Reliable Optical Networks With ODTN, Resiliency and Failover In Data Plane And Control Plane

Andrea Campanella[1], Boyuan Yan[1,6], Ramon Casellas[2], Alessio Giorgetti[3], Victor Lopez[4], Arturo Mayoral[5]

[1]*Open Networking Foundation, Menlo Park, USA*
[2]*CTTC/CERCA, Optical Networks and Systems Department, Castelldefels, Barcelona, Spain*
[3]*Scuola Superiore Sant'Anna / CNIT, Pisa, Italy*
[4]*Telefonica, Madrid, Spain*
[5]*Universitat Politècnica de Catalunya, Barcelona, Spain*
[6]*Beijing University of Posts and Telecommunications, Beijing, China*
andrea@opennetworking.org

Keywords: SDN, Optical Network Control, OpenConfig device models, Open Line System, resiliency, ONOS controller

Abstract

This paper reports a technical demonstration showing the use of the ONOS SDN controller for disaggregated transport networks, within the ODTN Project, covering the provisioning of data connectivity services and demonstrating advanced automatic failure recovery, both at the data and control planes.

1 Introduction

Disaggregation and modularization is a key component in the future of optical networks. A disaggregated model can be based on white boxes with open and standard APIs, where different optical network elements (such as ROADMs, transponders, line amplifiers, etc.) can be provided by different vendors. Such disaggregation allows for more flexible, reconfigurable and elastic network architectures and deployments. A key component in these networks is the optical domain controller, which is responsible for discovering the topology through multiple protocols and device-specific models, for providing service establishment, and to continuously manage and monitor the network. The overall disaggregated optical network architecture need to take into account several kinds of failures, both in the dataplane (such as node failures or fiber cuts and device malfunction), and in the control plane (such as the failure of a controller instance). This demonstration shows an optical network deployment controlled by the Open Network Operating System (ONOS®) [1]. During normal operation, ONOS first discovers the equipment through the Netconf protocol [2] and the corresponding device Yang models [3] (e.g OpenConfig [4]). Then, after receiving a request via its Northbound API, it provisions a bidirectional optical connectivity service across the network, configuring wavelength and power across the different devices. After the end-to-end optical channel is established, the demo simulates a dataplane failure (e.g., through a port down command), akeen to a real world fiber-cut. ONOS will automatically re-provision the path across the network to account for the failure, with minimal signal disruption. A second failure, this time in the control plane, will be demonstrated, showing the robustness of the ONOS controller and how the remaining instances recover.

2. Data Plane Deployment

The network is comprised of 2 Transponders and 2 ROADMs, which are connected through redundant links to support data plane failover. As Fig. 1 shows, the transponders are two Edgecore AS7716-24SC white box devices equipped with Lumentum CFP2-ACO Coherent Optical Transceivers on the optical side. The transponders expose a Netconf API modelling their capabilities and data through the OpenConfig Yang models. The Lumentum ACO card is integrated through a driver with the Transponder Abstraction Interface (TAI) [5] which exposes a high level set of APIs to configure transceiver capabilities.

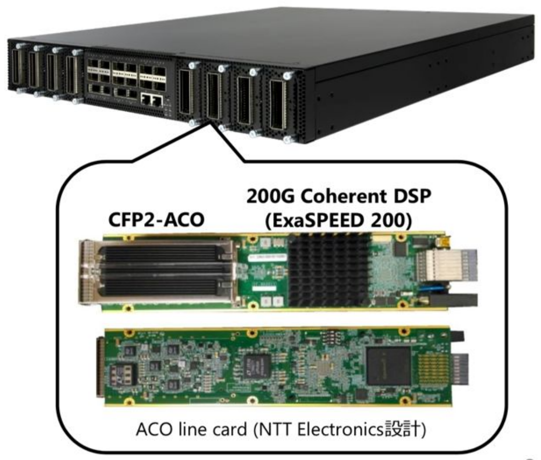
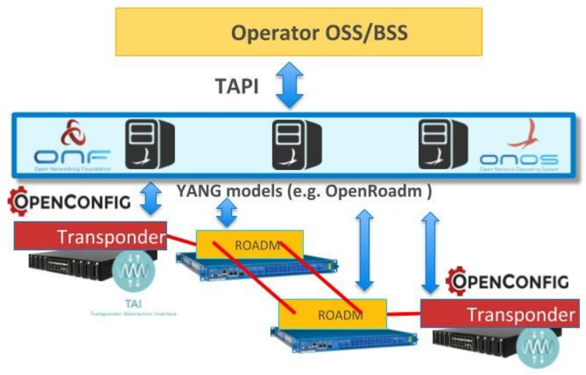


Fig. 1 Transponder with CFP2-ACO transceivers.

The ROADMS are Lumentum ROADM-20 white boxes that expose a Netconf API described by a Yang model provided with the device [9]. The Transponders Client ports have emulated end-hosts to generate traffic and measure network state.

3 Control Plane Architecture

The ONOS SDN Controller is deployed in a 3 instance scenario, as Fig. 2 shows. Every instance runs identical java code in a Java Virtual Machine (JVM) inside it's own Docker container. The instances share state through a 3 instance ATOMIX cluster [6], again deployed in a 3 instance scenario with 3 other Docker containers running a JVM. This results in 6 docker containers for the total deployment. 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.



4 Topology Discovery and Optical Connectivity Establishment

Topology information, involving devices' ports and capabilities (transceivers, physical-channels, identifiers, etc.) as well as links is discovered through OpenConfig interfaces based on NetConf protocol. Topological elements are stored in the ONOS dynamic configuration store. The storage is structured following pre-configured data models. In particular, ONF open Transport API (TAPI) [7] data models, so TAPI data nodes such as links, nodes, edge points and Service Interface Points (SIPs) are exposed to applications and high level API consumers such as network orchestrators or operators' OSS/BSS, through a RESTCONF [8] interface. Other than retrieving service interface points (SIPs), such interface may also be used to learn about the network topology and issue connectivity service requests.

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, 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 flow rules. For example, for the transceivers optical ports these rules convey the DWDM frequency and grid, port number, as well as information required to configure a cross-connection between the client side port and the line side port. 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. 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. 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.

5 Data Plane failure and recovery

With the traffic flowing we simulate a fiber-cut failure. The fiber cut is simulated by switching one of the ROADM's ports to a down state via the device CLI, completely separate from ONOS. Such port-down event is recognized by ONOS that in turn marks the link as failed. The link event gets parsed by the optical path computation module that, with the updated topology recomputes the path (i.e., using remaining devices and links).

Such re-computation is automatically triggered by the port-down event events with no human intervention. The same mechanism is used in case of device failure, when the ONOS controller loses the connection to a device it considers the device as failed and trigger 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, failover scenarios is done in the ONOS intent subsystem.
In the demo a pair of optical links will be deployed between the ROADMs to accommodate for the simulated link failure.

6 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 demo, we show ONOS deployed in a multi-instance scenario. In such scenario ONOS is capable of handling instance

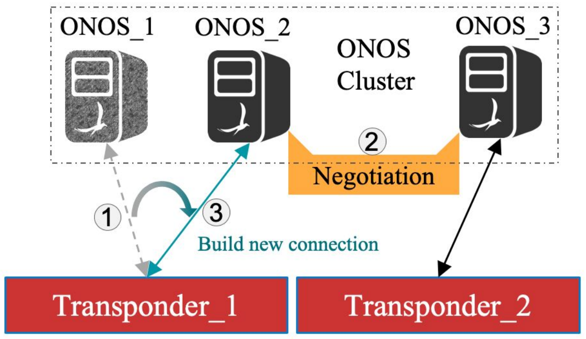


Fig. 3 Mastership movement in ONOS cluster.

failure by leveraging shared state and changing device mastership.

Initially, and as Fig. 3 shows, 3 docker containers of ONOS control the network together, where ONOS_1 and ONOS_3 control Transponder_1 and Transponder_2 respectively, having state synchronization between the instances. At step ①, ONOS_1 docker container is killed and Transponder_1 goes immediately out of management because no Netconf session is open to him. 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. At this point ONOS_2 is again in full control of Transponder 1 and can react to events and provision configuration on it. 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.

7 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 a common and open data model. Such deployment enables advanced recovery, both at the control and data plane layer. For the former, the recovery leverages ONOS distributed capabilities that provide a level of robustness against failures of the control processes. Overall, it shows the feasibility of the selected approach and the readiness of the system for production grade deployments. The demo shows also the feasibility of Open source software and open APIs for mission critical deployments

8 Acknowledgements

This work has been done under the ONF framework with an open community and the code lives under the APACHE 2.0 license, it was also partially supported by the EC H2020 METRO-HAUL Project with G.A 761727, and BUPT Ph.D. Students Short Term Exchange Program.

9 References

[1] Open Network Operating System', https://github.com/opennetworkinglab/onos, Accessed 6 May 2019.

[2] RFC6241: 'Network Configuration Protocol (NETCONF)', 2011.
[3] RFC6020: 'YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)', 2015.
[4] 'Open Config: Vendor-neutral, model-driven network management designed by users', http://openconfig.net/, accessed 6 May 2019.
[5] 'Disaggregated Transponder Chip Transport Abstraction Interface', https://github.com/Telecominfraproject/oopt-tai, Accessed 6 May 2019.
[6] ‘Atomix framework’, https://atomix.io/
[7] 'Open Transport Configuration & Control', https://wiki.opennetworking.org/display/OTCC/TAPI, Accessed 6 May, 2019.
[8] RFC8040: 'RESTCONF Protocol', 2017.
[9] A. Sgambelluri, J.-L. Izquierdo-Zaragoza, A. Giorgetti, et al "Fully Disaggregated ROADM White Box with NETCONF/YANG Control, Telemetry, and Machine Learning-based Monitoring" in Tech. Dig. OFC 2018.