BuDDI: Bug Detection, Debugging, and Isolation Middlebox for Software-Defined Network Controllers

Rohit Abhishek[1], Shuai Zhao[1], Sejun Song[1], Baek-Young Choi[1], Henry Zhu[2], Deep Medhi[1]
[1]University of Missouri-Kansas City, [2]Cisco Systems
{rabhishek, shuai.zhao, songsej, choiby, dmedhi)@umkc.edu, hezhu@cisco.com

***Abstract*—Despite tremendous software quality assurance efforts made by network vendors, chastising software bugs is a difficult problem especially, for the network systems in operation. Recent trends towards softwarization and opensourcing of network functions, protocols, controls, and applications tend to cause more software bug problems and pose many critical challenges to handle them. Although many traditional redundancy recovery mechanisms are adopted to the softwarized systems, software bugs cannot be resolved with them due to unexpected failure behavior. Furthermore, they are often bounded by common mode failure and common dependencies (CMFD). In this paper, we propose an online software bug detection, debugging, and isolation (BuDDI) middlebox architecture for software-defined network controllers. The BuDDI architecture consists of a shadow-controller based online debugging facility and a CMFD mitigation module in support of a seamless heterogeneous controller failover. Our proof-of-concept implementation of BuDDI is on the top of OpenVirtex by using Ryu and Pox controllers and verifies that the heterogeneous controller switchover does not cause any additional performance overhead.**

I. Introduction

Software faults in the operating network systems can cause not only critical system failures [5] but also various unexpected and transient results. Although network vendors have a series of development guidelines, check-pointing facilities, assurance processes, and debugging mechanisms to improve their software reliability, it is commonly accepted that maintaining a bug free network system is impossible. The recent networking paradigm changes towards softwarization and virtualization has increased the detrimental effect of software faults on network functions, protocols, controls, and applications. Furthermore, the increasing open-source and third-party software (TPS) used in Software-Defined Networks (SDN) aggravates software bug problems because vendors may not fully identify all issues during software quality assurance testing. To cope with the software reliability issues, a few software bug handling mechanisms have been proposed. For example, the bug tolerant router design has been proposed in [5], [8] to run multiple diverse copies of virtual router instances. LegoSDN to tolerate SDN application failure [6], [8] focuses on SDN-App failures, fail-stop crashes and byzantine failures. However, software bugs are hard to be resolved with the traditional redundancy-based failure detection and recovery mechanisms alone as software bugs can cause unexpected root cause failures, baffle failure detections, and hinder recovery mechanisms. In addition, some of the deterministic bugs in SDN controllers are often bound by common mode failure and dependencies (CMFD) that bring the system into the same software failures after a fail over. On-line debugging, and especially CMFD resolutions in SDN are still a relatively unexplored area. In this paper, we propose an online software Bug Detection, Debugging, and Isolation (BuDDI) middlebox architecture for SDN controllers. BuDDI consists of a shadow-controller based online debugging facility and a CMFD mitigation module in support of a seamless heterogeneous controller failover. For on-line bug detection and debugging, unlike a traditional $N + 1$ redundancy cluster system, we propose an $N + 2$ load balancing cluster system where components (N) have at least two independent failover components (+2). As illustrated in Figure 1, BuDDI facilitates a CMFD mitigation module by taking advantage of software diversity of the existing heterogeneous controllers. In addition, BuDDI enables a shadow controller that mirrors the active controller functions and turns on a verbose debugging mode for a specific failure module. Eventually, the two failover components will converge into one active controller. If the shadow-controller cannot

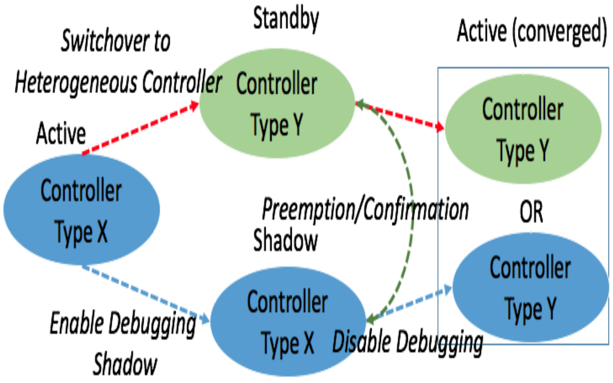

Fig. 1. BuDDI $N + 2$ Reliability

978-3-901882-85-2 © 2016 IFIP

identify a software bug in a given period, it sends a preemption message to the active CMFD module to take over the active role. Otherwise, it will confirm an active role for the CMFD module.

Controller switchover algorithms and shadow controllers debugging facilities are built on the top of OpenVirtex, which provides the facility to create virtual networks and to map them to the physical network. The middlebox acts as proxy between the physical network and the controllers. As a preliminary part of our experiment, we choose two of the heterogeneous controllers (Ryu [2] and Pox [1]) to verify that both the heterogeneous controller switchover and $N + 2$ redundancy mechanism supports do not cause any additional performance overhead in the proposed BuDDI mechanism.

The rest of the paper is organized as follows: Section II describes the related work. In Section III, we review the proposed architecture. We then present the experimental results in Section IV. Finally, we conclude in section V and describe our future work.

II. Related Work

Bug tolerance has been studied with respect to router software. Bug tolerant router design has been proposed in [5], [8]. The idea is to run multiple diverse copies of virtual router instances in parallel. Each of the running router instances running differs from the other, which reduces the possibility of simultaneous failures. These works address the traditional IP networks whereas in our work, we focus on SDN networks.

LegoSDN to tolerate SDN application failure [6], [8] focuses on SDN-App failures, fail-stop crashes, and byzantine failures, and not on controller failures. Here, a re-design of controller architecture has been presented that provides the features to (1) isolate the SDN-Apps from the controller and (2) isolate the SDN-Apps from the network, making the controllers and the network resilient to SDN-App failures. In our work, we focus on controller failure and isolation.

CoVisor [7], a network hypervisor, enables deployment of multiple control applications in a single network, operating on different controller platform, allowing administrators to combine multiple controllers to collaboratively process network traffic. Here, different applications running on multiple controllers are combined to produce a single flow table for each physical switch, and at the same time, the controller's view of topology is also restricted.

Our architecture is built on top of OpenVirtex (OVX). OVX helps to provide address virtualization to keep tenant traffic separate, enabling topology virtualization. OpenVirteX has the facility to create multiple virtual networks. Although OpenVirtex provides the capability to build a virtual network, which points towards the controller, it does not have any logic for fault tolerance or bug detection. Moreover, if the controller fails, the entire network is disconnected as there is no flexibility to change the controller. We address these issues in our proposed architecture.

TABLE I
Switchover Mode Definition

Definition	Explanation
Active Controller	Controller to which the network is connected
Standby Controller	Controller to which switchover takes place after bug is detected in the active controller
Shadow Controller	Controller used for debugging. Copy of the active controller
Heterogenous Controllers	Different types of controllers. E.g.,Pox + Ryu
Homogenous Controllers	Same types of controllers. E.g., Pox + Pox.
Switchover time	Time between bug detection and connection to standby controller

III. BuDDI

Our proposed architecture aims at bug detection, debugging and isolation (BuDDI) for SDN controllers. BuDDI is built on top of OpenVirteX, which is a network virtualization platform that enables operators to create and manage vSDNs [4]. BuDDI provides three main functionalities: (i) detection of any bug in the controller (ii) automatic debugging of the controller for bugs (iii) isolation by switching over the controllers. It works with all four main failure scenarios in the SDN deployment [6]: (i) controller server failure (hardware failure); (ii) controller crashes (bug in the controller code); (iii) network device failures (switch, application server or link failure); and (iv) SDN application (SDN-App) crashes (bugs in the application code). Our work focuses on the controller failures as well as on SDN-App failures. BuDDI is designed in such a way that whenever there is any failure in the Controller or the SDN-App crashes, it detects that the connection to the controller is down, and it switches over the controller. The same time debugging is also started. Important keywords are explained in Table I.

A. Proposed Architecture

The proposed architecture is shown in Figure 2. The middlebox is connected to the controllers via northbound OpenFlow, whereas with the physical network, via southbound OpenFlow.

Ryu, POX, FloodLight, Trema, and OpenDaylight are some of the most commonly used open source controllers [9]. These controllers vary from each other in one way or another, which gives them diversity and supports our claim of using a heterogeneous controller approach. Table II lists the basic differences between the controllers.

For this architecture there are three controllers used: active, standby and shadow controllers. The standby controller is a heterogenous controller, whereas the shadow controller is a homogenous controller. Failure in the controller can be caused due to indeterministic bug in the source code or due to a controller crash. When the active controller is down due to any bug in the SDN app, or due to any hardware failure in the controller, BuDDI switches the control from the active

TABLE II
COMPARISON AMONG CONTROLLERS [9]

	POX	Ryu	Trema	Floodlight	OpenDaylight
REST API	No	Yes(For SB Interface only)	No	Yes	Yes
Language Support	Python	Python-Specific + Message Passing Reference	C/Ruby	Java+Any Langugae that uses REST	Java
Platform Support	Linux, Mac OS, and Windows	Most Supported on Linux	Linux Only	Linux,Mac and Windows	Linux
OpenFlow Support	OF v1.0	OF v1.0-v1.3 and Nicira Extension	OF v1.0	OF v1.0	OF v1.0-v1.3

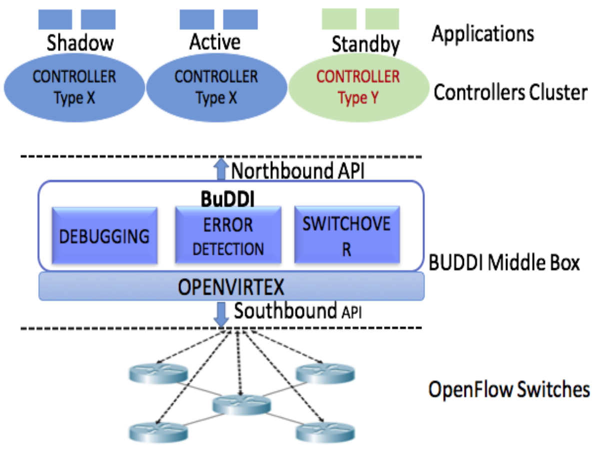

Fig. 2. BuDDI Middlebox Architecture.

controller to the standby controller, which is a heterogenous controller. We use heterogenous controller since different controllers are coded in different languages and there is less probability that the same bug would reoccur. At the same time, the shadow controller, which is homogenous controller, is also switched over, but is used for debugging the failure. The main aim of the shadow controller is to find the root cause of the failure. The shadow controller has the information about the application state, so if the automated debugging process passes the state at which the failure occurred in the active controller, then the shadow controller becomes the new active controller, or else, the standby controller continues to serve as the new active controller.

B. Functional Modules

Our architecture can be divided into three different functional modules as shown in Figure 2: (i) Fault Detection Module (ii) Isolation Module and the (iii) Debugging Module. Each module described below.

Fault Detection Module: The main function of this module is to monitor the applications running on the controller. Using the keep-alive messages exchanged between the datapaths and controllers, we try to monitor the connection between the virtual network and the controllers. If any online software bug causes the application to stop, an error message would be generated. By using this error message and the event log on the controllers, we can check which application on the controller has stopped or has encountered a bug. If there are any hardware failures or any network device failure, the keep-alive messages will not be exchanged and an error message would be generated.

Isolation Module: This module is responsible for switching over to a heterogenous controller and the homogenous controller at the same time. The heterogenous controller will take over as the active controller where as the homogenous controller will be used for debugging purposes. The main reason for a switchover to a heterogenous controller (in spite of applications being controller specific) is that if any non deterministic software bug is found, it would be affected if we switched over to same type of controller. Since different controllers are coded in different coding languages, the probability that we will encounter the same bug in different controllers is less. This module is also responsible for the synchronization of the flow states.

Debugging Module: As we know, applications are controller specific. So it will not be good to use a different controller for long time. So, we use the same type of controller as a shadow controller, which would be used in a debugging mode. An occurrence of a bug in an SDN application will most likely result in the SDN system being down. In order to seamlessly switchover and avoid failure, we use the shadow controller. The main idea is to see if we are getting the same error again in the shadow controller. If we encounter the same error in the same point, it would mean that there is a bug and it would be reported. If we do not encounter the same error in the shadow controller, it would mean that our active controller had some other failure and not a bug. The debugging module aims at detecting liveness bugs.

C. Switchover Procedure

Prior to the network getting connected to the controller, we need to have a copy of the active controller that would act as our shadow controller and also as a standby heterogenous controller. Once the network is operational, it points towards the middlebox. Inside the middlebox, a virtual topology is created. The virtual topology points towards the active controller and the error detection module is initialized to monitor the active controller. When any bug arises in the active controller, an error is generated in the error detection module. Once the error shows up in the error detection module, a notification is sent to the Isolation and Debugging module. Now the Isolation module will switch over the control of the entire network to the standby controller and transfer the flow states from the active controller to the standby controller. At the same time, it also makes a switchover to the shadow controller, which is an exact copy of the active controller, but this shadow

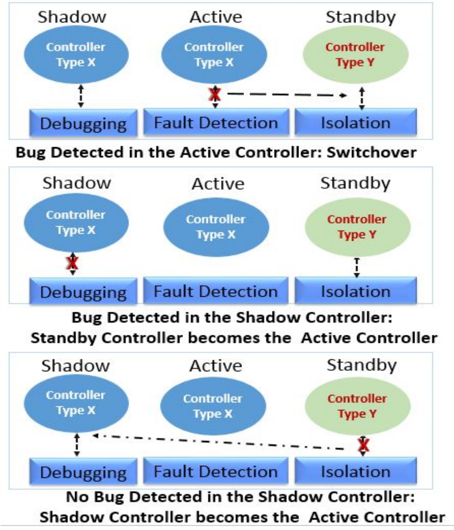

Fig. 3. Shadow Controller Switchover (no bug detected vs bug detected)

controller would be used by the Debugging module to find the root cause of the bug. If while debugging the shadow controller, it does not find the same bug, which occurred in the active controller, it would be mean that there was some other error that occurred in the active controller, which was not necessarily a bug. Then the whole network would be switched over to the shadow controller and it would act as the active controller. If while debugging the shadow controller, the debugging module finds the same error that occurred in the active controller, it would be mean that there is a software bug. At this point, the standby controller will continue to act as the controller. Figure 3 demonstrates the switchover procedure. Introduction of BuDDI adds a latency of 0.2 ms, which is because our architecture is built on top of OpenVirtex that adds the delay to the control channel [3].

IV. IMPLEMENTATION AND PRELIMINARY RESULTS

A. BuDDI Switchover Algorithm

Algorithm 1 shows the proposed BuDDI switchover mechanism. It covers how the switchover happens when any bug is detected and the convergence after the debugging process.

B. Simulation Setup and Results

In our initial experiment, we conducted switchover performance tests between heterogeneous and homogenous controllers by using both Ryu and Pox controllers. Our test environment is a regular virtual machine with 2 GB RAM and a Ubuntu 14.04 operating system. We use Mininet [10] to simulate different network sizes (3, 8, and 11 switches) with linear topology. Each experiment has ping traffic, and was conducted 15 times, and the average values were taken.

Figure 4 compares the switchover time for homogenous and heterogenous controllers in different simulated network sizes. Figure 5 presents two switchover time differences: (i) the switchover time difference between heterogeneous controllers, Ryu to Pox, and homogenous controllers, Ryu to Ryu (a red line); and (ii) the switchover time difference between heterogeneous controllers, Pox to Ryu, and homogenous controllers, Pox to Pox, (a blue line). The graph demonstrates that there is very little switchover time difference between the heterogeneous and homogenous controller switchovers for the example networks we tested.

Algorithm 1: BUDDI Switchover

```
Result: Controller Switched Over
Error Detection module starts monitoring the Active Controller ;
while Bug Detected do
    Switchover Module starts switchover to Standby Controller as the new Active Controller and to Shadow Controller for automated Debugging;
    Transfer buggy state to the shadow controller;
    Transfer flow state to the shadow controller;
    if Debugging Module finds same error in Shadow Controller then
        Continue using Standby Controller as Active controller
    else
        Switchover Module switches over to Shadow Controller as the new Active Controller;
        Transfer the flow state from Standby controller to Shadow controller
    end
end
```

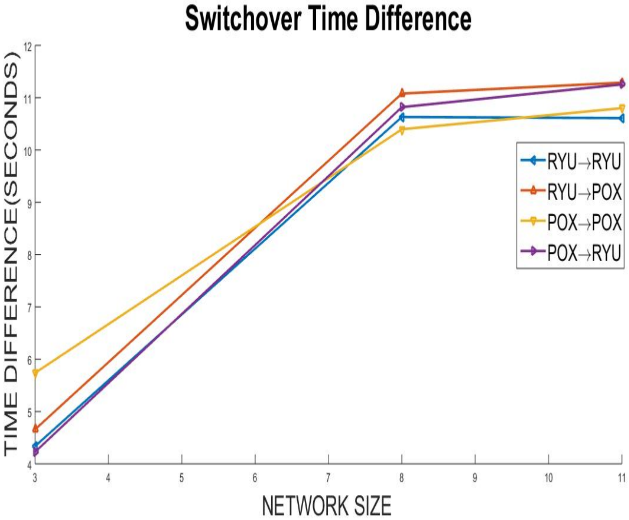

Fig. 4. Heterogenous vs Homogenous Controllers Switchover Time

As we can see from the graph, when the network size increases, the switchover time increases as well. The switchover time is application specific. The architecture has been tested with a layer 2 simple switch. Since our test environment is a regular virtual machine, there is very minimum overhead to run a shadow controller. The result shows promise that BuDDI can effectively support a CMFD module switchover with minimal overhead.

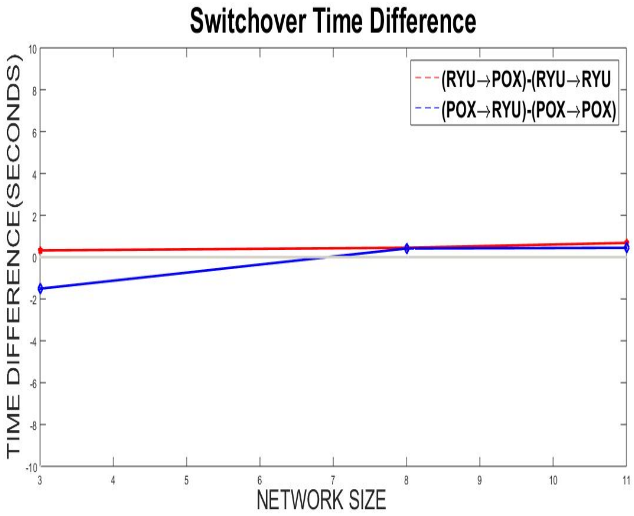

Fig. 5. Heterogenous vs Homogenous Controllers Switchover Time Comparison

V. Conclusion and Future Work

We proposed a novel online software bug detection, debugging, and isolation (BuDDI) middlebox architecture for software-defined network controllers. Unlike the traditional recovery solutions, the proposed solution facilitates on-line based quality assurance, prediction, debugging, and, especially, common cause software failure mode resolutions by using the existing controllers on the top of the open source clustering facility, OpenVirtex.

We verified that BuDDI supports our claim of a heterogeneous controller switchover without causing additional performance overhead. By using the BuDDI algorithms and protocols in future work, we will further investigate additional debugging features and design an automated compatibility matrix over other existing controllers. We also plan to design a facility to transfer the buggy states and modules instead of switching over the entire controllers. We will further test our system by injecting SDN errors both from the software and network . We will also consider a wide range of applications and a high load on the network.

References

[1] Pox wiki https://openflow.stanford.edu/display/onl/pox+wiki.

[2] Ryu sdn framework https://osrg.github.io/ryu-book/en/ryubook.pdf.

[3] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, G. Parulkar, E. Salvadori, and B. Snow. Openvirtex: Make your virtual sdns programmable. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 25–30. ACM, 2014.

[4] A. Al-Shabibi, M. De Leenheer, M. Gerola, A. Koshibe, W. Snow, and G. Parulkar. Openvirtex: A network hypervisor. In *Open Networking Summit 2014 (ONS 2014)*, 2014.

[5] M. Caesar and J. Rexford. Building bug-tolerant routers with virtualization. In *Proceedings of the ACM workshop on Programmable routers for extensible services of tomorrow*, pages 51–56. ACM, 2008.

[6] B. Chandrasekaran and T. Benson. Tolerating sdn application failures with legosdn. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, page 22. ACM, 2014.

[7] X. Jin, J. Gossels, J. Rexford, and D. Walker. Covisor: A compositional hypervisor for software-defined networks. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 87–101, 2015.

[8] E. Keller, M. Yu, M. Caesar, and J. Rexford. Virtually eliminating router bugs. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 13–24. ACM, 2009.

[9] R. Khondoker, A. Zaalouk, R. Marx, and K. Bayarou. Feature-based comparison and selection of software defined networking (sdn) controllers.

[10] M. Team. Mininet: An instant virtual network on your laptop (or other pc), 2012.