Enabling Fast Failure Recovery in OpenFlow networks using RouteFlow

Sachin Sharma  
Department of Computing  
National College of Ireland  
 Dublin, Ireland  
Email: Sachin.Sharma@ncirl.ie

Didier Colle  
Department of Information Technology  
Ghent University - IMEC  
Gent, Belgium  
Email: Didier.Colle@ugent.be

Mario Pickavet  
Department of Information Technology  
Ghent University - IMEC  
Gent, Belgium  
Email: Mario.Pickavet@ugent.be

Abstract—OpenFlow provides a protocol to control a network from an external server called controller. Moreover, RouteFlow presents a framework to run Internet routing protocols in OpenFlow networks by running them in virtual machines or containers. The problem is that OpenFlow networks running RouteFlow do not recover fast from a port failure (e.g., port down event). The failure recovery time is dependent on user configurable parameters and is in seconds. To overcome this problem, we implement a solution in which a port failure of a physical OpenFlow node is detected immediately in its corresponding virtual machine and an immediate action is taken by the routing protocol. Therefore, once a routing protocol running on the corresponding virtual machine detects this failure, it broadcasts the failure in the network and a new failure free path is immediately configured over the OpenFlow network. We implement the proposed solution in an OpenFlow controller and test it over single autonomous and multiple autonomous system scenarios (including OpenFlow and non-OpenFlow scenarios) of the Internet emulated on the virtual wall testbed of the Fed4Fire facility in Europe. The results show that an OpenFlow network can recover from a failure in a short time interval using the proposed solution.

Index Terms—SDN, OpenFlow, OSPF, BGP, RouteFlow

I. INTRODUCTION

Software Defined Networking (SDN) refers to an approach that aims to facilitate simplified network control by enabling programmatic efficient network configurations. It simplifies networks by removing the complex control plane (a part thereof) from the devices (e.g., switches or routers) and deploying it in external servers called controllers. OpenFlow is a de facto SDN protocol to communicate between the control and data plane of network devices [1]. Using OpenFlow, network devices have become much simpler, as they do not have to deal with complex decision making software (control plane). Today, SDN/OpenFlow has been deployed in several networks, e.g., in local-area networks [1], in access networks (such as in access points), in content delivery networks as well as in wide-area networks (such as in Google B4 [2]).

In order to run traditional IP routing protocols in OpenFlow networks, RouteFlow provides a framework with which a physical OpenFlow network (see the network of OpenFlow switches OF-A, OF-B, OF-C and OF-D in Fig. 1) is replicated on the controller using virtual machines (VM) or containers (see VM-A, VM-B, VM-C and VM-D in Fig. 1) and traditional routing protocols are run on the corresponding virtual machines (i.e., the routing protocol of OF-A is run on VM-A, the routing protocol of OF-B is run on VM-B and so on) [3]. Moreover, the decisions made by the traditional IP routing protocols running on VMs are converted to forwarding entries by the controller and installed into switch forwarding tables.

Motivation of using RouteFlow is that traditional routing protocol’s software (such as Quagga) can be used in OpenFlow networks without any modifications [4]. Using RouteFlow, we just need to run the same traditional software (e.g., using Quagga) in virtual machines instead of running them in an OpenFlow switch or router. Moreover, RouteFlow addresses the challenges of vendor lock-in, as it relies on Linux VMs (or containers) and switches/routers implementing the standard OpenFlow protocol. Therefore, it does not rely on a specific vendor [4]. Furthermore, RouteFlow is also important in performing inter-domain routing where each domain is controlled by a separate controller. RouteFlow has been successfully deployed in several network scenarios [5] including multiple autonomous network scenarios implemented in the European funded FP7-CityFlow project, emulating a city of 1 million users [6]. It is currently useful in inter-operating OpenFlow networks with traditional IP networks.

The problem is that OpenFlow networks running RouteFlow do not implement a fast failure recovery solution in networks. For failure recovery, OpenFlow networks depend on routing...
protocols running on a VM to detect a failure and recover from it. Currently, the failure recovery process of a routing protocol is delayed, as a failure in a link of a physical OpenFlow switch is not immediately detected by the corresponding link of the virtual machines. In our proposed solution in this paper, when an OpenFlow switch detects a link failure (e.g., port down failure) in one of the OpenFlow ports, it sends a port-failure message to the controller. The controller then disables the corresponding port of a VM. Therefore, a routing protocol running on the VM detects the failure immediately and broadcasts the failure in the network. A failure free path is then installed in the network. Hence, failure recovery is possible in a short time.

We implement the proposed solution in the controller running RouteFlow and test it in a single Autonomous System (AS) and multiple AS scenarios emulated on the virtual wall testbed of the Fed4Fire facility\(^1\). A single AS scenario is emulated using Mininet (an emulator for SDN experimentation)\(^2\) and multiple AS scenarios are emulated using different nodes of the Fed4Fire testbed by deploying a topology developed in the EU funded CityFlow project to emulate a city of 1 million users. In the multiple AS experiments, there are a mix of OpenFlow and non-OpenFlow networks, and an OpenFlow network communicates with a non-OpenFlow network using traditional standard routing protocols. These multiple AS experiments emulate inter-operation scenarios of OpenFlow and non-OpenFlow networks. A failure is emulated by disabling a port of an OpenFlow switch/router or a non-OpenFlow switch/router in the considered topology. The failure recovery time is compared for the cases when RouteFlow is deployed with and without the proposed solution. The results show that RouteFlow can recover fast when our solution is implemented in it.

The contributions of this paper are following: (1) a fast failure recovery solution for OpenFlow required to inter-operate with non-OpenFlow networks and (2) a large scale experimentation of inter operations of OpenFlow and non-OpenFlow networks on a real testbed such as the Fed4Fire testbed. The rest of the paper is organized as follows: Section II presents the related work. Section III presents the problem and our proposed solution, Section IV describes the emulation environment and results, and finally Section V concludes.

II. RELATED WORK

There are many works exist to perform fast failure recovery in OpenFlow. In [10], restoration and protection techniques are proposed for OpenFlow. It is concluded in this research that OpenFlow can meet carrier-grade fast failure recovery requirements in a large network serving many flows when protection is implemented in the network.

In [12], segment protection is implemented in an OpenFlow based Ethernet network. In this protection scheme, the working and redundant paths (with different priorities) are pre-configured for a segment of the network and when a failure is detected in the working path, an auto reject mechanism (proposed for protection) removes the working path and thereby, enables traffic to be forwarded through the redundant path forwarding entries.

In [13], protection schemes are implemented for each link in a network (instead of for each path). In these schemes, a protection path (and a BFD session) is established for each link and when a failure occurs in a link, traffic is redirected to the corresponding protection path. In addition, control traffic protection schemes are researched in [14]. In these schemes, multiple controllers are used to recover from controller failure scenarios. In [15], a mechanism is proposed to recover from a failure when the fast-failover group type in an OpenFlow switch is not available. For this case, another group type (such as SELECT\(^3\)) is used to implement protection.

In [16], a fast failure recovery solution was implemented for SDN with zero packet loss even though OpenFlow’s fast-failover group feature was not used. This solution is based on OpenState, an extension to OpenFlow that allows a switch to autonomously adapt in a stateful fashion. This reduces the need to depend on remote controllers to take a recovery action. In [17], a multicast tree based failure recovery solution is proposed for OpenFlow, while reducing the Ternary Content Addressable Memory (TCAM) consumption simultaneously.

None of the above works ran a fast failure recovery mechanism keeping in mind inter-operations activities with traditional networks. Therefore, it is difficult to run above mechanisms without any modification in inter-operation situations where there are OpenFlow and non-OpenFlow nodes in networks. In [6], quality of service is achieved in failure conditions in an OpenFlow enabled city network where RouteFlow was used to run traditional routing protocols in OpenFlow. However, the focus of that work was not fast failure recovery. The focus was that high priority flows should receive high precedence even after a failure occurs in the networks.

In this paper, we perform fast failure recovery in OpenFlow networks when traditional Internet routing protocols are run in the network. Therefore, it is possible to run our mechanism in inter-operation situations where there are traditional non-OpenFlow network domains together with OpenFlow network domains.

III. FAST FAILURE RECOVERY FOR OPENFLOW RUNNING ROUTEFLOW

This section describes the problem considered and provides our proposed solution.

A. Problem Description

Fig. 2 shows a failure scenario in an OpenFlow network running RouteFlow. The physical OpenFlow network topology and the virtual environment are the same for Fig. 1 and Fig. 2. The only difference between Fig. 1 and Fig. 2 is that the link between OpenFlow switches OF-A and OF-B is failed in Fig. 2 and OpenFlow switches OF-A and OF-B send port-failure

---

\(^1\)https://www.fed4fire.eu/testbeds/

\(^2\)https://www.opennetworking.org/software-defined-standards/specifications/

\(^3\)SELECT
messages (PORT_STATUS) to the controller. In this case we consider a port failure in which the port becomes down.

The problem is that RouteFlow does not take any action when the controller receives the PORT_STATUS message. Both virtual machines VM-A and VM-B continue to consider the link between them alive until the routing protocols running on the virtual machines (VM-A and VM-B) detect the failure once they stop receiving alive messages from each other. In the OSPF (Open Shortest Path First) routing protocol, this failure detection interval is the router dead interval which is mostly 4 times the hello send interval [8]. Generally, the hello send and router dead intervals are in seconds. Hence, an OpenFlow network takes a very long time to detect the failure and then to recover from it.

B. Our Proposed Fast Failure Recovery Solution

Fig. 3 shows our solution in the same topology as shown in Fig. 2. In our solution, when a link between OpenFlow switches (the link between OF-A and OF-B in Fig. 3) fails, both OpenFlow switches (OF-A and OF-B) send port-failure messages (PORT_STATUS) to the controller. When the controller receives any of the port-failure messages, it gets the corresponding VM ID (Identification number) from RouteFlow and passes this information to the VM by disabling the particular port of the VM. The routing protocol such as OSPF or BGP (Border Gateway Protocol) running on the VM then immediately detects the failure. The failure is then broadcasted over the network according to the routing protocol’s standard and a failure free path is established immediately in the virtual and physical network. Like RouteFlow, our solution prevents vendor lock-in, as we do not change anything in the functionality of the routing protocol. The routing protocols just take the actions defined in their standard on disabling a port of a VM.

IV. EXPERIMENTATION AND EMULATION RESULTS

Experimentation presented in this paper was performed on the virtual wall testbed of the Fed4Fire facility[1]. Fed4Fire is a project under the European union’s programme Horizon 2020 and offers the large federation worldwide of next generation Internet testbeds, which provide open, accessible and reliable facilities to do experiments.

We performed single AS experiments and multiple AS experiments on the Fed4Fire testbed. The single AS experiments were done on a node of the virtual wall testbed facility at IMEC, Gent, Belgium, provided by the Fed4Fire testbeds[3], while the multi AS experiments were done on multiple nodes of the testbed where a separate node was used to emulate an OpenFlow router, non-OpenFlow router or controllers. The following paragraphs explain the single and multiple AS emulation scenarios and results:

A. Single AS Emulation Scenario

A pcgen04 node of the virtual wall testbed[3] was chosen for the single AS emulation. This node has 8 CPU cores of Intel E5-2650v2 processor with 2.6 GHz speed, 48 GB RAM (Random-Access Memory) and one 250 GB hard disk. A single physical CPU core with enabled hyper-threading in this node appears as four logical CPUs to an operating system. Therefore, this node has 32 logical CPUs.

Fig. 4. Emulated Single AS Pan-European Core Topology

3https://www.fed4fire.eu/
We emulated a pan-European core topology (see Fig. 4) using Mininet [7] on the above node of the virtual wall testbed. There are 15 city nodes (see London, Paris etc. in Fig. 4) in the topology. A separate CPU was assigned to each city node including the controller and its virtual machines. The controller has an out-of-band connection with each OpenFlow city node in the network (not shown in Fig. 4). Open vSwitch version 2.3.0 is used for emulations. We used the method given in [9] to automatically configure RouteFlow in the above topology.

We run the OSPF routing protocol in the considered OpenFlow pan-European network topology using RouteFlow. The OSPF hello interval is kept as 1 second and the router dead interval is kept as 4 seconds. It means that OSPF running on a VM sends hello messages to its neighboring VMs after the interval of 1 second. If the OSPF routing protocol running on a VM does not receive a hello message from a neighboring VM until 4 seconds, the link between the VMs is considered to be broken. In the current RouteFlow, the OSPF depends on the router dead interval to declare the failure. However, using our solution, RouteFlow also depends on a PORT_STATUS message to declare a failure.

We describe the failure scenario of our emulation by disabling the link between emulated Milan and Rome nodes (see Fig. 4). Fig. 5 shows the considered emulation scenario using the controller traffic (the traffic coming to or going from the controller node). The traffic is captured using the tcpdump utility available in Linux. At the start of the experiment (from -99 seconds to -70 seconds), there are large spikes in Fig. 5. These spikes are due to messages transmitted in the warm-up period to run the OSPF protocol and configure forwarding entries in the network. These messages are the Address Resolution Protocol (ARP), OSPF and Flow-Mod messages. ARP messages are transmitted to know the MAC addresses, OSPF messages are transmitted to know a path to each node in the network and Flow-Mod messages are transmitted to insert forwarding entries in the network. There are also periodic spikes in the controller traffic. These spikes are due to ECHO, OSPF and ARP messages. These messages are transmitted to check the aliveness of the network.

At second 0, we fail the link between emulated Milan and Rome nodes and we see a large spike in the controller traffic after a few seconds (approximately after 4 seconds). This large spike is due to the traffic sent to recover from the failure and broadcast the failure information in the network. Fig. 5 just shows the control traffic when RouteFlow is used without our proposed solution. However, the traffic in the controller with our solution is similar to as shown in Fig. 5. The only difference is that using our solution, a large spike after second 0 appears quickly compared to as shown in Fig. 5, as our solution depends on the PORT_STATUS message to detect the failure. We compare this in the next section.

Each OpenFlow node in our experiment sends data traffic to all other OpenFlow nodes in the network. The data packet transmit interval to each OpenFlow node is 6 ms and packet size is 1000 bytes. We used an open-source traffic generator known as DITG (Distributed Internet Traffic Generator) [11] to transmit data packets in the network.

B. Single AS Results

Fig. 6 shows traffic destined to Rome on link Rome-Zagreb (see Fig. 4). After link Milan-Rome is taken down, this is the only link connecting Rome. Therefore, after failure detection, all traffic to Rome must follow link Rome-Zagreb. In Fig. 6, we see that traffic to Rome over link Rome-Zagreb increases from 10 packets per 10 ms (before a failure) to 25 packets per 10 ms (after the failure) and then becomes almost constant afterwards. When our solution is not applied in RouteFlow, increase in traffic after the failure takes around 4.160 seconds. However, when our solution is applied, the increase in traffic after the failure takes around 0.040 seconds. This is the failure recovery time, as all traffic to Rome should go through link Rome-Zagreb after the link between Rome-Milan is broken. The results show that RouteFlow using our solution can recover faster compared to RouteFlow when our solution is not applied.

Fig. 7 shows the average failure recovery time in our experiments when different links of the considered topology are broken. The failure recovery time with respect to the number...
A fast failure recovery scenario for inter-operation situations (Content Delivery Network) server. The description about this works to aggregation networks to the core network to a CDN (Term Evolution) and Fibre technologies network scenarios.

CityFlow project to emulate a city of 1 million users taking Fig. 8. This topology is developed in part of the EU funded C. Multiple AS Emulation Scenario

by RouteFlow adds some delay in recovery. external to the network and the virtual environment added up OpenFlow network. This is because the controller is located of the traditional non-OpenFlow network is shorter than the OpenFlow network emulated using the considered pan- increases. increases when the number of affected flows in the network increases.

As shown in Fig. 7, the failure recovery time in RouteFlow (without using our solution) is dependent on the router dead interval when OSPF is used as a routing protocol. We have also done experiments with a large value of router dead interval (i.e., 10 seconds). In these experiments, the failure recovery time was more than 10 seconds. However, RouteFlow using our solution does not depend on the router-dead interval to declare the port failure and hence, it always recovers fast using our solution. Fig. 7 also shows that the failure recovery time increases when the number of affected flows in the network increases.

Fig. 7 also shows the failure recovery time in the non-OpenFlow network emulated using the considered pan-European topology. It shows that the failure recovery time of the traditional non-OpenFlow network is shorter than the OpenFlow network. This is because the controller is located external to the network and the virtual environment added up by RouteFlow adds some delay in recovery.

C. Multiple AS Emulation Scenario

The topology of the multiple AS experiments is shown in Fig. 8. This topology is developed in part of the EU funded CityFlow project to emulate a city of 1 million users taking into account xDSL (Digital Subscriber Line), LTE (Long-Term Evolution) and Fibre technologies network scenarios. The figure shows the Internet infrastructure from access networks to aggregation networks to the core network to a CDN (Content Delivery Network) server. The description about this topology is given in [6]. Using this topology, we emulate a fast failure recovery scenario for inter-operation situations of affected flows are shown in Fig. 7. As the router dead interval in OSPF is kept as 4 seconds, OSPF in RouteFlow detects the failure after 4 seconds. Once the failure is detected, the failure recovery action is taken. Therefore, the failure recovery time in RouteFlow (when our proposed solution is not applied) is more than 4 seconds. In our solution once the failure is detected by the controller, the controller disables the corresponding port immediately and the failure recovery action is taken immediately. Therefore, the failure recovery time is short using our solution.

As shown in Fig. 7, the failure recovery time in RouteFlow (without using our solution) is dependent on the router dead interval when OSPF is used as a routing protocol. We have also done experiments with a large value of router dead interval (i.e., 10 seconds). In these experiments, the failure recovery time was more than 10 seconds. However, RouteFlow using our solution does not depend on the router-dead interval to declare the port failure and hence, it always recovers fast using our solution. Fig. 7 also shows that the failure recovery time increases when the number of affected flows in the network increases.

Fig. 7 also shows the failure recovery time in the non-OpenFlow network emulated using the considered pan-European topology. It shows that the failure recovery time of the traditional non-OpenFlow network is shorter than the OpenFlow network. This is because the controller is located external to the network and the virtual environment added up by RouteFlow adds some delay in recovery.

D. Multiple AS results

between OpenFlow and non-OpenFlow networks. AS1 and AS3 in Fig. 8 emulate OpenFlow networks and AS2 emulate a non-OpenFlow network.

OSPF is used as a routing protocol within an AS network and BGP is used as a routing protocol to communicate between different AS networks. RouteFlow is used in the OpenFlow networks to run the aforementioned routing protocols in their networks. The failure recovery scenario is emulated by forwarding the traffic from an OpenFlow network to a non-OpenFlow network (as shown in Fig. 8 and breaking an OpenFlow or non-OpenFlow link. The rest of the scenario is the same as given for the single AS experiments in the previous subsection.

Fig. 9 shows the average failure recovery time of the experiment performed on the multiple AS topology shown in Fig. 8. In this experiment, the failure recovery time is calculated when an OpenFlow link or a non-OpenFlow link is broken (see Fig. 8). In the case an OpenFlow link is broken, RouteFlow without our solution takes a very long time to recover from the failure because it depends on the router dead interval to declare the failure. However, RouteFlow with the proposed solution recovers fast, as the failure is detected immediately by the virtual environment created by RouteFlow.
Traditional routing protocols such as OSPF and BGP running on a non-OpenFlow router detect the failure immediately if their interface becomes down (e.g., due to the port failure). Therefore, an immediate failure recovery action is taken by a non-OpenFlow router on a port down event. Hence, we see short failure recovery time in Fig. 9 when a non-OpenFlow link is broken. In this case, the failure recovery time is approximately the same irrespective of whether our solution is applied in RouteFlow.

V. CONCLUSIONS

In this paper, we implemented a fast failure recovery solution in OpenFlow networks when RouteFlow is used to run traditional routing protocols such as OSPF and BGP. In our solution, when the controller receives a port failure message from an OpenFlow switch, it immediately disables the corresponding port of a virtual machine in the virtual environment. This starts the recovery process earlier than the time given by user configurable parameters of the routing protocols (such as the router dead interval of OSPF) and therefore, recovery happens faster. Like RouteFlow, our solution also prevents vendor lock-in, as we do not change anything in the functionality of routing protocols.

In addition, we performed two types of extensive experiments on the large scale European Fed4Fire testbed facility: (1) single AS experiments and (2) multiple AS experiments. The results of the single AS experiments show that an OpenFlow network running RouteFlow together with the proposed solution recovers faster compared to when the proposed solution is not integrated with RouteFlow. In addition, the results show that traditional non-OpenFlow networks recover faster than OpenFlow networks when the same routing protocols are run in both networks. Furthermore, inter-operation results of OpenFlow with non-OpenFlow networks in multiple AS experiments show that inter-operation works fine when RouteFlow is applied in OpenFlow networks. In addition, it shows that inter-operated networks recover faster when our solution is applied in RouteFlow.

ACKNOWLEDGEMENT

The work on this project has been supported by the Horizon 2020 Fed4FIREplus project, Grant Agreement No. 723638.

REFERENCES