to DHT-flipper state, we have to find the expected number of rounds in which there will be one flipper with unique maximum Pri in the neighborhood.

Theorem 3: If X denote the random variable indicating the number of rounds required to find a unique maximum priority in the closed neighborhood of v then $E[X] \leq e$, where e represents *"Euler-Mascheroni constant"*.

Proof: For calculating expected number of rounds, we need to determine the probability distribution function

$$Pr[X = r] = (1 - P(N,B))^{(r-1)} \times P(N,B)$$

Clearly this is a geometric distribution and we can say that the expected number of rounds can be calculated as following

$$E[X] = \frac{1}{P(N,B)} = \frac{(B+1)^N}{N \times \sum_{i=1}^{B} i^{N-1}}$$

$$E[X] \leq \frac{(B+1)^{B+1}}{(B+1) \times \int_0^B i^B \, di} = \frac{(B+1)^{B+1}}{B^{B+1}} = \left(1 + \frac{1}{B}\right)^{(B+1)}$$

Note here that the value of N is upper bounded by $(B+1)$. Hence

$$\lim_{B \to \infty} \left(1 + B^{-1}\right)^{(B+1)} = e$$

Therefore $E[X] \leq e$. This result signifies that each flipper needs *"e"* moves on average for the transition from $Label = Wait$ state to $Label = NIB$ state. ∎

Theorem 4: Only the following sequence or subsequence of state change is possible for each flipper during the execution of the protocol.
(Wait→Swi→Wait→Swi), (Wait→Swi→Wait→NIB), (NIB→Swi→Wait→Swi), (NIB→Swi→Wait→NIB)

Proof: We can see in Figure 2 that if a flipper executes R_{4b} then it will not execute any other rule. So no other flipper in its neighborhood can go to $Label = NIB$ state. It can also be shown that if a flipper executes R_{4b} then its neighbors can only execute R_2.

Now from Theorem 3 we can say that each node takes expected e moves to go from $Label = Wait$ state to $Label = NIB$ state. Hence the sequences will take expected $2 + e$ moves. This is true for each flipper. Therefore, we can conclude $\mathcal{O}(n)$ is the expected number of moves for convergence. ∎

Being a self-stabilized algorithm flippers are most of the time available except the convergence time. We have also shown

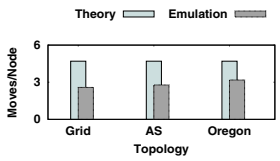


Fig. 3: Moves per flipper

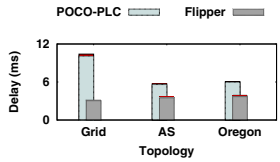


Fig. 4: Flow setup delay

that, the expected convergence time is also within a finite and acceptable bound. Therefore, we can argue that FLIPPER is scalable.

C. FLIPPER is Partition Tolerant

A partition tolerant network can function individually and independently, even if it gets partitioned due to link or node failure. As flipper readjustment does not require any bootstrapping, therefore the proposed architecture is partition tolerant. FLIPPER requires each switch-flipper to have atleast one DHT-flipper in their neighborhood. Corollary 1.1 ensures this property. Therefore, even the network becomes partitioned due to failure, FLIPPER helps them to function individually.

V. Analysis of Flipper Performance from Simulation over Synthetic Networks

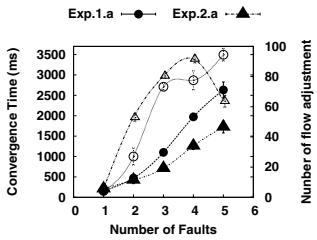
To evaluate the performance of FLIPPER, we simulate the proposed method and compared with one standard fault resilient SDN based framework, called POCO-PLC [18]. POCO-PLC is a distributed SDN platform that uses 20% of controller nodes to provide a Pareto optimal fault resiliency. The controllers act as NIB also. However, POCO-PLC provides an off-line solution of controller placement problem. On the other hand, POCO-PLC can handle limited node failure, whereas FLIPPER can sustain arbitrary node failures.

A. Simulation Setup

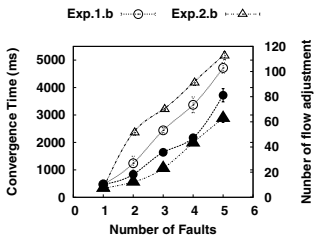
For simulation we use NS-3.22 [19] network simulator. We use three different topologies. Topology 1 is a synthetic 64×64 regular grid topology. Topology 2 (AS Topology [20]) and Topology 3 (Oregon [21]) are real autonomous system data sets taken from University of Oregon Route Views Project BGP logs, where each node represents a border router. For simulation purpose, we consider that these border routers are flipper devices. In each case, flippers are connected via 100 Mbps capacity and 2 ms delay Ethernet channels. Each flipper is configured to generate 4 flows/second with 5 Mbps data rate.

B. Results and Analysis

Figure 3 depicts the average number of moves executed by a flipper in case of random number of flipper failures when SS-MIS is used. It shows that the simulation results do not exceed the theoretical expected bound, which is $2 + e$ (see Theorem 4). The number of used DHT-flipper depends not only on the number of nodes but also on the topology itself. We found that, the required number of DHT-flippers for the two real dataset does not exceed 30%. This result is optimistic in a sense that, if the existing network infrastructure is to be deployed, then $25\% - 30\%$ of the total number of flippers are required to act as DHT-flipper for reducing flow set-up



(a) Effect of Flipper Failure



(b) Link Failure

Fig. 5: Effect of Failure

TABLE I: Emulation Topology Properties

Attributes	Values	Attributes	Values
Nodes	50	Edges	136
Avg Degree	5.44	Max Degree	20
Average DHT-flippers	18 ± 1.23	Average flow set-up delay	42.29 ± 7.2ms

delay. Figure 4 presents a comparison between the proposed flipper architecture and the existing POCO-PLC framework in terms of flow setup delay. The POCO-PLC framework uses a heuristic Pareto-optimal solution for distributed controller placement. Their work suggests the delay will be Pareto-optimal in case of 20% controller usage in most of the network scenarios. However, Figure 4 shows that, in case of flipper, $5\% - 10\%$ increase in number of controllers can reduce flow setup delay by more than 60% for both of the real networks. The performance improvement in terms of flow setup delay is due to the fact that, each switch-flipper has a DHT-flipper in it's neighborhood.

VI. Analysis of FLIPPER from Emulation over a Testbed

Motivated by the simulation results, we have emulated our proposed architecture on top of *mininet* [22]. *mininet* creates virtual nodes for emulated environments over a real networking testbed.

A. Testbed Setup

We have taken a 50 node topology extracted from Oregon dataset [21]. Each node is configured to act a switch-flipper and DHT-flipper with the help of existing OpenVSwitch [23] and OpenVSwitch database server [24] respectively. The flippers are connected via links of 5 Mbps and 2 ms delay. The link characteristics are configured with Linux *"tc"* utility. Each node generates 4 random TCP flows consuming 5 Mbps of bandwidth each. The rest of the topology characteristics and the cumulative results from the emulation are given in Table I. Each flipper periodically checks for the states of adjacent neighbors and links within a time period of 20ms. When a DHT-flipper fails, the newly appointed DHT-flipper interacts with switch-flippers via *"JSON-rpc"* and gathers flow table as well as link state information. The objective of these experiments is to identify the effect of different types of failures on data plane operation. As POCO-PLC uses static role assignment, the convergence time of the protocol becomes irrelevant. Therefore, we do not compare flipper with POCO-PLC in emulated experiments.

B. Effect of Node Failure

We select variable number of flippers as candidates for failure. To visualize the effect of mutual separation (in terms of hop counts) between failed flippers, candidate flippers are selected in following two ways. **Experiment 1.a:** The selected flippers are 1-hop away from each other. **Experiment 1.b:** The selected flippers are more than 2 hops distance apart. The chosen nodes are selected carefully, so that there exists at least one path between the source and the destination of each TCP flow even after the chosen nodes fails. The system can not accept a new flow, until the flipper readjustment converges. Therefore, the convergence time of the flipper readjustment is significant in case of failure. Convergence time for flipper readjustments are shown in Figure 5a. The results suggest that, the effect of multiple flipper failure is dependent on the separation of the failed flippers. However, the convergence time difference reduces for Experiments 1.a and 1.b at $k = 5$ as the diameter of the used topology is 5.

The role change of flippers results in path adjustment of flows. Out of generated 200 random flows, Figure 5a shows the number of flows needs to be readjusted due to the change in data plane topology. The plot signifies that the number of flow adjustments also depend on the separation between the failed flippers. The higher separation between failed flippers requires large number of role change operations to reach convergence. This results higher number of flow adjustment. The result also shows that, the increase in number of flipper failure increases the number of flows required to be rerouted.

C. Effect of Link Failure

To visualize the effect of link failure on data plane operation, we perform similar experiments as mentioned earlier. In this experimental setup, k links are chosen randomly so that there is at least one path between the source to the destination of each flow. We perform the following two experiments by selecting variable number of k links as following. **Experiment 2.a:** The failed links are 1-hop distance apart and **Experiment 2.b:** The failed links are at least 2-hop distance apart. These selected links are disconnected simultaneously to perform the failure experiments. The emulation results shown in Figure 5b reveals that, the convergence time and required number of flow adjustments depends on the number of failed links and separation between the failed links.

VII. Background and Related Works

Traditional SNMP based for network management system [25]–[27] resulted complex and rigid architectures. [28] shows that, network configurations are highly error prone. The error of configuration happens due to the complexity of managing each network devices individually. To reduce the network management overhead, SDN came into existence. Some of the popular SDN control plane approaches are, [29]–[33]. To ensure scalability, SDN control plane for service provider network needs to be distributed. According to Panda *et.al.* [34], it is not possible to ensure strong consistency, availability and partition tolerance simultaneously in case of distributed control platform. Increase in number of controllers increases scalability and management overhead both [35]. On the other hand, reduction of controllers makes the control plane a bottleneck [36]. Therefore, designing of distributed control platform for service provider network is non-trivial. Although, [7], [8] have proposed distributed control plane, fault-tolerance remains an issue in case of distributed control plane. To However, some of the fault resilient distributed control planes are refereed in [37]. Among the existing works, POCO-PLC [18] proposes a Pareto optimal, fault-resilient off-line control plane. However, designing a fault tolerant SDN network management system is non-trivial due to the fact that selecting a recovery strategy might take longer convergence time. These limitations have motivated us to design a dynamic architecture, which can reduce the flow initiation delay and can provide fault tolerance.

VIII. Commonly Asked Questions

Here we discuss the answer of some questions that may arise while reading this paper.

How FLIPPER is different from SDN?

Standard SDN platform uses static role assignments at the time of deployment. Static deployment limits performance of SDN in case of topology changing networks. In case of controller failure, SDN might cease to perform. In such cases, FLIPPER provides more availability than SDN by utilizing dynamic role assignment of flippers.

Can FLIPPER Work in fail-open and fail-close semantic?

Fail-open and fail-close semantics provide partition tolerance in case of fault resilient architecture. Fault-resilience architectures handle specific types of failures. In Section IV, we prove that, the proposed FLIPPER is fault-tolerant. In a fault-tolerant architecture the effect of failure only affects in terms of delay. Therefore, we argue that FLIPPER provides a stronger solution to handle network partitioning problem.

Why our emulation results are not comparable with existing works?

Existing SDN based architectures do not focus on fault tolerance, and most of the cases the solutions are off-line and static deployment based. So, they can not handle arbitrary failure. Once the SDN controllers fail, the switches under the influence of the controllers can not perform data forwarding until a new controller is configured to work with them. Therefore, the term convergence time becomes irrelevant in that context.

IX. Conclusion

In this work, we propose FLIPPER which supports SDN like network management and control, while avoiding the controller bottleneck problem, and supporting a stronger notion of fault tolerance. Built over the existing ONIX architecture, FLIPPER supports a scalable notion of dynamic role adaptation based on a distributed self-stabilizing algorithm. The simulation result shows the benefits of FLIPPER, whereas the emulation over a real testbed conveys the feasibility of FLIPPER implementation over the existing network infrastructure.

X. Acknowledgement

This work is supported by TATA Consultancy Services, India through TCS Research Fellowship Program. We would like to thank Dr. Sushanta Karmakar, for providing help in the mathematical proofs. We also thank anonymous reviewers for improving the presentation of the paper.

References

[1] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[2] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "SIMPLE-fying middlebox policy enforcement using SDN," *ACM SIGCOMM computer communication review*, vol. 43, no. 4, pp. 27–38, 2013.

[3] R. Ozdag, "Intel® ethernet switch FM6000 series-software defined networking," *See goo.gl/AnvOvX*, p. 5, 2012.

[4] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for SDN? implementation challenges for software-defined networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 36–43, 2013.

[5] D. Levin, M. Canini, S. Schmid, and A. Feldmann, "Incremental SDN deployment in enterprise networks," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 473–474.

[6] D. Levin, M. Canini, S. Schmid, F. Schaffert, and A. Feldmann, "Panopticon: Reaping the benefits of incremental SDN deployment in enterprise networks," in *Proceedings of 2014 USENIX Annual Technical Conference (USENIX ATC 14)*, 2014, pp. 333–345.

[7] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: A distributed control platform for large-scale production networks," in *Proceedings of the 9th USENIX Conference on OSDI, 2010*. USENIX Association, 2010, pp. 1–6.

[8] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "Onos: towards an open, distributed sdn os," in *Proceedings of the 3rd HotSDN, 2014*. ACM, 2014, pp. 1–6.

[9] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A network programming language," in *ACM Sigplan Notices*, vol. 46, no. 9. ACM, 2011, pp. 279–291.

[10] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed SDN controller," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 7–12.

[11] G. Yao, J. Bi, Y. Li, and L. Guo, "On the capacitated controller placement problem in software defined networks," *IEEE Communications Letters*, vol. 18, no. 8, pp. 1339–1342, 2014.

[12] K. Phemius, M. Bouet, and J. Leguay, "Disco: Distributed multi-domain SDN controllers," in *Proceedings of 2014 IEEE Network Operations and Management Symposium (NOMS)*. IEEE, 2014, pp. 1–4.

[13] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and OpenFlow: From concept to implementation," *IEEE Communications Surveys Tutorials*, vol. 16, no. 4, pp. 2181–2206, 2014.

[14] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[15] E. W. Dijkstra, "Self-stabilizing systems in spite of distributed control," *Communications of the ACM*, vol. 17, no. 11, pp. 643–644, 1974.

[16] V. Turau, "Linear self-stabilizing algorithms for the independent and dominating set problems using an unfair distributed scheduler," *Information Processing Letters*, vol. 103, no. 3, pp. 88–93, 2007.

[17] M. Gradinariu, S. Tixeuil *et al.*, "Self-stabilizing vertex coloring of arbitrary graphs," *Proceedings of OPODIS 2000*, pp. 55–70, 2000.

[18] D. Hock, M. Hartmann, S. Gebert, T. Zinner, and P. Tran-Gia, "Poco-plc: Enabling dynamic pareto-optimal resilient controller placement in sdn networks," *Proceedings of the 33rd INFOCOM, 2014*, 2014.

[19] "Ns-3.22 - nsnam," https://www.nsnam.org/wiki/Ns-3.22.

[20] "Snap autonomous systems AS-733 data set," http://snap.stanford.edu/data/as.html.

[21] "Snap autonomous systems - oregon-1 data set," http://snap.stanford.edu/data/oregon1.html.

[22] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hotnets,2010*. ACM, 2010, pp. 19:1–19:6.

[23] OVS, "Open vSwitch," http://openvswitch.org/.

[24] "Open vSwitch database server," http://openvswitch.org/support/dist-docs/ovsdb-server.1.txt.

[25] T. Benson, A. Akella, and D. A. Maltz, "Unraveling the complexity of network management." in *NSDI*, 2009, pp. 335–348.

[26] H. Ballani and P. Francis, "Conman: a step towards network manageability," in *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4. ACM, 2007, pp. 205–216.

[27] X. Chen, Z. M. Mao, and J. van der Merwe, "Towards automated network management: Network operations using dynamic views," in *Proceedings of the 2007 SIGCOMM Workshop on Internet Network Management*, ser. INM '07. ACM, 2007, pp. 242–247.

[28] A. Bierman, M. Bjorklund, K. Watsen, and R. Fernando, "Restconf protocol," *IETF draft, work in progress*, 2014.

[29] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, N. McKeown, and S. Shenker, "Sane: A protection architecture for enterprise networks." in *Usenix Security*, 2006.

[30] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," in *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4. ACM, 2007, pp. 1–12.

[31] M. Desai and T. Nandagopal, "Coping with link failures in centralized control plane architectures," in *Proceedings of the 2nd COMSNET, 2010*. IEEE Press, 2010, pp. 79–88.

[32] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: A scalable fault-tolerant layer 2 data center network fabric," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, pp. 39–50, 2009.

[33] Z. Yu, M. Li, X. Yang, and X. Li, "Palantir: Reseizing network proximity in large-scale distributed computing frameworks using sdn," in *Proceedings of the 7th CLOUD, 2014*. IEEE, 2014, pp. 440–447.

[34] A. Panda, C. Scott, A. Ghodsi, T. Koponen, and S. Shenker, "Cap for networks," in *Proceedings of the 2nd HotSDN, 2013*. ACM, 2013, pp. 91–96.

[35] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized?: state distribution trade-offs in software defined networks," in *Proceedings of the 1st HotSDN,2012*. ACM, 2012, pp. 1–6.

[36] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *IEEE Communications Magazine,*, vol. 51, no. 2, pp. 136–141, 2013.

[37] A. S. da Silva, P. Smith, A. Mauthe, and A. Schaeffer-Filho, "Resilience support in software-defined networking: A survey," *Computer Networks*, vol. 92, Part 1, pp. 189 – 207, 2015.