Reliable Optical Networks with ODTN: Resiliency and Fail-over in Data and Control Planes

Andrea Campanella, Boyuan Yan, Ramon Casellas, *Senior Member, IEEE, Member, OSA,* Alessio Giorgetti, Victor Lopez, Yongli Zhao, and Arturo Mayoral.

(Invited Paper)

***Abstract*—This paper reports a technical demonstration showing the use of the ONOS SDN controller for disaggregated transport networks, summarizing the latest developments and results within the Open Networking Foundation (ONF) Open Disaggregated Transport Networks (ODTN) project.**

The demonstration mainly covers the dynamic provisioning of data connectivity services and advanced automatic failure recovery, both at the control and data plane levels. For the provisioning, we demonstrate the usage of open and standard protocols and interfaces. For the recovery part, we first demonstrate, covering the control plane, how ONOS can behave as a logically centralized controller while using multiple coordinated instances for robustness. We show how the different devices remain under control even in the event of a failure of one of such instances, relying on the ATOMIX framework and the dynamic real-time negotiation of device mastership. For the data plane, we demonstrate the capabilities of the controller to perform automatic restoration of optical services.

***Index Terms*—Disaggregated Optical Networks, Software Defined Networking (SDN), Control Plane Resilience, Open and Disaggregated Transport Networks (ODTN), Open Networking Foundation (ONF), Open Network Operating System (ONOS).**

I. Introduction

DISAGGREGATION and modularization are two key concepts in the evolution of future optical networks. A disaggregated model is based on white boxes with open and standard Application Programming Interfaces (APIs), controlled by a third party management entity, preferably open-source. Where, different optical network elements (such as Re-configurable Optical Add-Drop Multiplexers or ROADMs, transponders, line amplifiers, etc.) can be provided by different vendors [1]. Thus, disaggregation breaks the vendor lock-in dependencies enabling more flexible, re-configurable and elastic network architectures and deployments.

A key component in these networks is *the optical domain controller*, which is responsible for control plane functions and network management, such as discovering the topology through multiple protocols and device-specific models, providing service establishment, and continuously manage and monitor the network [2].

In this scenario, besides proprietary controllers owned by device vendors, several open-source initiatives emerged to implement Software Defined Networking (SDN) controllers supporting disaggregated optical networks. Among them, the Open Networking Foundation (ONF) is extending the functionalities of their Open Network Operating System (ONOS) [4], [8] trough the on-going activities of the Open and Disaggregated Transport Networks (ODTN) working group.

It is worth noting that, with the adoption of SDN principles, the overall disaggregated optical network architecture needs to take into account several kinds of failures, both in the data plane (e.g., node failures, fiber cuts and device malfunction), and in the control plane (e.g. the failure of the controller). The demonstration reported in this paper aims at showing how recent developments in the scope of control of disaggregated optical networks aim at addressing these reliability requirements.

This paper extends the work in [3], in which we demonstrate an optical network deployment discovered, managed and controlled by ONOS. Specifically, with respect to [3], this paper provides further details on the implementation of ONOS clustering technology, on the network topology initialization and device feature discovery process, and on the procedures used by the controller to configure an end-to-end optical connection (i.e., *lightpath*). Moreover, this paper considers two additional failure scenarios on the data plane (i.e., loss of configuration, and device failure). Finally a more detailed analysis of the control plane failure is also performed.

Other previous work [4]–[7] includes validation on real hardware. However, the transponders utilized in [4]–[6] are not commercial products but experimental hardware developed for research purposes. Whereas, [5]–[7] do not consider failure scenarios ans the work in [4] only considers, on the data plane, the node failure scenario.

In the demonstration reported in this paper ONOS first discovers the equipment through the NETCONF [9] protocol and the corresponding device YANG [10] models. Then, ONOS provisions a bidirectional lightpath, after receiving a request via its Northbound API.

Once the lightpath is established, the first goal of the demo is to demonstrate the robustness of the ONOS controller. Indeed, the typical ONOS deployment consists of a number of instances; during normal behaviour the instances collaborate operating in a continuously synchronized manner sharing the

A. Campanella is with Open Networking Foundation, Menlo Park, USA, e-mail: andrea@opennetworking.org.

B. Yan is with Open Networking Foundation, Menlo Park, USA and Beijing University of Posts and Telecommunications, China

R. Casellas is with CTTC/CERCA, Optical Networks and Systems Department, Castelldefels, Barcelona, Spain.

A. Giorgetti is with Scuola Superiore Sant'Anna, Pisa, Italy

V. López and A. Mayoral are with Telefónica gCTIO, Madrid, Spain

Y. Zhao is with Beijing University of Posts and Telecommunications, China

Manuscript received December 6, 2019; revised December XX, 2020.

different operations and devices of the network thus increasing the system scalability. In case of failure of one instance the remaining instances take over the control of the underlying infrastructure. This is one of the main differentiating factors of the ONOS controller framework with respect to other SDN controllers. ONOS behaves as a logically centralized controller (thus enabling a seamless implementation of an application ecosystem, as expected), while internally managing a distributed and synchronized set of instances.

The second goal of the demonstration it to show the operations automatically performed to react to data-plane failures. In particular, the demonstration firstly simulates a fiber-cut through a port down command. Thus, the ONOS controller automatically re-provisions the path across the network to account for the failure. The re-provisioning, if a protection path was not previously set-up, takes 500 ms for ONOS to compute the new configuration, and 900 ms for the devices to tune the lasers on the new paths, so a total of 1.4 seconds of traffic disruption for a non protected path is expected. The demo then simulates a device malfunction by erasing existing ROADM media channel configuration directly from the network element. In this case the ONOS controller detects the misalignment and automatically re-configure the device with its correct configuration. Finally, the demo simulates a node failure (i.e., the connection between the controller and a device is interrupted); In this case, lightpaths passing through the failed device are re-routed, while the controller continuously tries to re-establish the connectivity with the lost device.

The paper is organized as follows. First, Sec. II provides an update on recent developments of the ODTN working group. Then, Sec. III details the data plane deployment used in the demonstration. Sec. IV presents the SDN-based control plane architecture, highlighting the decomposition into logical instances. The procedures adopted by the SDN Controller to dynamically obtain the network topology and to provision connection services are respectively described in Sec. V and Sec.VI. Then, Sec. VIII and Sec. VII respectively report data plane and control plane recovery procedures, providing detailed description of the steps adopted by the controller. Finally, Sec. IX draws the conclusion.

II. Recent ODTN activities

The ODTN working group was created at the ONF in 2018 for extending ONOS to support and monitor disaggregated optical networks. A detailed report of the ODTN activities up to summer 2019 can be found in [4]. This section summarizes the most recent ODTN activity. In particular, focus is placed on handling failures and achieving resiliency as described in this paper. New features were also developed, among all we integrated Bit Error Rate (BER) retrieval capabilities from OpenConfig Transponders. Pre- and post- BER Forward Error Correction (FEC) is collected and then exposed through REST APIs, Command Line Interface (CLI) and User Interface (UI). Integrating this parameter allows a more in-depth look at optical link state, enabling ease of diagnostics and analysis. BER is no available over every OpenConfig transponder.

The ODTN project also dedicates effort in expanding the pool of optical equipment that it can control and manage. For OpenConfig based transponders integration and testing was done with Fujitsu 1FINITY T100 [26] and Infinera Groove G30s [25]; for ROADMs, drivers were included in ONOS to control CzechLight equipment [23], as shown at TIP summit 2019. To allow a full end-to-end open source stack on top of white box hardware for packet-optical transponders the Stratum [24] operating system from ONF has been extended to support optical configuration through OpenConfig and gNMI. Such an integration included also drivers in ONOS. Thanks to the described effort all of the capabilities discussed in this paper are offered now with Stratum over the Edgecore Cassini packet-optical transponder.

ODTN has also been extended to be capable of leveraging the GNPy optical simulation and planning tool [22] for path computation across the optical domain. Through the use of GNPy ODTN is now aware of different optical impairments, such as fiber loss or device gain capabilities. The path is selected among the possible ones by using the GSNR value: the path with highest value, thus lowest signal interference, is chosen and configured in the network.

III. Data plane deployment

The network used in the demonstration is comprised of two CASSINI optical transponders and two Reconfigurable Optical Add/Drop Multiplexers (ROADMs). The transponders are modelled as OpenConfig terminal devices [12] and controlled by specific drivers within ONOS that adheres to the OpenConfig YANG model definition [21]. The two transponder are then connected to the ROADMs which in turn are connected through redundant links to support data plane failover. As depicted in Fig. 1, the transponders are two Edgecore AS7716-24SC white box devices equipped with Lumentum CFP2-ACO Coherent Optical Transceivers on the optical side. The transponders, through OCNos from IPInfusion, expose a NETCONF API that models their capabilities and data through the OpenConfig YANG models. The Lumentum ACO card is integrated through a driver with the Transponder Abstraction Interface (TAI) [13] which exposes a high level set of APIs to configure transceiver capabilities. The transponders client ports are attached to emulated end-hosts to generate traffic and measure network state.

The ROADMs are Lumentum ROADM-20 white boxes that expose a NETCONF API described by a not-standard YANG model provided with the device vendor [6], so a specific ONOS drivers that map high level applications commands to operations on the device have been implemented, and is now part of the official ONOS distribution.

IV. Control Plane Architecture

The ONOS SDN Controller is deployed in a scenario with 3 instances, as illustrated in Fig. 2. Every instance runs identical java code in a Java Virtual Machine (JVM) inside its own Docker container. A group of ONOS instances is known as an ONOS cluster. The cluster shares state through a 3 instance ATOMIX partition set [14]. This ATOMIX cluster is

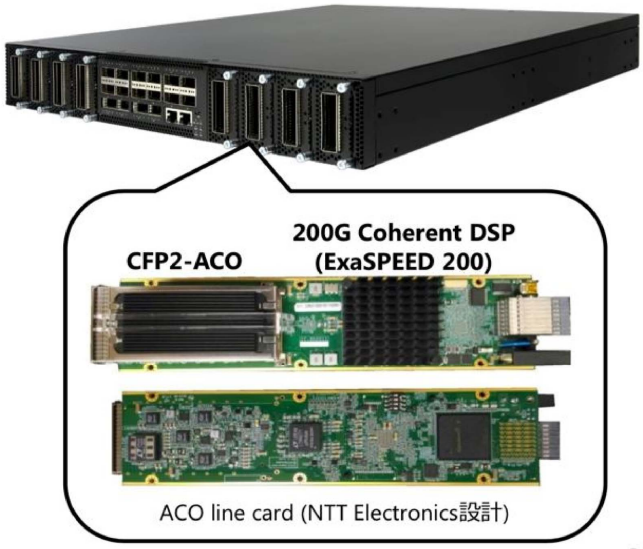

Fig. 1. CASSINI Edge-core AS7716-24SC optical transponder equipped with Lumentum CFP2-ACO transceivers.

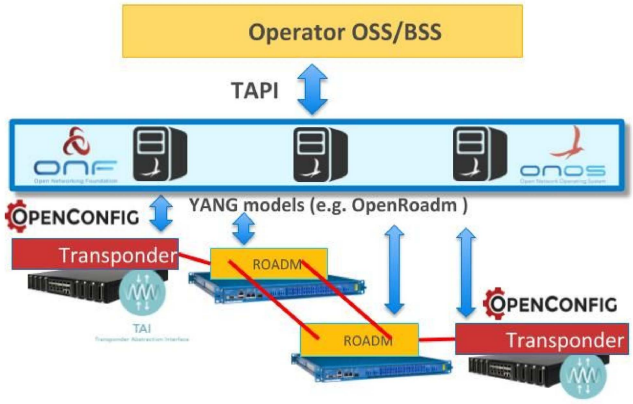

Fig. 2. Control Plane Architecture deploying an ONOS cluster composed of three instances

again deployed in a 3 instance scenario with 3 other Docker containers running a JVM. Both ONOS and ATOMIX are deployed with an odd number of instances to avoid the split brain problem of distributed systems [15].

The described approach results in 6 docker containers for the total deployment. For this demonstration, all the docker containers run over a bare metal server with an 10-core x86 CPU and 64GB memory capability. The server is connected through a separate management network to the network devices. In real deployments a number of servers or a Kubernetes cloud native environment can be used to host each required docker container to avoid single points of failure.

A. Atomix distributed in memory database

In order to resist in case of control plane failure without loss of data ONOS leverages Atomix, a distributed, persistent, in memory database based on the RAFT [16] consensus protocol. Atomix is a fully featured framework for building fault-tolerant distributed systems. Combining ZooKeeper's consistency with Hazelcast's usability and performance, Atomix uses a set of custom communication APIs, a shared Raft cluster, and a multi-primary protocol to provide a series of high-level primitives for building distributed systems and to solve many common distributed systems problems including cluster management, asynchronous messaging, group membership, leader election, distributed concurrency control, partitioning, and replication. Atomix abstracts the distributed aspect by exposing a series of Java APIs very keen to the Collection framework, such as `EventuallyConsistentMap`, `ConsistentMap`, `AtomicCounter` and others. Such easy to use APIs are at the foundation of each subsystem in ONOS.

B. ONOS Distributed stores

All information inside ONOS is saved in distributed stores. Depending on the requirements of a service, the algorithm used to store and distribute data between nodes can have different characteristics (e.g. strongly consistent, eventually consistent, etc.). This is made possible by Atomix exposing Java Collections having such characteristics thus allowing each service's store to implement the appropriate distribution mechanism according to it's data. Historically, the store for Mastership management used Hazelcast's distributed structures as a strongly consistent back-end. Since ONOS 1.4 the Atomix framework is used instead. The stores for network state such as Devices, Links, and Hosts uses an optimistic replication technique complemented by a background gossip protocol to ensure eventual consistency. Simply put, the same subsystems of two different nodes synchronize directly with one another through the Store. The Store only synchronizes the states of the subsystem that it is part of. For example, a DeviceStore only knows about the state of devices, and does not have any knowledge of how host or link information is tracked. A different approach is instead taken with Flow Rules and configuration. Since these are edicts that must be enforced on the network a strongly consistent storage through Atomix is used, demanding the nodes to have knowledge of the state before any operation happens towards the devices, thus allowing controller nodes to fail at any time with no disruption.

Upon ONOS saving any information in the stores, the underlying Atomix receives it and shares it among it's partitions, over a single TCP connection in port 9876, ensuring consistency. In the case targeted in our demonstration, devices, ports, links, intents and flows each have their corresponding distributed stores. Thus, if a failure affecting an ONOS instance occurs, the information being shared through Atomix is not lost, since it is shared and can still be read by the remaining ONOS instances. When a new instance re-connects to the ONOS cluster, it can retrieve all data from the distributed stores in order to synchronize state.

V. Network Topology Initialization and Equipment Discovery

In our particular SDN deployment, the topology information, involving devices' ports and capabilities (transceivers,

```
{
  "devices": {
    "netconf:10.10.255.2:2022": {
      "netconf": {
        "ip": "10.10.255.2", "port": "2022",
        "username": "admin", "password": "admin"
      },
      "basic": {
        "name": "TxRx1", "driver": "terminal-device"
      }
    },
    "netconf:10.10.255.3:2022": {
      "netconf": {
        "ip": "10.10.255.3", "port": "2022",
        "username": "admin", "password": "admin"
      },
      "basic": {
        "name": "TxRx2", "driver": "terminal-device"
      }
    }
  }
}

{
  "links": {
    "netconf:10.10.255.2:2022/10101-netconf:10.10.255.3:2022/10102": {
      "basic" : {
        "type": "OPTICAL", "bidirectional": true, "metric": 1
      }
    },
    "netconf:10.10.255.3:2022/10102-netconf:10.10.255.2:2022/10101": {
      "basic" : {
        "type": "OPTICAL", "bidirectional": true, "metric": 1
      }
    }
  }
}
```

Fig. 3. JSON payload in the REST request to initialize the network topology vision at the ONOS controller, it includes two devices (transponders) and the two connecting links.

physical-channels, identifiers, etc.) as well as links are discovered through OpenConfig interfaces based on NETCONF protocol.

ONOS receives from an external Operation Support Systems (OSS) or Business Support Systems (BSS) a Javascript Object Notation (JSON) encoded request containing the endpoint information of the different devices and the drivers (which defines the protocol/management interface to be used to discover the devices) to use for such devices. Fig. 3 shows an example JSON used to initialize the network topology vision of the ONOS controller including two devices and the connecting links.

Upon receiving the device endpoint information ONOS establishes connection and after a successful discovery (typically involving exchanges with the devices retrieving topological information), topological elements are stored in different ONOS stores:

- ONOS topology store the topology graph consisting on edge and vertices.
- ONOS device store for device and ports information, also available media channels (optical wavelengths) are stored here.
- ONOS Dynamic Configuration Store (DCS), which can be later queried using a diversity of north-bound protocols, e.g TAPI.

The DCS storage is structured following pre-configured data models. In particular, ONF Transport API (TAPI) [17] data models, so TAPI data nodes such as Links, Nodes, Node-Edge-Points and Service Interface Points (SIPs) are exposed to applications and high level API consumers such as network orchestrators or operators' OSS and BSS, through a REST-CONF [18] interface. Other than retrieving SIPs, such interface may also be used to learn about the network topology and issue connectivity service requests, which trigger the establishment of data plane services (ITU-T fixed-grid Optical Channels and/or flexi-grid network media channels.)

A. Detailed Provisioning

Consider the provisioning of a service, involving two transponders and two Lumentum ROADMs. Initially, the devices are provisioned in the ONOS SDN controller, by using a dedicated REST interface (i.e., the interface to be used by external OSS/BSS systems). The devices are thus declared, and ready to be controlled by the SDN Controller. The first part involves the creation of the NETCONF session:

```
16:23:14.606 INFO [NetconfControllerImpl] Creating NETCONF session to netconf
       :10.100.100.93:830 with apache-mina
16:23:14.606 INFO [NetconfControllerImpl] Creating NETCONF session to netconf
       :10.100.100.94:830 with apache-mina
16:23:14.611 INFO [NetconfControllerImpl] Creating NETCONF session to netconf
       :10.100.100.2:830 with apache-mina
16:23:14.611 INFO [NetconfControllerImpl] Creating NETCONF session to netconf
       :10.100.100.3:830 with apache-mina
16:23:14.617 INFO [NetconfSessionMinaImpl] Connecting to netconf:admin@10
       .100.100.3:830 with timeouts C:5, R:5, I:300
16:23:14.620 INFO [NetconfSessionMinaImpl] Connecting to netconf:superuser@10
       .100.100.93:830 with timeouts C:5, R:5, I:300
16:23:14.620 INFO [NetconfSessionMinaImpl] Connecting to netconf:superuser@10
       .100.100.94:830 with timeouts C:5, R:5, I:300
16:23:14.620 INFO [NetconfSessionMinaImpl] Connecting to netconf:admin@10
       .100.100.2:830 with timeouts C:5, R:5, I:300
16:23:14.621 INFO [NetconfSessionMinaImpl] Creating NETCONF session to netconf
       :10.100.100.94:830
16:23:14.621 INFO [NetconfSessionMinaImpl] Creating NETCONF session to netconf
       :10.100.100.2:830
16:23:14.621 INFO [NetconfSessionMinaImpl] Creating NETCONF session to netconf
       :10.100.100.3:830
16:23:14.621 INFO [NetconfSessionMinaImpl] Creating NETCONF session to netconf
       :10.100.100.93:830
```

After the session is established, the SDN Controller can retrieve the details of the devices, notably, device attributes, and data nodes like the vendor, serial number, hardware and software vendors, etc. At this stage the devices are added to the DCS as mentioned above.

```
16:23:14.918 INFO [ClientLineTerminalDeviceDiscovery] Device retrieved details
16:23:14.919 INFO [ClientLineTerminalDeviceDiscovery] VENDOR    SSSA-CNIT
16:23:14.920 INFO [ClientLineTerminalDeviceDiscovery] HWVERSION 1.0.0
16:23:14.920 INFO [ClientLineTerminalDeviceDiscovery] SWVERSION 1.0.0
16:23:14.921 INFO [ClientLineTerminalDeviceDiscovery] SERIAL    610610
16:23:14.921 INFO [ClientLineTerminalDeviceDiscovery] CHASSISID 128
16:23:14.924 INFO [DeviceManager] Local role is MASTER for netconf:10.100.100.3:830
16:23:14.928 INFO [DeviceManager] Device netconf:10.100.100.3:830 connected
16:23:14.959 INFO [TopologyManager] Topology DefaultTopology{time=1319599588167402,
       creationTime=1572967394955, computeCost=531659, clusters=1, devices=1, links
       =0} changed
16:23:14.988 INFO [TopologyManager] Topology DefaultTopology{time=1319599615296802,
       creationTime=1572967394982, computeCost=136662, clusters=1, devices=1, links
       =0} changed
16:23:15.028 INFO [ClientLineTerminalDeviceDiscovery] Device retrieved details
16:23:15.029 INFO [ClientLineTerminalDeviceDiscovery] VENDOR    SSSA-CNIT
16:23:15.030 INFO [ClientLineTerminalDeviceDiscovery] HWVERSION 1.0.0
16:23:15.030 INFO [ClientLineTerminalDeviceDiscovery] SWVERSION 1.0.0
16:23:15.031 INFO [ClientLineTerminalDeviceDiscovery] SERIAL    610610
16:23:15.031 INFO [ClientLineTerminalDeviceDiscovery] CHASSISID 128
16:23:15.035 INFO [DeviceManager] Local role is MASTER for netconf:10.100.100.2:830
16:23:15.036 INFO [DeviceManager] Device netconf:10.100.100.2:830 connected
```

After the initial discovery, OpenConfig and Lumentum ONOS drivers allow the discovery of ports, and port types, which are later modelled in ONOS internal topology model and can be retrieved via a TAPI interface.

```
16:23:15.054 INFO [TopologyManager] Topology DefaultTopology{time=1319599684355829,
      creationTime=1572967395051, computeCost=108143, clusters=2, devices=2, links
      =0} changed
16:23:15.171 INFO [ClientLineTerminalDeviceDiscovery] Parsing Component port-10101
      type oc-platform-types:PORT
16:23:15.176 INFO [ClientLineTerminalDeviceDiscovery] Parsing Component port-10102
      type oc-platform-types:PORT
16:23:15.177 INFO [ClientLineTerminalDeviceDiscovery] Parsing Component port-10103
      type oc-platform-types:PORT
(snip)
16:23:15.220 INFO [ClientLineTerminalDeviceDiscovery] Parsing Component port-10138
      type oc-platform-types:PORT
16:23:15.221 INFO [ClientLineTerminalDeviceDiscovery] Parsing Component port-10139
      type oc-platform-types:PORT
16:23:15.223 INFO [ClientLineTerminalDeviceDiscovery] Parsing Component port-10140
      type oc-platform-types:PORT
16:23:15.604 INFO [LumentumNetconfRoadmDiscovery] Lumentum ROADM20 - discovered
      details:
16:23:15.605 INFO [LumentumNetconfRoadmDiscovery] TYPE       ROADM
16:23:15.606 INFO [LumentumNetconfRoadmDiscovery] VENDOR     Lumentum:ROADM with Twin
       1X20 WSS
16:23:15.606 INFO [LumentumNetconfRoadmDiscovery] HWVERSION 100004
16:23:15.606 INFO [LumentumNetconfRoadmDiscovery] SWVERSION dcian_R2.1.4_136
16:23:15.606 INFO [LumentumNetconfRoadmDiscovery] SERIAL     WBDF17900071
16:23:15.607 INFO [LumentumNetconfRoadmDiscovery] CHASSISID 1
```

Parameters of the device are next retrieved, like port types, tunability restrictions, operational state, etc. The following example shows retrieval of a ROADM port.

```
16:23:16.295 INFO [LumentumNetconfRoadmDiscovery] Lumentum NETCONF - retrieved port
      3001,true,FIBER,0,{portName=Optical Line, entity-description=Optical Line,
      operational-state=normal, output-power=-50.00, optical-los-hysteresis=3.00,
      input-power=-50.00, outvoa-target-attenuation=0.00, outvoa-actual-attenuation
      =17.60, optical-los-threshold=-42.00, optical-loo-hysteresis=3.00, input-low-
      degrade-threshold=-50.00, input-low-degrade-hysteresis=3.00, optical-loo-
      threshold=-50.00}
```

This step concludes the equipment discovery phase (including 5 devices) for which the ONOS controller takes about 1.7 seconds.

VI. Optical Connectivity Establishment

The demo simulates an OSS/BSS that issues a connectivity request through TAPI to ONOS, to obtain end to end connectivity between two client-side ports of the transponders. ONOS processes the request, translates the received request into intents, stores into the distributed intent subsystem store and performs path computation and resource allocation (i.e., wavelength assignment), resulting in specific configurations of Transponders and ROADMs.

Such configurations are stored in the ONOS system in the form of flow rules. For example, for the transceivers optical ports these rules convey the DWDM grid, channel's central frequency and port number, as well as information required to configure a cross-connection between the client side and line side ports. Each device has an ONOS driver that maps flow rules into actual device configuration based on the underlying device data model (i.e., edit-config NETCONF messages are generated with the proper XML code). In particular, for the transponders, OpenConfig constructs are created and sent down to the device through the pre-established NETCONF connection.

As defined by the OpenConfig model, the line-side to client-side cross-connection is installed through a logical channel association, while an optical channel construct with frequency and power for the specific transceiver is to configure the Lumentum ACO card. Within the ONOS intent framework the configuration of the client side and of the line side are respectively mapped into an OpticalCircuit intent and an OpticalConnectivity intent [4].

After the configuration of both transponders and both ROADMs we show that the two hosts connected to the client-side ports of the Transponders can reach each other and exchange traffic.

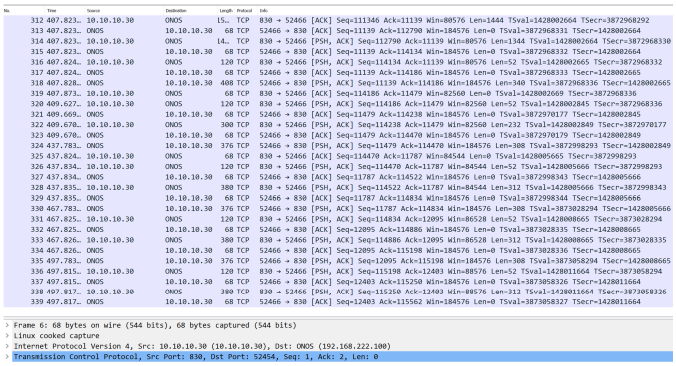

Fig. 4. Wireshark capture of a section of exchanges between the ONOS SDN controller and the NETCONF devices for the provisioning of an optical connection.

A. Detailed Discovery

At the provisioning stage, an OpticalConnectivityIntent has been pushed between the two CASSINI transponder passing thorough the Lumentum ROADM. This is mapped into a Flow Rule that uses Lumentum API to provision the cross-connect, as well as the OpenConfig devices. In particular note the selected OchSignal -35×50 GHz, which is translated into the 191.35 THz wavelength.

```
16:25:15.457 INFO [LumentumFlowRule] Lumentum device, FlowRule coming from
      OpticalConnectivityIntentCompiler
16:25:15.467 INFO [TerminalDeviceFlowRule] TerminalFlowRule built with name
      LineIngress-LinePort-10101-ochSig-Frequency{frequency=191350000000000Hz}
16:25:15.468 INFO [TerminalDeviceFlowRule] TerminalFlowRule built with name
      LineEgress-LinePort-10101-ochSig-Frequency{frequency=191350000000000Hz}
16:25:15.469 INFO [ClientLineTerminalDeviceFlowRuleProgrammable] Sending LINE
      FlowRule to device netconf:10.100.100.3:830 LINE port 10101, frequency
      Frequency{frequency=191350000000000Hz}
16:25:15.469 INFO [ClientLineTerminalDeviceFlowRuleProgrammable] Sending LINE
      FlowRule to device netconf:10.100.100.2:830 LINE port 10101, frequency
      Frequency{frequency=191350000000000Hz}
16:25:15.476 INFO [LumentumNetconfRoadmFlowRuleProgrammable] Lumentum ROADM20 - RPC
      add-connection sent to device netconf:10.10.10.30:830
16:25:15.518 INFO [ClientLineTerminalDeviceFlowRuleProgrammable] OpenConfig added
      flowrule TerminalDeviceFlowRule{id=bf000012c33c7d, deviceId=netconf
      :10.100.100.3:830, priority=100, selector=[IN_PORT:10101, OCH_SIGID:OchSignal
      {-35 x 50.00GHz +/- 25.00GHz}, OCH_SIGTYPE:FIXED_GRID], treatment=
      DefaultTrafficTreatment{immediate=[OUTPUT:10101], deferred=[], transition=
      None, meter=[], cleared=false, StatTrigger=null, metadata=null}, tableId=0,
      created=1572967515459}
16:25:15.457 INFO [ClientLineTerminalDeviceFlowRuleProgrammable] Optical Channel
      Frequency <components xmlns=thttp://openconfig.net/yang/platform'><component
      nc:operation='merge'><name>oc1/0</name><oc-opt-term:optical-channel <xmlns:oc
      -opt-term='http://openconfig.net/yang/terminal-device'></oc-opt-term:config><
      oc-opt-term:frequency> 191650000 </oc-opt-term:frequency> </oc-opt-term:
      config> </oc-opt-term:optical-channel></component></components>
16:25:15.457 INFO [ClientLineTerminalDeviceFlowRuleProgrammable] Optical Channel
      Frequency <components xmlns='http://openconfig.net/yang/platform'><component
      nc:operation='merge'><name>oc1/0</name><oc-opt-term:optical-channel <xmlns:oc
      -opt-term='http://openconfig.net/yang/terminal-device'></oc-opt-term:config><
      oc-opt-term:frequency> 191650000 </oc-opt-term:frequency> </oc-opt-term:
      config> </oc-opt-term:optical-channel></component></components>
16:25:15.525 INFO [ClientLineTerminalDeviceFlowRuleProgrammable] OpenConfig added
      flowrule TerminalDeviceFlowRule{id=bf0000f9652c51, deviceId=netconf
      :10.100.100.2:830, priority=100, selector=[IN_PORT:10101], treatment=
      DefaultTrafficTreatment{immediate=[OCH:OchSignal{-35 x 50.00GHz +/- 25.00GHz
      }, OUTPUT:10101], deferred=[], transition=None, meter=[], cleared=false,
      StatTrigger=null, metadata=null}, tableId=0, created=1572967515455}
16:25:16.754 INFO [LumentumNetconfRoadmFlowRuleProgrammable] Device netconf
      :10.10.10.30:830 applyFlowRules added 1
```

For this step, i.e. forwarding of computed configurations to the devices the ONOS controller takes about 1.2 seconds.

Thus adding the devices to the ONOS controller allowing ONOS to discover the devices details, requesting a service, configuring the devices based on the computed parameters and then removing the connection takes approximately 3 seconds. Fig. 4 shows a (small) part of the involved exchanged messages. A more detailed analysis of the time contributions involved in the connection establishment process can be found in [4].

One can also remove the given configuration across the whole network. The following traces shows the value for the

configuration being set to 0 and then removed by ONOS on the CASSINI devices and the same happening in the Lumentum ROADM.

```
Drop in CASSINI
16:25:15.457 INFO [CassiniIFlowRuleProgrammable] Optical Channel Frequency <
       components xmlns=thttp://openconfig.net/yang/platform'><component nc:
       operation='merge'><name>ocl/0</name><oc-opt-term:optical-channel <xmlns:oc-
       opt-term='http://openconfig.net/yang/terminal-device'></oc-opt-term:config><
       oc-opt-term:frequency>0</oc-opt-term:frequency> </oc-opt-term:config> </oc-
       opt-term:optical-channel></component></components>
16:25:15.457 INFO [CassiniFlowRuleProgrammable] Optical Channel Frequency <
       components xmlns='http://openconfig.net/yang/platform'><component nc:
       operation='merge'><name>ocl/0</name><oc-opt-term:optical-channel <xmlns:oc-
       opt-term='http://openconfig.net/yang/terminal-device'></oc-opt-term:config><
       oc-opt-term:frequency>0</oc-opt-term:frequency> </oc-opt-term:config> </oc-
       opt-term:optical-channel></component></components>
16:25:15.457 INFO [CassiniFlowRuleProgrammable] OPENCONFIG netconf
       :10.100.100.94:830: removedFlowRules removed [2]
16:25:15.457 INFO [CassiniFlowRuleProgrammable] OPENCONFIG netconf
       :10.100.100.93:830: removedFlowRules removed [2]
```

B. Launch Power Configuration

During this demo we have also shown ONOS being capable of configuring launch power at each hop of the network, thus ensuring proper OSNR for the end to end path to be established. Currently the power calculation in done outside of ONOS and is manually inserted for each port in a vendor independent way through the PowerConfig behaviour.

Such behavior abstracts the underlying models. The following trace shows ONOS configuring a power at -6 dBs on the line port of the CASSINI through Openconfig.

```
2019-09-13715:48:38,650 INFO qtp1054806295-7361 [CassiniTerminalDevicePowerConfig]
       246 - org.onosproject.onos-drivers-odtn-driver - 2.3.0.SNAPSHOT I Setting per
        <rpc xmlns--urn:ietf:params:xml:ns:netconf:base:1.0"><edit-confip<target><
       candidate/></target><config><ccmponents xmlns-"http://openconfig.net/yang/
       platform"><component><none>ocl/0</name><optical-channel xmlns-"http://
       openconfig.net/yang/terminal-device"><config><target-output-power>-6<target-
       output-power></config></optical-channel></component></components></config></
       edit-config></rpc>
```

VII. Control plane failure and recovery

Different types of failures can happen at the control plane layer, e.g., a process within one of the ONOS instances is faulty or the supporting physical server has failed. This can lead to subsequent failures and undefined behaviour of the system. In this demonstration, we focus on ONOS deployed in a multi-instance scenario. In such scenario ONOS is capable of handling instance failure by leveraging shared state and changing device mastership.

A. Device Mastership

In ONOS each instance of the cluster has a role against all of the devices, the roles are MASTER, STANDBY or NONE. Every device has a single MASTER instance at a given time, while all the remaining instances stay in STANDBY mode. The Master is elected for a device by the MastershipService through an Atomix based election mechanism. Only the Master of a given device can act upon it by applying configuration.

B. Demo scenario

Initially, and as Fig. 5 shows, 3 docker containers of ONOS control the network together. The nodes are known to each other, please note the $*$ to indicate the current node.

```
onos@root > nodes
id=10.100.100.101, address=10.100.19.101:9876, state=READY, version=2.3.0.a118a0bf7f
       , updated=9m47s ago *
id=10.100.100.102, address=10.100.19.102:9876, state=READY, version=2.3.0.a118a0bf7f
       , updated=9m48s ago
id=10.100.100.103, address=10.100.100.103:9876, state=READY, version=2.3.0.
       a118a0bf7f, updated=9m45s ago
```

ONOS relies on raft partitions to replicate, share and distribute information.

```
onos@root > partitions
-----------------------------------------------------------
Name                  Term                  Members
-----------------------------------------------------------
1                       0                   atomix-1
                                            atomix-2 *
                                            atomix-3
-----------------------------------------------------------
2                       0                   atomix-1
                                            atomix-2 *
                                            atomix-3
-----------------------------------------------------------
3                       0                   atomix-1 *
                                            atomix-2
                                            atomix-3
-----------------------------------------------------------
```

ONOS_1 and ONOS_3 control Transponder_1 and Transponder_2 respectively, having state synchronization between the instances. One ROADM is managed by ONOS_2 while the other by ONOS_3.

```
onos@root > masters
10.192.19.169: 1 devices
  netconf:10.100.100.93:830
10.192.19.170: 1 devices
  netconf:10.100.100.2:830
10.192.19.171: 2 devices
  netconf:10.100.100.3:830
  netconf:10.100.100.94:830
```

At step (1), ONOS_1 docker container is killed resulting in the other instances receiving the notification for that instance with INACTIVE state.

```
09:30:15,193 | INFO  | atomix-cluster-events | AtomixClusterStore              |
       192 - org.onosproject.onos-core-primitives - 2.3.0.SNAPSHOT | Updated node
       10.100.100.101 state to INACTIVE
```

Transponder_1 goes immediately out of management because no NETCONF session is open to it. Mastership of Transponder 1 is then moved to active ONOS_2 via negotiation at step (2). The new master will firstly read the state information of the device in both ONOS traditional datastore and TAPI data tree. Finally at step (3), ONOS_2 establishes a new NETCONF channel with Transponder_1 that was controlled by the deactivated ONOS_1.

```
09:30:15,560 INFO [DeviceManager] Local role is MASTER for netconf:10.100.100.3:830
09:30:15,567 INFO [NetconfControllerImpl] Creating NETCONF session to netconf
       :10.100.100.93:830 with apache-mina
09:30:16,608 INFO [NetconfSessionMinaImpl] Connecting to netconf:superuser@10
       .100.100.93:830 with timeouts C:5, R:5, I:300
09:30:16,623 INFO [NetconfSessionMinaImpl] Creating NETCONF session to netconf
       :10.100.100.93:830
09:30:16,636 INFO [DeviceManager] Device netconf:10.100.100.3:830 connected
```

At this point ONOS_2 is again in full control of Transponder_1 and can react to events and provision configuration on it. The timestamps in the logs show a time of 751ms from ONOS_1 gets recognized as down to when the connection gets re-established. This measurement averages the 700-800 ms time in all the experiments we have done. The 751ms are spent for mastership election, session re-establishment and store notification.

```
onos@root > masters
10.192.19.169: 0 devices
10.192.19.170: 2 devices
  netconf:10.100.100.93:830
  netconf:10.100.100.2:830
10.192.19.171: 2 devices
  netconf:10.100.100.3:830
  netconf:10.100.100.94:830
```

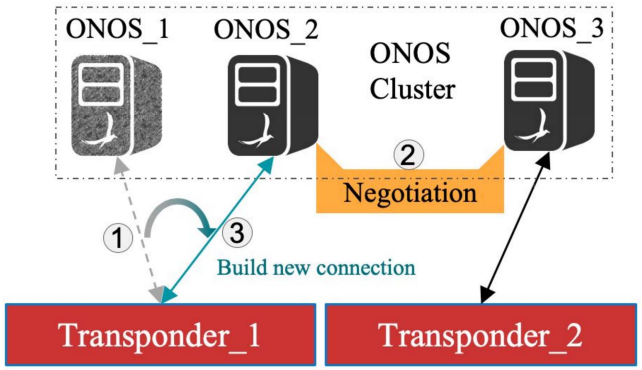

Fig. 5. Mastership movement in ONOS cluster

The whole recovery procedure enables a cluster of ONOS instances to maintain control of the network at any given time with no inconsistent state. The control plane recovery, as the data plane one, is completed automatically without operator intervention.

C. SSH reestablishment

ONOS ensures a continuous connection with the underlying network devices. In our demonstration, the NETCONF SSH session times out according to the device's own configuration. ONOS, upon detecting the loss of the SSH channel proceeds to immediately re-open it from the master instance. This procedure is done automatically and ensures continuous connectivity each device in the network ensuring immediate reaction to failures. The following trace shows ONOS reestablishing connection with the different elements.

```
16:32:47,135 INFO Thread-5008 [NetconfControllerImpl] 215 - org.onosproject.onos-
      protocols-netconf-ctl - 2.3.0.SNAPSHOT - Trying to reestablish connection
      with device netconf:10.128.200.3:830
16:32:47,145 INFO Thread-5009 [NetconfControllerImpl] 215 - org.onosproject.onos-
      protocols-netconf-ctl - 2.3.0.SNAPSHOT - Trying to reestablish connection
      with device netconf:10.128.200.2:830
16:32:47,283 INFO onos-netconfdevicecontroller-connection-reopen-702 [
      NetconfControllerImpl] 215 - org.onosproject.onos-protocols-netconf-ctl -
      2.3.0.SNAPSHOT - Connection with device netconf:10.128.200.2:830 was
      reestablished
16:32:47,289 INFO onos-netconfdevicecontroller-connection-reopen-701 [
      NetconfControllerImpl] 215 - org.onosproject.onos-protocols-netconf-ctl -
      2.3.0.SNAPSHOT - Connection with device netconf:10.128.200.3:830 was
      reestablished
16:42:48,136 INFO Thread-5035 [NetconfControllerImpl] 215 - org.onosproject.onos-
      protocols-netconf-ctl - 2.3.0.SNAPSHOT - Trying to reestablish connection
      with device netconf:10.128.200.3:830
16:42:48,139 INFO Thread-5034 [NetconfControllerImpl] 215 - org.onosproject.onos-
      protocols-netconf-ctl - 2.3.0.SNAPSHOT - Trying to reestablish connection
      with device netconf:10.128.200.2:830
16:42:48,294 INFO onos-netconfdevicecontroller-connection-reopen-704 [
      NetconfControllerImpl] 215 - org.onosproject.onos-protocols-netconf-ctl -
      2.3.0.SNAPSHOT - Connection with device netconf:10.128.200.2:830 was
      reestablished
16:42:48,318 INFO onos-netconfdevicecontroller-connection-reopen-703 [
      NetconfControllerImpl] 215 - org.onosproject.onos-protocols-netconf-ctl -
      2.3.0.SNAPSHOT - Connection with device netconf:10.128.200.3:830 was
      reestablished
```

ONOS also provides an exponential back-off for the retry. Upon a certain number of failed retries ONOS marks the device as OFFLINE, notifying all the listeners and removing it from any future path computation.

VIII. Data plane failure and recovery

The internal architecture of ONOS is layered, where multiple systems offer services to higher level systems and applications. In particular, a flexible and advanced system in

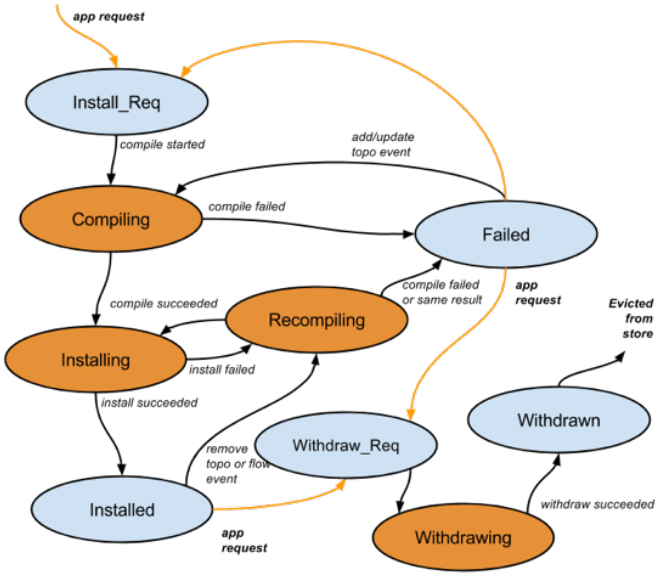

Fig. 6. Intents Finite State Machine, source [11].

ONOS to provision services is based on *Intents*, i.e., high level descriptions of data services [11]. The ODTN project application maps north bound interface requests using the T-API protocol to optical intents, re-using the existing intent framework. More specifically, OpticalCircuit intents are used to model connectivity between two client transponder ports, and OpticalConnectivity intents are used to model connectivity between line port of transponders passing through intermediate ROADMs [4]. By using optical intents, we leverage the automated failure recovery built into the ONOS intent framework, that underlays all the several types of intents provided by the platform, optical included. Fig. 6 shows the finite state machine that ONOS uses for intents installation, compilation and failure detection [11].For the purpose of this work, it is important to note that an *Installed* intent may move to the *Recompiling* state in case of an affecting event on the topology (e.g., a fiber-cut); recompilation may result on the computation of a different path along which the traffic may flow and the desired intent can be recovered. If a new path is present and compiled it will be installed in the network as show previously.

In the following, we detail the different aspects of the demonstration.

A. Loss of Configuration

With the traffic flowing we simulate a loss of configuration in one of the Lumentum ROADM, by manually removing, using its Command Line Interface (CLI), the ONOS installed cross-connection. This emulates a device reboot, a loss of configuration or a mis-configuration in which a given service is removed from the device. ONOS has periodic polling and automated checking of established configurations (flow rules). Thus, the loss of configuration event is picked up by ONOS during its periodic state reconciliation. During the demo we set such period at 5s, but it can be brought even lower to achieve optimal performance, at the cost o putting more

strain on the network. After ONOS picks up the absence of the expected configuration (flow) in the ROADM it immediately sends a request to re-install the same configuration.

```
16:43:48,101 | INFO | FlowRuleDriverProvider-0 | FLowRuleManager I 247 - org.
       onosproject.onos-drivers-lumentum - 2.3.0. SNAPSHOT | Missing flow rule on
       netconf:10.100.100.2:830
16:43:48,479 | INFO | FlowRuleDriverProvider-0 | Lumentum-FlowRule I 247 - org.
       onosproject.onos-drivers-lumentum - 2.3.0. SNAPSHOT | Lumentum device,
       FlowRule coming from OpticalConnectivityIntentCompiler
16:42:48,601 | INFO | FlowRuleDriverProvider-0 |
       LumentumNetconfRoadmFlowRuleProgrammable | 247 - org.onosproject.onos-drivers
       -lumentum - 2.3.0. SNAPSHOT | Lumentum ROAM-20 - RPC add-connection sent to
       device netconf:10.100.100.2:830
```

The provisioning follows the same procedure as in Sec. V-A. This operation can take at most the configured period of state reconciliation (e.g. 5s) plus O(500)ms for ONOS to recompute the path as shown in the previous logs to add to the amount for the device to save and re-apply the configuration. As expected the recovery time is directly related to the polling frequency. Faster recovery is possible at the expenses of increased message rate of the polling.

A further measurement is presented though a snippet of the ping command

```
ping -D localhost | while read row
do
  awk '{ sub(/[0-9]{10}/, strftime("%Y-%m-%d %H:%M:%S", substr($0,2,10))) }1' <<< "
       $row"
done
```

such command gives out a millisecond timestamp per each ping and allows us to corroborate the measurements obtained by the ONOS logs.

```
[2020-01-21 16:43:47.030] 64 bytes from 10.0.0.5: icmp_seq=28407 ttl=64 time=0.463
       ms
[2014-01-21 16:43:48.057] 64 bytes from 10.0.0.5: icmp_seq=28408 ttl=64 time=0.591
       ms

[2014-11-10 16:43:49.456] 64 bytes from 10.0.0.5: icmp_seq=28425 ttl=64 time=1024 ms
[2014-11-10 16:43:50.353] 64 bytes from 10.0.0.5: icmp_seq=28426 ttl=64 time=0.955
       ms
```

It can be noted that the total amount of time is 1.4s, the deletion of the ROADM state was issued at 16:43:48.140, thus it took ONOS a 400 ms to recognize the fault, then the 100 ms to recompute and reinstall. The device took 900 ms to re-tune the laser after the configuration push from ONOS.

B. Fiber-cut

With the traffic flowing we simulate a fiber cut. The fiber cut is simulated by switching one of the ROADM's ports to a down state via the device Command Line Interface (CLI), completely separate from ONOS. Such port-down event is recognized by ONOS that in turn marks the link as failed, broadcasting a LINK_DOWN event. The link event gets parsed by the optical path computation module inside the Intent subsystem that, with the updated topology recomputes the path (i.e., using remaining devices and links). Such recomputation is automatically triggered by the LINK_DOWN event, which is processed by the controller with no human intervention. Upon re-computing the path according to the available topology and selected algorithm ONOS installs the required rules in the devices along the path. The provisioning follows the same approach of Sec. V-A.

Same measurements of times based on pings as shown in VIII-A have been done for the fiber-cut scenario resulting in a 1.9 seconds of disruption. The following capture shows how a total disruption time of 1.9 seconds in the case of fiber-loss, with 200 ms spent in signaling the loss to ONOS, 800 ms to re-compute the path and the expected 900 ms for lasers to be re-tuned.

```
[2020-01-21 16:50:21.030] 64 bytes from 10.0.0.5: icmp_seq=28407 ttl=64 time=0.474
       ms
[2014-01-21 16:50:22.057] 64 bytes from 10.0.0.5: icmp_seq=28408 ttl=64 time=0.566
       ms

[2014-11-10 16:50:23.956] 64 bytes from 10.0.0.5: icmp_seq=28425 ttl=64 time=1057 ms
[2014-11-10 16:50:24.952] 64 bytes from 10.0.0.5: icmp_seq=28426 ttl=64 time=0.977
       ms
```

C. Device Failure

The same mechanism is used in case of device failure. ONOS tries to avoid the failed resource like in the previous case. In particular, when the ONOS controller loses the TCP/SSH connection to a device it considers such device to be in a failed state. Device disconnection triggers a DEVICE_DOWN event that triggers the recovery of intents traversing the failed device. ONOS tries to re-use as much as possible of the existing path, configuring on this fallback path the same wavelength for ports and links. All of the event handling, path computation, fail-over scenarios is done in the ONOS intent subsystem. Also in the case of DEVICE_DOWN event the provisioning happens as in Sec. V-A. In this scenario, the recovery time is also dependent on the actual mechanism to detect the liveliness of the device. Relying of default mechanisms associated to the transport protocols (TCP, SSH) the detection of a failed device can take several seconds or minutes. Using Keep-Alives (directly if the protocol supports such feature or polling the device using operations in order to obtain a reply) can reduce this time but again at the expenses of increased rate of messages.

The case of a device failure yields recovery measurements close to the fiber cut because the procedure in ONOS is the same.

IX. Conclusion

This demonstration has shown the use of the ONOS SDN controller to provision data connectivity services across a disaggregated optical network with real hardware exposing common and open data models. Such deployment enables advanced recovery, both at the control and data planes. For the former, the recovery leverages ONOS distributed stores and offers high availability through Atomix, providing a level of robustness against failures of the control processes. For the latter, ONOS offers failure detection mechanisms that immediately react to network failures re-configuring the overall network to continuously provides end-to-end services with a disruption of 1.9 seconds in the case of data plane failures and no signal disruption in case of control-plane failures.

This demonstration showcased the first example of a resilient control plane for optical networks achieved through the use of open source software, open APIs and open source device models. The chosen APIs, protocols and models reflect also the current requirements from service providers and offerings of the Optical industry.

Overall, this work showed the feasibility of the selected approach while further work is required to evaluate the scalability

of the system with a realistic number of network devices and a realistic traffic matrix.

ACKNOWLEDGMENT

This work has been done under the ONF framework with an open community and the code lives under the APACHE 2.0 license. This work has been partially funded by the European Commission through the H2020-ICT-2016-2 METRO-HAUL project (G.A. 761727) and the Spanish Ministry MINECO project AURORAS (TEC RTI2018-99178-B-I00). Work supported by the BUPT Ph.D. Students Short Term Exchange Program.

REFERENCES

[1] E. Riccardi, P. Gunning, Ó. G. de Dios, M. Quagliotti, V. López, and A. Lord, *An Operator view on the Introduction of White Boxes into Optical Networks,* J. Light. Technol. vol. 36, no. 15, pp. 3062-3072 (2018).

[2] R. Casellas, R. Martínez, R. Vilalta, and R. Muñoz, *Control, Management, and Orchestration of Optical Networks: Evolution, Trends, and Challenges,* J. Light. Technol. vol. 36, no. 7, pp. 1390-1402 (2018).

[3] A. Campanella, Boyuan Yan, R. Casellas, A. Giorgetti, V.r Lopez, and A. Mayoral, *Reliable Optical Networks With ODTN, Resiliency and Failover In Data Plane And Control Plane",* in Proc. European Conference on Optical Communication (ECOC) - Demo Session, Dublin, Sept. 2019.

[4] A. Giorgetti, A. Sgambelluri, R. Casellas, R. Morro, A. Campanella, and P. Castoldi, *Control of Open and Disaggregated Transport Networks using Open Network Operating System (ONOS),* J. Opt. Commun. Netw., vol. 12, no. 2, pp. A171-A181 (2020).

[5] R. Morro et al., *Automated end-to-end Carrier Ethernet Provisioning Over a Disaggregated WDM Metro Network with a Hierarchical SDN Control and Monitoring Platform*, in Proc. European Conference on Optical Communication (ECOC) - Demo Session , Rome, Sept. 2018.

[6] A. Sgambelluri et al., *Fully disaggregated ROADM white box with NETCONF/YANG control, telemetry, and machine learning-based monitoring*, in Tech. Dig. Optical Fiber Communication Conference (OFC) - Demo Session, San Diego, 2018.

[7] *Open Networking Foundation (ONF), TIP Summit 2019* https://www.opennetworking.org/events/tip-summit19/ Accessed: 13 Feb. 2020.

[8] *Open Network Operating System*, https://github.com/opennetworkinglab/onos accessed 27 Nov. 2019.

[9] R. Enns, ed. *Network Configuration Protocol NETCONF*, IETF Request for Comments 6241, June 2011.

[10] M. Bjorklund, Ed., *The YANG 1.1 Data Modeling Language*, IETF Request for Comments 7950, August 2016.

[11] A. Campanella, *Intent Based Network Operations*, in Tech. Dig. Optical Fiber Communication Conference (OFC), San Diego, 2019.

[12] *OpenConfig Project*, http://www.openconfig.net, Accessed: 12 Apr. 2019.

[13] *Disaggregated Transponder Chip Transport Abstraction Interface* https://github.com/Telecominfraproject/oopt-tai, Accessed: 12 Apr. 2019.

[14] *Atomix framework*, https://atomix.io, Accessed: 20 Nov. 2019.

[15] A. S. Muqaddas, P. Giaccone, A. Bianco, G. Maier, *Inter-controller traffic to support consistency in ONOS clusters,* IEEE Trans. Netw. Service Manag. vol 14, no. 4, pp. 1018-1031 (2017).

[16] R. Hammer, Sheng Liu, L. Jagadeesan, M. R. Rahman, *Death by Babble: Security and Fault Tolerance of Distributed Consensus in High-Availability Softwarized Networks*, in Proc. Network Softwarization (NetSoft), Paris, Jun. 2019.

[17] *Open Networking Foundation (ONF), Transport API project* https://wiki.opennetworking.org/display/OTCC/TAPI Accessed: 12 Apr. 2019.

[18] *RFC8040: 'RESTCONF Protocol'*, 2017.

[19] *ConfD, management agent software framework for network elements* https://www.tail-f.com/confd-basic/ Accessed: 2019-4-12.

[20] *The Metro-haul project* http://metro-haul.eu.

[21] *ODTN Drivers* https://github.com/opennetworkinglab/onos/tree/master/drivers/odtn-driver/src/main/java/org/onosproject/drivers/odtn.

[22] J. Auge, G. Grammel, E. le Rouzic, V. Curri, G. Galimberti, and J. Powell, *Open optical network planning demonstration*, in Tech. Dig. Optical Fiber Communication Conference (OFC), San Diego, 2019.

[23] *Czechlight ROADMs* https://czechlight.cesnet.cz/en/open-line-system/sdn-roadm.

[24] *Stratum white box operating system* https://www.opennetworking.org/stratum/.

[25] *Groove G30 Transponder* https://www.infinera.com/products/groove-g30-network-disaggregation-platform.

[26] *Fujitsu 1FINITY T100 Transponder* https://www.fujitsu.com/us/products/network/technologies/1finity-t100/.

Andrea Campanella Andrea Campanella, 27 from Milan, Italy, is employed as a Member of Technical staff (MTS) at Open Networking Foundation (ONF). Andrea obtained a Master's degree (110/110) in computer science at the Public University of Milan in December 2016 after a Bachelor degree in Digital communication in July 2014. The master degree program and the final thesis work focused on computer networks and SDN, with specific attention to southbound protocols and paths to production. Andrea started working with ONF through an internship in October 2015. Andrea is currently the leader of the Optical Disaggregated Transport Network (ODTN) Reference Design and Exemplar Platform, acting also as the bridge between the ONF ODTN community and TIP's OOPT group. Andrea also worked on P4 solution, Trellis hardening and tools. He is currently part of the first team working on configuration for the new μ platform. Andrea also sits on the ONOS Technical Steering Team. Andrea has done tutorials, co-authored papers and presented demos at several conferences both for optical, OFC, ECOC and for packet, e.g. Netsoft.

Boyuan Yan received his B.E. in Communication Engineering from Beijing University of Posts and Telecommunications (BUPT) in 2015. He is currently pursuing a Ph.D. at BUPT. His research interests include software defined optical networking, open optical network, and network resource allocation with machine learning.

Ramon Casellas Ramon Casellas (SM'12) graduated in 1999 from UPC, Barcelona and from the ENST, Paris, with an Erasmus student exchange program grant. He completed a PhD degree in 2002 and worked at the ENST as an Associate Professor. In March 2006, he joined the CTTC Optical Networking Dept., where he is currently holding a Senior Researcher position. His research interest areas include Traffic Engineering; Control and Management of Optical Transport Networks (GMPLS/PCE, SDN and NFV). He has been involved in several international , national and industrial R&D projects and has co-authored 5 book chapters, over 200 journal and conference papers and 5 IETF RFCs. He is an ONF contributor and member of the ONF ODTN project.

Alessio Giorgetti received the Ph.D. degree from Scuola Superiore Sant'Anna (SSSA), Pisa, Italy, in 2006. In 2007, he has been visiting scholar at Centre for Advanced Photonics and Electronics, University of Cambridge, UK. Currently, he is an Assistant Professor at SSSA. His research interests include: optical network architecture and control plane, industrial network design, software defined networking. He is author of more than 100 publications including international journals, conference proceedings, and patents. He is an ONF contributor and member of the ONF ODTN project.

Víctor López (M.Sc. 2005 - Ph.D. 2009) is a Technology Expert at Systems and Network Global Direction in Telefónica gCTIO. He works on the global IP and transport processes of the Telefonica group. Moreover, he is involved in the research projects as well as in the definition of the Telefonica network guidelines for IP/optical segment. He serves as co-chair at the Open Optical Packet Transport group in the Telecom Infra Project. He started his carrier in Telefónica I+D as a researcher (2004). He moved to Universidad Autonoma de Madrid in 2006, where he obtained the Ph.D. He became an Assistant Professor at UAM (2009), working on metro-backbone evolution projects. In 2011, he joined Telefonica I+D as a Technology Specialist working on funded research projects from the Telefonica group and the European Commission. He has co-authored more than 200 publications, 5 patents and contributed to IETF and ONF. Moreover, he is the editor of the book Elastic Optical Networks: Architectures, Technologies, and Control (308 pages, Springer 2016).

Yongli Zhao is currently an associate professor at the Institute of Information Photonics and Optical Communications (IPOC) at Beijing University of Posts and Telecommunications (BUPT). He received a B.S. in Communication Engineering and a Ph.D. in Electromagnetic Field and Microwave Technology from BUPT, respectively, in 2005 and in 2010. He has published more than 150 journal and conference articles. His research focuses on software defined optical networks, flexi-grid optical networks, network virtualization, and so on.

Arturo Mayoral is a Technology Expert in Transport Networks at the Global Systems and Network department of Telefónica GCTIO. He received the Ph.D. degree in telecommunications engineering from the Universitat Politècnica de Catalunya (UPC) in 2019. His research interests include optical network architecture, Software Defined Networking (SDN) and control plane technologies. He is author or co-author on over 50+ journals and conference papers. He graduated in Telecommunications Engineering by the Universidad Autónoma de Madrid in 2013 and he started his professional career in 2012, as undergraduate researcher in Telefonica I+D where developed his Final Career's Project, awarded with the Best Final Project Prize by the Official College of Telecommunication Engineers (COIT). In September 2013 he joined the Centre Tecnologic de Telecomunicacions de Catalunya (CTTC), where he combined a SW Engineer role with his academic studies (PhD). In 2019, after two years as Telecom consultant in Wipro Ltd, he joined Telefonica GCTIO as Technology Expert in Optical Transport Networks and SDN which is his current role.