The Ryu Action Node v1.01

Dzuy Pham, Jason But
Centre for Advanced Internet Architectures, Technical Report 161006A
Swinburne University of Technology
Melbourne, Australia
dhpham@swin.edu.au, jbut@swin.edu.au

Abstract—**In previous work we developed the Ryu Action Node (RAN), a DIFFUSE Action Node capable of deploying prioritisation within a Software Defined Networking (SDN) environment. We now extend the previous RAN implementation, separating it from the Ryu Simple Switch application which limited its capabilities and future development. To improve SDN support within DIFFUSE, additional functionality was added to the RAN to enable deployment of DIFFUSE messages with multiple flow rules and increased OpenFlow version support. Our tests verify that the RAN functions correctly without affecting other concurrently running Northbound Applications.**

I. Introduction

Software Defined Networking (SDN) [1] is a framework used to abstract the functionality of lower-level computer networking into layers. Abstraction introduces flexibility in managing and designing for an agile network, as restrictions on low-level access to switching/routing hardware is lifted and programming the hardware can be achieved through well-defined APIs. The network is split into three layers:

Application Plane - Systems that program the network
Control Plane - Controllers which manage the network
Data Plane - SDN switches which forward the packets

In the SDN network the Control Plane and Data Plane are separated. A centralised controller manages the network while the SDN hardware implements the switching/routing.

A commonly used standard for communication between these two layers is OpenFlow [2, 3], a standardised communications protocol bridging the Control plane and Data plane together.

Northbound applications manage the network through a centralised controller via well-defined APIs. This allows the network to be more dynamic and flexible in responding to changing network conditions.

Ryu [4] is a popular centralised SDN controller with access to OpenFlow APIs. It also includes a Simple Switch and Router Northbound Applications.

DIFFUSE (Distributed Firewall and Flow-shaper Using Statistical Evidence) [5] is a network prioritisation scheme constructed around ML-based techniques. It segregates network traffic by 5-tuple identifier, classifies flow types, and subsequently deploys prioritisation [6].

DIFFUSE has two main modules making up its architecture; the Classifier Node (CN) and Action Node (AN). The CN classifies live traffic, while the AN subsequently implements network traffic prioritisation, DIFFUSE was originally supported on FreeBSD [5] and OpenWRT [7].

The Ryu Action Node (RAN) [6] is a standalone Northbound Application based on the original DIFFUSE AN. The current RAN implementation is an extension of the Ryu Simple Switch application providing both AN capabilities and switching functionality. It listens for DIFFUSE Remote Actions Protocol (RAP) [5] messages, and deploys prioritisation in the SDN Flow Table.

The current RAN implementation has a number of limitations.

1) Restricted compatibility with other applications
2) Unsupported Multi-Message parsing capabilities
3) Limited OpenFlow protocol version support

We have rewritten the RAN as a standalone Northbound Application[1]. Support for the DIFFUSE Multi-Message has been added, as has support for an increased number of OpenFlow protocol versions.

We ran a variety of tests to verify the functionality of the RAN remained intact following the separation of the RAN and Simple Switch application. The RAN was tested with the REST Router and Simple Switch applications to verify it is capable of running as a standalone RYU application. We tested the Multi-Message and Multi-Version features to confirm correct functionality.

This report is organised as follows. Section II outlines the improvements made to the RAN application. Section III describes our testing schedule. Section IV outlines our test and verification results.

[1]Available at http://caia.swin.edu.au/urp/diffuse/sdn

II. RAN Improvements

Improvements were made to the RAN to enhance the compatibility between the DIFFUSE and SDN ecosystems. These improvements aim to facilitate extensibility for future RAN development and in supporting new OpenFlow revisions.

Version 1.0 of the RAN was tightly integrated with Ryu's Simple Switch application, enabling both network switching and prioritisation within the same application. For this version, the RAN was separated from the Simple Switch application to allow for concurrent use with other Northbound Applications.

Version 1.01 of the RAN can be downloaded at http://caia.swin.edu.au/urp/diffuse/sdn.

Support was also added for parallel use with different OpenFlow version SDN switches, increasing the range of compatible SDN devices for the RAN. The final improvement involved adding support for the Multi-Message feature of DIFFUSE, enabling the parsing and processing of packets containing multiple DIFFUSE message flow rules.

A. Northbound Support

Northbound Applications are high level programs that program the Control plane and Data plane through an SDN controller (using an API such as Ryu) in order to provide the intelligence of the SDN system. Multiple Northbound Applications can be used concurrently to control different aspects of an SDN system.

Following our modifications, the RAN is now a standalone Ryu Northbound Application, by designing the RAN to be an independent and unbounded application, a greater range of Ryu Northbound Routing Applications can be used concurrently.

The RAN's Simple Switch integration was removed such that its sole purpose is now to perform DIFFUSE message parsing and priority implementation. This allows us to focus on developing the prioritisation and statistics features of the RAN without being constricted by the structure of the Simple Switch application.

Although the RAN is now separated from the Simple Switch, slight modifications to other Northbound Applications may be required to allow them to run concurrently with the RAN. The RAN is programmed to implement its functionality using 5-tuple flow rules on the SDN switch Flow Table 0. This requires deployment of routing/switching flow rules in the subsequent SDN switch Flow Table 1.

The use of sequential Flow Tables allows network traffic to first be prioritised in the SDN switch Flow Table 0 before being routed/switched by rules installed in Flow Table 1 by another Northbound Application. If both the RAN and other Northbound Routing Applications were to run on the same table, conflicting flow rules may result in compatibility issues.

B. Multi-Message Support

DIFFUSE employs a variety of methods to deliver DIFFUSE messages containing 5-tuple and class information (generated by the Classifier Node) to the RAN. Previously unsupported was the feature to decode multiple information element sets within a single DIFFUSE message and parse the containing the 5-tuple class configuration.

The RAN was upgraded to handle DIFFUSE messages containing a single template with multiple information element sets including the classified class. Each information element set contains 5-tuple information that is parsed by the RAN using the configuration file and will configure the SDN Flow Table as separate flow rules.

C. Multi-Version Support

The RAN application may also encounter a wide of range of OpenFlow SDN switches running different versions of the OpenFlow protocol. Multi-Version support allows for the use of compatible OpenFlow switches with different versions, as well as dealing with unsupported versions. This support allows the introduction of new switches in a running RAN implementation with increased stability and reliability.

If an RAN parses a DIFFUSE message the output flow rules are implemented onto all supported and connected SDN switches. SDN switches running an unsupported version the OpenFlow protocol will not have flow rules installed but the SDN switch can still be controlled by other supported Northbound Applications.

The implementation of Multi-Version support also facilitates additional future OpenFlow versions through the use of a more modular programming structure where new OpenFlow versions can be easily added.

III. RAN Tests

Following the modifications made to the RAN, it is necessary to confirm that the application functions as expected under a variety of test conditions. The RAN relies on other Northbound Applications to perform the routing/switching functionality in an SDN network, our tests ensure that the RAN does not affect the functionality of these applications, thus confirming compatibility between them.

This section describes the set up for each test. All RAN tests use the same `conf.ini` file (see Figure 1) configuration for prioritising flows. Specific prioritisation techniques such as Metering and DSCP [6] were not deployed as the RAN's functionality was being tested rather than the SDN switch's prioritisation ability. These tests are to verify the RAN's capability to create correct flow rules from 5-tuple information, and consequently confirmed by checking the implemented SDN flow rules and gathered statiastics for matching network traffic.

The tests were run on a physical SDN switch (Pica8 P-3295) running PicOS 2.6.3 and connected to a Ryu controller running ryu-manager v4.3. Two hosts running Debian 8.2 were also connected to the SDN switch to generate traffic and verify results.

Tests requiring more than one SDN switch, multiple subnets, or multiple versions, were run on a virtual test bed. The virtual test bed was used in place of the physical test bed as it made topology configuration easier and reduced errors due to poor network configuration. The Multi-Version support test was also run on the virtual test bed due to the virtual test bed supporting a greater number of OpenFlow protocol versions.

The virtual test bed consists of Mininet 2.2.1rc1 running on an Ubuntu 16.04 machine. Mininet is used to virtualise the controller, SDN switch and host machines. The virtual controller utilises ryu-manager v4.0, while the SDN switch ran Open vSwitch v2.5.0. Two hosts were also virtualised as separate running instances of the Ubuntu 16.04 machine.

DIFFUSE messages were sent to both Ryu controllers using the Fake Classifier Node (FCN) [8] on a separate Ubuntu 16.04 machine.

```
[default]
queue = 0

[myclass0]
queue = 0

[myclass1]
queue = 1

[myclass2]
queue = 2
```

Figure 1. RAN Configuration Settings

A. Single Simple Switch

The aim of this test is to verify the RAN's functionality when running concurrently with the Simple Switch application, and checks that the Simple Switch application's functionality remains unaffected when paired with the RAN.

The network configuration for this test (see Figure 2 and Table I) consists of two hosts connected to a single SDN switch, programmed by the Ryu controller. In this configuration the Ryu controller runs the RAN and Simple Switch (OpenFlow v1.3) applications, whilst an FCN sends DIFFUSE messages to the RAN.

The Simple Switch application was responsible for detecting flows and programming the SDN switch Flow Table 0 to forward traffic. The RAN was responsible for receiving DIFFUSE messages from an FCN and deploying prioritisation rules to the SDN switch Flow Table 1.

The RAN's functionality is verified by sending four different DIFFUSE messages to the RAN as per Table II and inspecting the SDN switch Flow Table 0 for correctness. Initially there are two default rules on the SDN switch Table 0. The RAN's functionality is verified if four more flow rules are found to be added onto the SDN switch Table 0 with 5-tuple and class configurations matching our DIFFUSE messages from Table II.

To verify that the Simple Switch application's functionality remains unchanged, the SDN switch Flow Table 1 is checked to contain the instantiated flow rules that allow traffic flow. Live traffic is sent through the switch so statistics can be gathered on the SDN switch Flow Tables 0 and 1 to further verify the Simple Switch's compatibility with the RAN. The live traffic is sent between the hosts using iPerf (UDP and TCP) and ping.

The SDN switch's statistics initially start at zero. The functionality of the SDN switch application is verified if Host 1 can connect to Host 2 which can be seen when the statistics (Number of Packets) by the forwarding flow rules increases, demonstrating that network traffic is being forwarded between the two hosts.

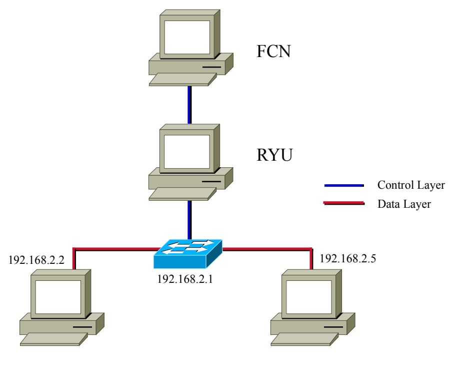

Figure 2. Physical Testbed Single Simple Switch

Table I
SINGLE SIMPLE SWITCH NETWORK CONFIG

	IP	Route	MAC Address
H1	192.168.2.2/24	192.168.2.1	08:00:27:93:18:de
H2	192.168.2.5/24	192.168.2.1	08:00:27:7a:00:e3

Table II
SINGLE SIMPLE SWITCH FCN COMMANDS

SRC IP	DST IP	DST PORT	PROTO	CLASS
192.168.2.2	192.168.2.5	5001	UDP	myclass0
192.168.2.2	192.168.2.5	-	TCP	myclass1
192.168.2.5	192.168.2.2	-	TCP	myclass1
192.168.2.2	192.168.2.5	-	ICMP	myclass2

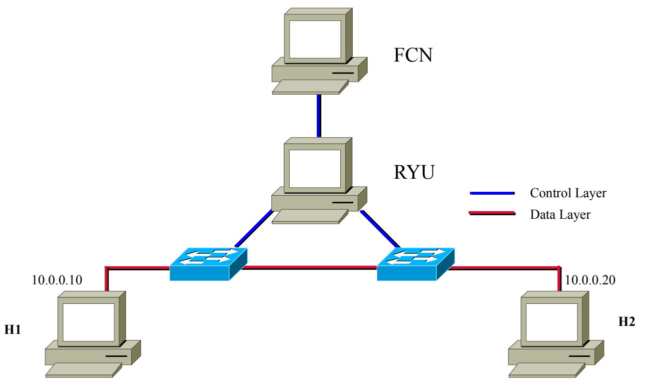

Figure 3. Virtual Testbed Two Connected Simple Switches

B. Multiple Connecting Switches

This test confirms the same functionality as in Section III-A except with a more complex networking scenario. Here we confirm that both the RAN and Simple Switch applications are able to function correctly in a multiple switch environment. The network configuration is shown in Figure 3 and Table III.

As multiple switches are required, this test is executed in the Mininet test environment. As with the test from Section III-A, the Simple Switch application detects flows and programs traffic forwarding on the SDN switch Flow Table 1 on both switches. Similarly, the RAN deploys prioritisation rules from the FCN to SDN Flow Table 0 on both switches.

The verification for the RAN's functionality in this test is similar to Section III-A, however Flow Table 0 on both SDN switches needs to be inspected to confirm the RAN implemented flow rules have been correctly set up. Four DIFFUSE messages as per Table IV are sent to the RAN to be parsed. The RAN's functionality in a multiple connecting switch scenario is verified if four more flow rules matching Table IV are found to have been added to Flow Table 0 on both SDN switches.

Just as in Section III-A, checking that the Simple Switch application's functionality still remains unaffected when running multiple switches is important. The SDN switches' statistics are reset to zero. Live network traffic is sent from Host 1 to Host 2 through the two SDN switches. The Simple Switch application is verified when Host 1 is able connect to Host 2. This is checked through Flow Table 1 on both SDN switches for increasing packet/byte counts in instantiated forwarding rules.

Table III
MULTIPLE CONNECTING SWITCHES NETWORK CONFIG

	IP	Route	MAC Address
H1	10.0.0.10/24	10.0.0.1	ee:7b:c2:4a:d5:b9
H2	10.0.0.20/24	10.0.0.1	3e:2d:c4:e3:59:2d

Table IV
MULTIPLE CONNECTING SWITCHES FCN COMMANDS

SRC IP	DST IP	DST PORT	PROTO	CLASS
10.0.0.10	10.0.0.20	5001	UDP	myclass0
10.0.0.10	10.0.0.20	-	TCP	myclass1
10.0.0.20	10.0.0.10	-	TCP	myclass1
10.0.0.10	10.0.0.20	-	ICMP	myclass2

C. Single Router Same Subnet

This test aims to verify the RAN's functionality when running the RAN concurrently with the REST Router application and to ensure the REST Router application's functionality remains unaffected.

The network configuration for this test (see Figure 4 and Table V) consists of two hosts connected to a single SDN switch and programmed by the Ryu controller. The Ryu controller runs the RAN and REST Router applications, while an FCN sent DIFFUSE messages to the RAN.

As only one SDN switch is required, this test is executed on the physical testbed. Similar to Section III-A, the RAN deploys prioritisation rules from the FCN to SDN Flow Table 0 on both switches.

The virtual router instantiated by the REST Router application detects and deploys routing flow rules on the SDN switch Table 1 for the subnet `192.168.2.1/24`. In this SDN configuration the two hosts are set in the same subnet with IP address `192.168.2.2` and

`192.168.2.5` respectively. This results in host communications through the SDN switch being logically the same as in a Layer 2 network.

The expected outcomes of this test should match those in Section III-A. We expect to see all flows specified by the FCN programmed into Flow Table 0 of the SDN Switch by the RAN. Similarly, we expect that traffic routing rules be installed as expected by the Router application into Flow Table 1, and that the switch correctly routes traffic between the two end hosts. All tests are further verified through the collection of packet statistics for individual rules programmed into both Flow Tables 0 and 1.

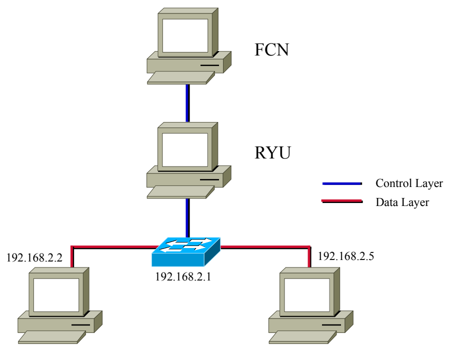

Figure 4. Physical Testbed Single Router Same Subnet

Table V
SINGLE ROUTER SAME SUBNET NETWORK CONFIG

	IP	Route	MAC Address
H1	192.168.2.2/24	192.168.2.1	08:00:27:39:e5:6d
H2	192.168.2.5/24	192.168.2.1	08:00:27:7a:00:e3

Table VI
SINGLE ROUTER SAME SUBNET FCN COMMANDS

SRC IP	DST IP	DST PORT	PROTO	CLASS
-	-	5001	UDP	myclass0
-	-	5001	TCP	myclass1

D. Single Router Different Subnets

This test expands the verification tests from Section III-C by moving the two hosts to different subnets, thereby testing the actual routing component of the REST Router applications in conjunction with the RAN. A single SDN switch is deployed with Flow Tables 0 and 1 programmed as per Section III-C.

The actual network configuration is set up as per Figure 5 and Table VII, the same outcomes are expected as from the previous test.

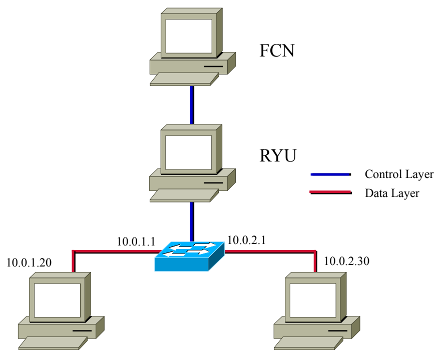

Figure 5. Virtual Testbed Single Router Different Subnet

Table VII
SINGLE ROUTER DIFFERENT SUBNETS NETWORK CONFIG

	IP	Route	MAC Address
H1	10.0.0.10/24	10.0.1.1	86:14:99:18:c4:de
H2	10.0.1.20/24	10.0.2.1	56:bd:0c:f5:5f:22

Table VIII
SINGLE ROUTER DIFFERENT SUBNETS FCN COMMANDS

SRC IP	DST IP	DST PORT	PROTO	CLASS
-	-	5001	UDP	myclass0
-	10.0.0.10	5001	TCP	myclass1
-	10.0.1.20	5001	TCP	myclass1
-	10.0.1.20	-	ICMP	myclass2

E. Multiple Routers Different Subnets

This test achieves the same purpose as the test from Section III-B while using the REST Router Northbound application. This test is designed to confirm that both the RAN and the REST Router applications continue to correctly program the SDN switches in a multiple switch environment. The network configuration for this set up is shown in Figure 6 and Table IX.

In this case, the Router Application continues to operate as per the configurations from the tests in Section III-C and Section III-D. Forwarding rules are expected to be appropriately programmed into Flow Table 1 on both SDN switches. As with the test from Section III-B, we expect the RAN to program prioritisation rules in Flow Table 0 on both switches as per Table X).

Verification is done as per the test in Section III-B, confirming all expected rules are being programmed into all tables on both switches. Further verification is achieved by confirming that traffic flow still functions for flows between the two tests hosts, while also confirming that traffic is being appropriately counted by flow rules programmed into both SDN Flow Tables.

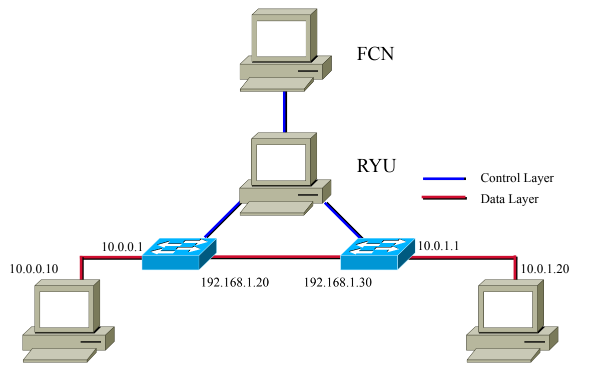

Figure 6. Virtual Testbed Multiple Routers Different Subnets

Table IX
MULTIPLE ROUTERS DIFFERENT SUBNETS NETWORK CONFIG

	IP	Route	MAC Address
H1	10.0.0.10/24	10.0.0.1	f2:e8:9e:73:dd:70
H2	10.0.1.20/24	10.0.1.1	3a:8d:8c:0c:59:06
S1	10.0.0.1		6e:d3:4a:4a:1d:2f
	192.168.1.20	192.168.1.30	ae:22:ea:74:5b:1b
S2	10.0.1.1		a2:3a:c3:f9:bc:14
	192.168.1.30	192.168.1.20	0e:15:d3:67:4f:ad

Table X
MULTIPLE ROUTERS DIFFERENT SUBNETS FCN COMMANDS

SRC IP	DST IP	DST PORT	PROTO	CLASS
10.0.0.10	10.0.1.20	5001	UDP	myclass0
10.0.1.20	10.0.0.10	-	TCP	myclass1
10.0.0.10	10.0.1.20	-	TCP	myclass1
-		-	ICMP	myclass2

F. Multi-Messages Support

The aim of this test is to verify the RAN's new feature to correctly parse and implement multiple 5-tuple flow rules that are contained within a single DIFFUSE message.

The network configuration for this experiment involves the Ryu controller running the RAN application, programming a single SDN switch and a custom Multi-Message DIFFUSE packet sent to the RAN for implementation.

In order to test Multi-Message Support, a custom Multi-Message DIFFUSE packet was created and embedded with multiple SDN switch Flow Table rules.

No traffic was generated through the switch, as we are only confirming that all updates contained within a Multi-Message DIFFUSE packet are correctly parsed and implemented by the RAN into the SDN switch Flow Table. Previous tests confirm that the switch behaves correctly with live traffic.

The structure of the custom Multi-Message DIFFUSE packet includes the same Header and Template structure as found in the single DIFFUSE message [5]. In addition, three sets of information elements are appended to this packet containing the 5-tuple parameters as per Table XI.

To confirm that the Multi-Message is supported, the custom Multi-Message DIFFUSE packet is sent to the RAN. The SDN switch Flow Table 0 is checked for the three installed rules with correct 5-tuple match parameters and class configurations. If the rules are found on the table we have validated that Multi-Message support is functioning as expected.

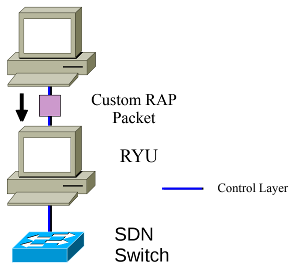

Figure 7. Multi-Message Testbed

Table XI
MULTI-MESSAGE FCN

SRC IP	DST IP	SRC PORT	DST PORT	PROTO	CLASS
10.0.0.1	10.0.0.2	80	8000	UDP	myclass1
192.168.1.1	192.168.1.2	22	30000	TCP	default
130.156.213.64	123.246.18.54	-	-	TCP	myclass5

G. Multi-Version Support

The aim of this test is to ensure that RAN can handle DIFFUSE message implementations when connected to SDN switches running a variety of OpenFlow protocol versions.

The network configuration (see Figure 8) for this test contains three SDN switches running OpenFlow v1.0, v1.3 and v1.4 respectively. Each switch is controlled by a single Ryu controller running the RAN application. The SDN switch running OpenFlow v1.0 represents an unsupported switch, while the SDN switches running Open Flow v1.3 and v1.4 are supported by the RAN. An FCN was used to send DIFFUSE messages to the RAN for implementation.

Four different DIFFUSE messages were sent to the RAN as per Table XII. The 5-tuple of the FCN messages varies between messages.

The RAN installation of the parsed DIFFUSE messages are checked in the SDN switch Flow Tables. The test is verified if the rules are correctly implemented in the supported switches, and no rules are deployed to the OpenFlow v1.0 switch.

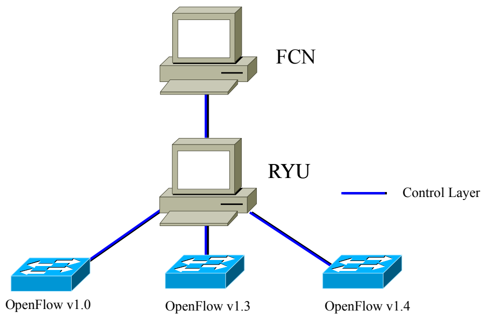

Figure 8. Multi-Version Testbed

Table XII
MULTI-VERSION FCN COMMANDS

SRC IP	DST IP	DST PORT	PROTOCOL	CLASS
-	10.0.0.20	5001	UDP	myclass0
-	10.0.0.10	5001	TCP	myclass1
-	10.0.0.20	5001	TCP	myclass1
-	10.0.0.20	-	ICMP	myclass0

IV. RESULTS

In this section we outline the results seen when running the experiments to verify the functionality of the modifications made to the RAN.

A. Single Simple Switch

The aim of this test is to verify the RAN's functionality when running concurrently with the Simple Switch application, and checks if the Simple Switch application remains unaffected when paired with the RAN.

Table XIII lists the contents of Flow Table 0 and 1 on the SDN switch following the experiment. We can visually confirm that the rules match the DIFFUSE messages from Table II while the corresponding actions match the configuration file from Figure 1. The table also lists the packet and byte count of traffic matching each rule as seen by the switch.

Based on the throughput traffic generated by iPerf and ping, we can confirm that the rules in Flow Table 0 are correctly matching flows as they pass through the switch.

Similar results are seen in Flow Table 1 as programmed by the Simple Switch application. In this case appropriate forwarding rules are programmed and packet/byte counts match the generated traffic.

This confirms not only that both applications are functioning as expected, but that they are able to do so when run concurrently in a single SDN switch deployment.

B. Multiple Connecting Switches

Expanding upon Section IV-A, the aim of this test seeks to determine the RANs functionality when running the RAN concurrently with the Simple Switch application across two connecting SDN switches and checks if the Simple Switch applications functionality remains unaffected.

Table XIV lists Flow Tables 0 and 1 of both switches following our experiment. We can visually confirm that the rules match the DIFFUSE messages from Table IV while the corresponding actions match the configuration file from Figure 1. The table also lists the packet and byte count of traffic matching each rule as seen by the switches.

Throughput traffic from iPerf and ping can be used again to confirm the rules in Flow Table 0 for both switches are correctly matched as they pass through the SDN switches.

Similar results also occur on Flow Tables 1 on both switches as the forwarding rules programmed by the Simple Switch application and packet/byte counts match the throughput traffic.

This confirms both the RAN and Simple Switch application are compatible with multiple concurrently running SDN switches and also verifies they are functioning as expected.

C. Single Router Same Subnet

This test verifies the RAN's functionality when running concurrently with the REST Router application and ensures that the REST Router's functionality remains unaffected.

The contents of Flow Tables 0 and 1 on the Switch after the experiment are displayed in Table XV. We can visually confirm that rules from the DIFFUSE messages are correctly formed as are appropriate rules installed by the REST Router. In both cases, rules are installed in the correct Flow Table.

Proper network functionality is confirmed through successful transfer of test flows and proper packet/byte counts on the associated rules in the Flow Tables.

This confirms that both applications function as expected, and are able to do so when run concurrently on flows within a single subnet on a single SDN Switch deployment.

D. Single Router Different Subnet

This test expands from Section IV-C by increasing the subnets used, thereby testing the routing component of the REST Router applications in addition to the previous verifications.

As in Section IV-C, Table XVI lists the RAN deployed flow rules in Flow Table 0 and the REST Router deployed rules in Flow Table 1.

Flow Tables 0 and 1 in Table XVI also confirm proper network functionality as packet/byte counts increase for matched forwarding rules during network test traffic.

The routing ability of the REST Router application remains unaffected when it runs concurrently with the RAN. Both applications are verified to run correctly in a multi-subnet single SDN Switch environment.

E. Multi Router Different Subnet

Continuing from Section IV-D, additional complexity is added to the network configuration to verify the REST Router application's compatibility with the RAN in a multiple SDN switch scenario. This further validates the functionality of the RAN and REST Router when running concurrently.

Similar to Section IV-C and Section IV-D, Table XVII lists implemented DIFFUSE messages in Flow Table 0 and REST Router forwarding rules in Flow Table 1 in both switches. We visually confirm the RAN's and REST Router rule deployment across both switches.

Network functionality is also verified through inspecting network traffic statistics for corresponding flow rules in Flow Table 0 and 1 as the packet/byte counts increase during live traffic flow.

This confirms that both the RAN and REST Router applications function correctly together in a multiple SDN switches environment.

F. Multi-Message Support

The aim of this test is to verify the RANs new feature to correctly parse and implement multiple 5-tuple flow rules that are contained within a single DIFFUSE message.

The custom DIFFUSE message was sent to the RAN for parsing. The output SDN switch Flow Table 0 (see Table XIX) was inspected and listed the three correctly implemented rules with 5-tuple matching Table XI, including action configurations as per Figure 1.

This confirms the RAN is able to parse multiple rules embedded in a single DIFFUSE message and is able to deploy the rules to the SDN switch correctly.

G. Multi-Version Support

The aim of this test is to ensure the RAN can implement DIFFUSE messages to the correct SDN switches in a network where a variety of SDN switches are running different OpenFlow protocol versions.

FCN messages (as per Table XII) were sent to the RAN. The SDN switches Flow Table 0 (see Table XX) lists the implemented flow rules.

Flow Table 0 verifies that the DIFFUSE messages have been parsed and implemented correctly as the two switches running OpenFlow v1.3 and v1.4 contained the expected flow rules and the SDN Switch running OpenFlow v1.0 contained no FCN rules.

This confirms the RAN is able to identify supported and unsupported SDN switches and can correctly deploy SDN flow rules to only the supported SDN switches.

V. Conclusion

In previous work a Ryu Action Node (RAN) was developed [6] to provide support for deployment of DIFFUSE [5] prioritisation rules in an OpenFlow based SDN network. While functional, this tool contained a number of deficiencies in that:

- Tight integration with the Simple Switch application limited its application and future development
- No support for DIFFUSE RAP messages with multiple rules
- Only OpenFlow v1.3 was supported

In this paper we have demonstrated a new version of the RAN (v1.01), where the Northbound Application has been separated from the Simple Switch Application to create a fully functional stand-alone Ryu application. The application has also been extended to support Multi-Messages and multiple OpenFlow protocol versions.

We have tested our new RAN implementation in conjunction with the Ryu Simple Switch and REST Router applications and verified that it functions correctly both in its respective tasks, but also in ensuring the other application's functionality has not been compromised.

Furthermore, support for multi-rule embedded DIFFUSE messages and multiple OpenFlow protocol versions has been verified in our test network.

This new tool will allow for deployment of DIFFUSE in an SDN controller network with full support for all actions requested by a current DIFFUSE Classifier Node.

The SDN DIFFUSE Ryu Action Node can be found at http://caia.swin.edu.au/urp/diffuse/sdn.

Acknowledgements

The development of the DIFFUSE RAN software has been made possible by a gift from the Cisco University Research Program Fund of the Silicon Valley Community Foundation.

References

[1] M.-K. Shin, K.-H. Nam, and H.-J. Kim, "Software-defined networking (sdn): A reference architecture and open apis," in *ICT Convergence (ICTC), 2012 International Conference on*. IEEE, 2012, pp. 360–361.

[2] OpenFlow Switch Consortium and others, "Openflow switch specification version 1.0.0," 2009.

[3] B. Pfaff, B. Lantz, B. Heller *et al.*, "Openflow switch specification, version 1.3.0," *Open Networking Foundation*, 2012.

[4] Ryu Framework Community. (2016) Ryu sdn framework. [Online]. Available: http://osrg.github.io/ryu/index.html

[5] S. Zander and G. Armitage, "Design of DIFFUSE v0.4 - Distributed Firewall and Flow-shaper Using Statistical Evidence," Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, Tech. Rep. 110704A, 04 July 2011. [Online]. Available: http://caia.swin.edu.au/reports/110704A/CAIA-TR-110704A.pdf

[6] D. Pham and J. But, "Supporting SDN and OpenFlow within DIFFUSE," Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, Tech. Rep. 160429A, 29 April 2016. [Online]. Available: http://caia.swin.edu.au/reports/160429A/CAIA-TR-160429A.pdf

[7] N. Williams and S. Zander, "Real Time Traffic Classification and Prioritisation on a Home Router using DIFFUSE," Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, Tech. Rep. 120412A, 12 April 2012. [Online]. Available: http://caia.swin.edu.au/reports/120412A/CAIA-TR-120412A.pdf

[8] D. Pham and J. But, "Developing a Fake Classifier Node for DIFFUSE," Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, Tech. Rep. 160422A, 22 April 2016. [Online]. Available: http://caia.swin.edu.au/reports/160422A/CAIA-TR-160422A.pdf

Table XIII
PICA8 SWITCH OUTPUT TABLE - SINGLE SIMPLE SWITCH

Table	Priority	Cookie	Match Fields	Actions	Duration	Packets	Bytes
0	20	0x0	udp, nw_src=192.168.2.2, nw_dst=192.168.2.5, tp_dst=5001	set_queue:0, goto_table:1	223.721s	46967	71201972
0	21	0x0	tcp, nw_src=192.168.2.5, nw_dst=192.168.2.2	set_queue:1, goto_table:1	223.678s	21184	1631412
0	21	0x0	tcp, nw_src=192.168.2.2, nw_dst=192.168.2.5	set_queue:1, goto_table:1	223.699s	25173	32951943
0	22	0x0	icmp, nw_src=192.168.2.2, nw_dst=192.168.2.5	set_queue:2, goto_table:1	223.657s	70	10092
0	0	0x0	-	goto_table:1	226.618s	86	22456
1	0	0x0	-	CONTROLLER:65535	226.618s	0	0
1	1	0x0	in_port=10, dl_dst=08:00:27:93:18:de	output:13	188.251s	21259	1653160
1	1	0x0	in_port=13, dl_dst=08:00:27:7a:00:e3	output:10	215.116s	72387	104414255

Table XIV
PICA8 SWITCH OUTPUT TABLE - MULTIPLE SIMPLE SWITCHES

Table	Priority	Cookie	Match Fields	Actions	Duration	Packets	Bytes
0	21	0x0	tcp, nw_src=10.0.0.10 , nw_dst=10.0.0.20	set_queue:1, goto_table:1	220.757s	25375	73647790
0	21	0x0	tcp, nw_src=10.0.0.20 , nw_dst=10.0.0.10	set_queue:1, goto_table:1	220.738s	24290	1603404
0	22	0x0	icmp, nw_src=10.0.0.10, nw_dst=10.0.0.20	set_queue:2, goto_table:1	220.722s	60	5880
0	20	0x0	udp, nw_src=10.0.0.10, nw_dst=10.0.0.20, tp_dst=5001	set_queue:0, goto_table:1	225.554s	49609	75008808
0	0	0x0		CONTROLLER:65535	226.554s	0	0
0	1	0x0		goto_table:1	227.554s	77	8064
1	1	0x0	in_port=1, dl_dst=ee:7b:c2:4a:d5:b9	output:2	228.554s	24359	1611132
1	1	0x0	in_port=2, dl_dst=3e:2d:c4:e3:59:2d	output:1	229.554s	75058	148671914
1	0	0x0		CONTROLLER:65535	230.554s	3	214

Table	Priority	Cookie	Match Fields	Actions	Duration	Packets	Bytes
0	21	0x0	tcp, nw_src=10.0.0.10, nw_dst=10.0.0.20	set_queue:1, goto_table:1	223.283s	25378	73647988
0	21	0x0	tcp, nw_src=10.0.0.20, nw_dst=10.0.0.10	set_queue:1, goto_table:1	223.264s	24290	1603404
0	22	0x0	icmp, nw_src=10.0.0.10, nw_dst=10.0.0.20	set_queue:2, goto_table:1	223.248s	60	5880
0	20	0x0	udp, nw_src=10.0.0.10, nw_dst=10.0.0.20, tp_dst=5001	set_queue:0, goto_table:1	223.303s	49609	75008808
0	0	0x0		CONTROLLER:65535	233.079s	0	0
0	1	0x0		goto_table:1	233.079s	77	8064
1	1	0x0	in_port=2, dl_dst=ee:7b:c2:4a:d5:b9	output:1	210.600s	24359	1611132
1	1	0x0	in_port=1, dl_dst=3e:2d:c4:e3:59:2d	output:2	210.565s	75057	148667480
1	0	0x0		CONTROLLER:65535	233.079s	11	10902

Table XV
PICA8 SWITCH OUTPUT TABLE - SINGLE ROUTER SAME SUBNET

Table	Priority	Cookie	Match Fields	Actions	Duration	Packets	Bytes
0	20	0x0	udp, tp_dst=5001	set_queue:0, goto_table:1	197.427s	49347	74808422
0	21	0x0	tcp, tp_dst=5001	set_queue:1, goto_table:1	197.407s	22122	31157386
0	1	0x0		goto_table:1	204.377s	19949	1546422
0	0	0x0		CONTROLLER:65535	204.377s	0	0
1	0	0x0		NORMAL	204.368s	0	0
1	35	0x1	ip, nw_dst=192.168.2.5	dec_ttl, set_field:08:9e:01:93:a5:21->eth_src, set_field:08:00:27:7a:00:e3->eth_dst, output:10	21.687s	63165	94179306
1	1037	0x1	ip, nw_dst=192.168.2.1	CONTROLLER:65535	197.453s	0	0
1	35	0x1	ip, nw_dst=192.168.2.2	dec_ttl, set_field:08:9e:01:93:a5:21->eth_src, set_field:08:00:27:39:e5:6d->eth_dst, output:13	21.397s	17982	1393025
1	1	0x0	arp	CONTROLLER:65535	204.372s	0	0
1	1	0x0	ip	drop	204.372s	142	12070
1	36	0x1	ip, nw_src=192.168.2.0/24, nw_dst=192.168.2.0/24	NORMAL	197.456s	0	0
1	2	0x1	ip, nw_dst=192.168.2.0/24	CONTROLLER:65535	197.457s	10129	11927829

Table XVI
PICA8 SWITCH OUTPUT TABLE - SINGLE ROUTER DIFFERENT SUBNET

Table	Priority	Cookie	Match Fields	Actions	Duration	Packets	Bytes
0	21	0x0	tcp, nw_src=10.0.0.10, nw_dst=10.0.1.20	set_queue:1, goto_table:1	199.653s	23677	73521250
0	21	0x0	tcp, nw_src=10.0.1.20, nw_dst=10.0.0.10	set_queue:1, goto_table:1	199.636s	22563	1489174
0	22	0x0	icmp, nw_src=10.0.0.10, nw_dst=10.0.1.20	set_queue:2, goto_table:1	199.609s	60	5880
0	20	0x0	udp, nw_src=10.0.0.10, nw_dst=10.0.1.20, tp_dst=5001	set_queue:0, goto_table:1	199.673s	49565	74942280
0	0	0x0		CONTROLLER:65535	204.428s	0	0
0	1	0x0		goto_table:1	204.428s	69	7728
1	1037	0x1	ip, nw_dst=10.0.0.1	CONTROLLER:65535	199.702s	0	0
1	1037	0x2	ip, nw_dst=10.0.1.1	CONTROLLER:65535	199.695s	0	0
1	35	0x1	ip, nw_dst=10.0.0.10	dec_ttl, set_field:52:8b:b7:ee:0b:2b->eth_src, set_field:86:14:99:18:c4:de->eth_dst, output:1	85.624s	22623	1496492
1	35	0x2	ip, nw_dst=10.0.1.20	dec_ttl, set_field:82:67:9e:31:b0:d4->eth_src, set_field:56:bd:0c:f5:5f:22->eth_dst, output:2	8.010s	73301	148469336
1	36	0x1	ip, nw_src=10.0.0.0/24, nw_dst=10.0.0.0/24	NORMAL	199.702s	0	0
1	36	0x2	ip, nw_src=10.0.1.0/24, nw_dst=10.0.1.0/24	NORMAL	199.695s	0	0
1	2	0x1	ip, nw_dst=10.0.0.0/24	CONTROLLER:65535	199.702s	1	74
1	2	0x2	ip, nw_dst=10.0.1.0/24	CONTROLLER:65535	199.696s	1	74
1	1	0x0	arp	CONTROLLER:65535	204.420s	8	336
1	1	0x0	ip	drop	204.420s	0	0
1	0	0x0		NORMAL	204.420s	0	0

Table XVII
PICA8 SWITCH OUTPUT TABLE - MULTI ROUTER DIFFERENT SUBNET

Table	Priority	Cookie	Match Fields	Actions	Duration	Packets	Bytes
0	22	0x0	icmp	set_queue:2, goto_table:1	198.879s	120	11760
0	21	0x0	tcp, nw_src=10.0.0.10, nw_dst=10.0.1.20	set_queue:1, goto_table:1	198.916s	10945	73205218
0	21	0x0	tcp, nw_src=10.0.1.20, nw_dst=10.0.0.10	set_queue:1, goto_table:1	198.897s	9873	651626
0	20	0x0	udp, nw_src=10.0.0.10, nw_dst=10.0.1.20, tp_dst=5001	set_queue:0, goto_table:1	198.933s	49471	74800152
0	0	0x0		CONTROLLER:65535	207.247s	0	0
0	1	0x0		goto_table:1	207.247s	6	1722
1	1037	0x1	ip, nw_dst=10.0.0.1	CONTROLLER:65535	199.014s	0	0
1	1037	0x2	ip, nw_dst=192.168.1.100	CONTROLLER:65535	199.006s	0	0
1	35	0x1	ip, nw_dst=10.0.0.10	dec_ttl, set_field:6e:d3:4a:4a:1d:2f->eth_src, set_field:f2:e8:9e:73:dd:70->eth_dst, output:2	132.536s	9934	659018
1	36	0x1	ip, nw_src=10.0.0.0/24, nw_dst=10.0.0.0/24	NORMAL	199.014s	0	0
1	36	0x2	ip, nw_src=192.168.1.0/24, nw_dst=192.168.1.0/24	NORMAL	199.006s	0	0
1	2	0x1	ip, nw_dst=10.0.0.0/24	CONTROLLER:65535	199.014s	0	0
1	2	0x2	ip, nw_dst=192.168.1.0/24	CONTROLLER:65535	199.006s	0	0
1	1	0x0	arp	CONTROLLER:65535	207.241s	5	210
1	1	0x10000	ip	dec_ttl, set_field:ae:22:ea:74:5b:1b->eth_src, set_field:0e:15:d3:67:4f:ad->eth_dst, output:1	198.979s	60476	148011250
1	0	0x0		NORMAL	207.241s	0	0

Table XVIII
PICA8 SWITCH OUTPUT TABLE - MULTI ROUTER DIFFERENT SUBNET - 2

Table	Priority	Cookie	Match Fields	Actions	Duration	Packets	Bytes
0	22	0x0	icmp	set_queue:2, goto_table:1	199.618s	120	11760
0	21	0x0	tcp, nw_src=10.0.0.10, nw_dst=10.0.1.20	set_queue:1, goto_table:1	199.655s	10945	73205218
0	21	0x0	tcp, nw_src=10.0.1.20, nw_dst=10.0.0.10	set_queue:1, goto_table:1	199.636s	9873	651626
0	20	0x0	udp, nw_src=10.0.0.10, nw_dst=10.0.1.20, tp_dst=5001	set_queue:0, goto_table:1	199.672s	49472	74801664
0	0	0x0		CONTROLLER:65535	207.987s	0	0
0	1	0x0		goto_table:1	207.986s	10	1890
1	1037	0x1	ip, nw_dst=10.0.1.1	CONTROLLER:65535	199.729s	0	0
1	1037	0x2	ip, nw_dst=192.168.1.200	CONTROLLER:65535	199.721s	0	0
1	35	0x1	ip, nw_dst=10.0.1.20	dec_ttl, set_field:a2:3a:c3:f9:bc:14->eth_src, set_field:3a:8d:8c:0c:59:06->eth_dst, output:1	8.233s	60476	148011250
1	36	0x1	ip, nw_src=10.0.1.0/24, nw_dst=10.0.1.0/24	NORMAL	199.729s	0	0
1	36	0x2	ip, nw_src=192.168.1.0/24, nw_dst=192.168.1.0/24	NORMAL	199.720s	0	0
1	2	0x1	ip, nw_dst=10.0.1.0/24	CONTROLLER:65535	199.730s	1	1512
1	2	0x2	ip, nw_dst=192.168.1.0/24	CONTROLLER:65535	199.721s	0	0
1	1	0x0	arp	CONTROLLER:65535	207.981s	9	378
1	1	0x10000	ip	dec_ttl, set_field:0e:15:d3:67:4f:ad->eth_src, set_field:ae:22:ea:74:5b:1b->eth_dst, output:2	199.712s	9934	659018
1	0	0x0		NORMAL	207.980s	0	0

Table XIX
PICA8 SWITCH OUTPUT TABLE - MULTI-MESSAGE SUPPORT

Table	Priority	Cookie	Match Fields	Actions	Duration	Packets	Bytes
0	20	0x0	udp, nw_src=10.0.0.1, nw_dst=10.0.0.2, tp_src=80, tp_dst=8000	set_queue:1, goto_table:1	3.979s	0	0
0	20	0x0	tcp, nw_src=192.168.1.1, nw_dst=192.168.1.2, tp_src=22, tp_dst=30000	set_queue:0, goto_table:1	3.979s	0	0
0	20	0x0	tcp, nw_src=130.156.213.64, nw_dst=123.246.18.54	set_queue:0, goto_table:1	3.978s	0	0
0	0	0x0		CONTROLLER:65535	21.919s	0	0
0	1	0x0		goto_table:1	21.918s	0	0

Table XX
PICA8 SWITCH OUTPUT TABLE - MULTI-VERSION SUPPORT

```
ovs-ofctl -O OpenFlow10 dump-flows s1 NXST_FLOW reply (xid=0x4):
```

Table	Priority	Cookie	Match Fields	Actions	Duration	Packets	Bytes
0	0	0x0		CONTROLLER:65535	102.783s	0	0
0	1	0x0		resubmit(,1)	102.783s	0	0

```
ovs-ofctl -O OpenFlow14 dump-flows s2 OFPST_FLOW reply (OF1.4) (xid=0x2):
```

Table	Priority	Cookie	Match Fields	Actions	Duration	Packets	Bytes
0	21	0x0	tcp, nw_src=10.0.0.10, nw_dst=10.0.0.20	set_queue:1, goto_table:1	3.410s	0	0
0	21	0x0	tcp, nw_src=10.0.0.20, nw_dst=10.0.0.10	set_queue:1, goto_table:1	3.383s	0	0
0	22	0x0	icmp, nw_src=10.0.0.10, nw_dst=10.0.0.20	set_queue:2, goto_table:1	3.363s	0	0
0	20	0x0	udp, nw_src=10.0.0.10, nw_dst=10.0.0.20, tp_dst=5001	set_queue:0, goto_table:1	3.426s	0	0
0	0	0x0		CONTROLLER:65535	148.653s	0	0
0	1	0x0		goto_table:1	148.652s	0	0

```
ovs-ofctl -O OpenFlow13 dump-flows s3 OFPST_FLOW reply (OF1.3) (xid=0x2):
```

Table	Priority	Cookie	Match Fields	Actions	Duration	Packets	Bytes
0	21	0x0	tcp, nw_src=10.0.0.10, nw_dst=10.0.0.20	set_queue:1, goto_table:1	4.127s	0	0
0	21	0x0	tcp, nw_src=10.0.0.20, nw_dst=10.0.0.10	set_queue:1, goto_table:1	4.100s	0	0
0	22	0x0	icmp, nw_src=10.0.0.10, nw_dst=10.0.0.20	set_queue:2, goto_table:1	4.080s	0	0
0	20	0x0	udp, nw_src=10.0.0.10, nw_dst=10.0.0.20, tp_dst=5001	set_queue:0, goto_table:1	4.143s	0	0
0	0	0x0		CONTROLLER:65535	101.912s	0	0
0	1	0x0		goto_table:1	101.912s	0	0