Open Software Development Kit (OpenSDK) for Optical Network Disaggregation

Filippo Cugini,[1,*] Davide Scano,[2] Andrea Sgambelluri,[2]
Francesco Paolucci,[1] Alessio Giorgetti,[3] and Piero Castoldi[2]

[1] *CNIT, Pisa, Italy*
[2] *Scuola Superiore Sant'Anna, Pisa, Italy*
[3] *CNR, Pisa, Italy*

[*] *filippo.cugini@cnit.it*

Abstract: OpenSDK is proposed to provide vendor-neutral, micro-service-based control of underlying optical hardware. Disaggregation is then achieved without requiring standard Southbound interfaces from the SDN Controller. Validation is performed enforcing smart operations on IPoWDM white box. © 2024 The Author(s)

1. Introduction

Aiming at achieving cost savings and avoid vendor lock-in, disaggregated optical networks are characterized by network elements such as transponders/transceivers and line systems controlled by vendor-neutral open interfaces. So far, the NETCONF/YANG southbound interface (SBI) has been considered a key technology for disaggregation. YANG provides standardized data models to represent network elements and their attributes [1,2].

In the last decade, huge effort has been devoted to the definition of standard YANG data models. However, their adoption is still very limited, due to several open issues. First, multiple independent initiatives such as OpenROADM, OpenConfig and IETF have been working in isolation on the definition of their own data models. As a consequence, each of these data models is supported by only a few system vendors. Second, they do not always provide clear descriptions on how to interpret/implement the models, such as for disaggregated optical line systems (OLS), leading to interoperability issues even under the same initiative. Third, network control is always performed by proprietary SDN Controllers, which can hardly introduce advanced effective features without losing compatibility. Specifically, advanced functionalities would require software modules also deployed at the network element [3–5], relying on functional models that are beyond the current scope of an SDN agent using standard YANG models.

In this work, we propose open software development kit (OpenSDK) as a different approach to provide optical network disaggregation. OpenSDK enables third-party tools and resources to run on the disaggregated hardware elements. In OpenSDK, network elements need to (1) support micro-services on top of a basic operating system (nowadays rather common, as in SONiC Operating System (OS) [6,7]), and (2) provide standard comprehensive set of tools (i.e., SDK) to control and manage the underlying optical hardware components of a network element. This proposal differs from previously presented solutions, such as Transponder Abstraction Interface (TAI) [8], since TAI were envisioned to be used together with standard YANG data models (i.e., without enabling proprietary tools to run on the network elements) and as such they represented an additional and redundant layer of abstraction. On the contrary, with OpenSDK, vendor neutral control of disaggregated hardware is guaranteed by standardized SDKs. Thus, the southbound interface including the communication protocol from the (proprietary) SDN controller to the network element hosting the (proprietary) SDN Agent can be of any type. Furthermore, OpenSDK facilitates the deployment of SDN-controlled smart solutions at the network element, e.g., based on local AI processing [9]. To facilitate the adoption of OpenSDK solution, the work performed by disaggregation initiatives should be exploited as much as possible. This includes the state and config parameters by OpenROADM/OpenConfig as well as the Linecard Abstraction Interface (LAI) [6] and other SONiC-based deployments, augmented to provide more accurate device control and smart capabilities.

2. Proposed OpenSDK solution

Fig. 1(left) shows a traditional non-disaggregated solution provided by a single vendor which leads to vendor lock-in. Fig. 1(center) shows the current disaggregated approach having the Vendor of the SDN Controller controlling white boxes provided by other Vendors. In this case, standard NETCONF/YANG SBI is adopted to communicate with the network element. Such communication is handled at the network element by an SDN software agent configuring the underlying hardware (e.g., coherent pluggables modules in IPoWDM white boxes) using proprietary

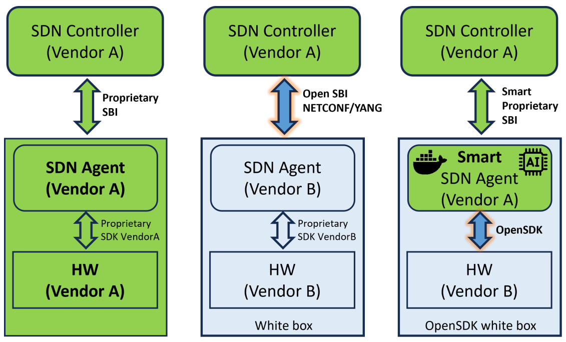


Fig. 1. Left: non-disaggregated solution provided by a single vendor. Center: current disaggregated approach based on standard NETCONF/YANG SBI. Right: proposed OpenSDK solution.

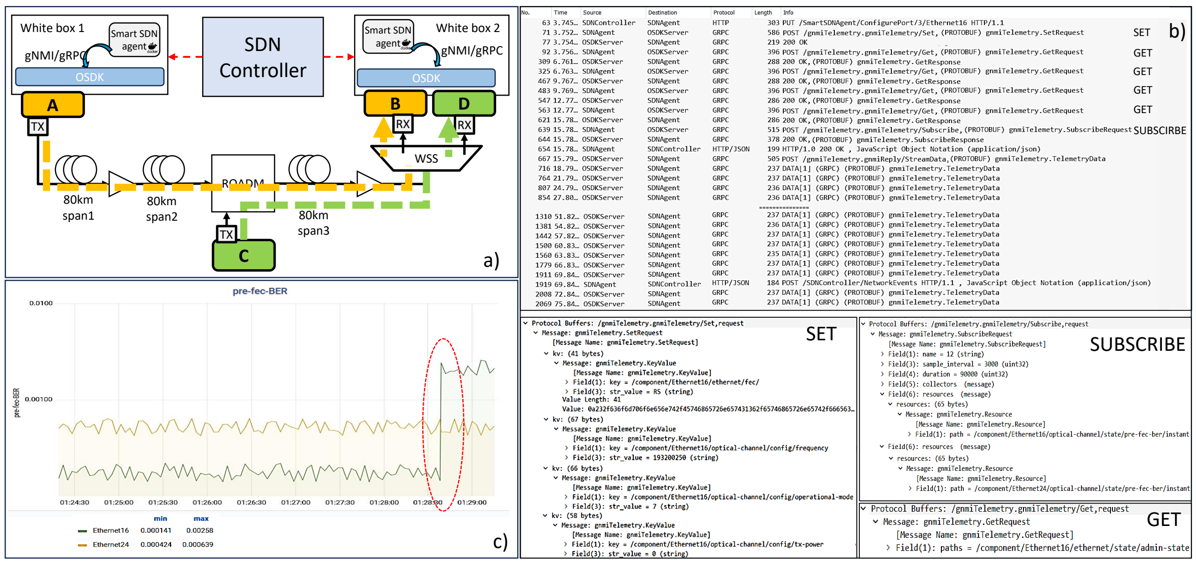
SDK. Fig. 1(right) shows the proposed disaggregated solution based on OpenSDK. In this case, the Vendor of the SDN Controller provides the white box with a proprietary, micro-service-based (e.g., in the form of a docker container) software module including the SDN Agent. Communication between Controller and network elements is provided with proprietary data models and protocols. The proprietary software module can also encompass smart management features of the underlying hardware, including smart AI-empowered applications for correlation of locally retrieved data, monitoring and failure management.

OpenSDK provides Application Programming Interface (API) for the SDK as well as libraries and software submodules that implement the SDK API. The SDK API provides methods and functions for developers to interact with the optical network element. This includes functions for configuring transmission parameters, monitoring signal quality, and managing network connections. More specifically, we implemented OpenSDK according to a hierarchical structure, having the higher layer describing the whole network element (i.e., platform) and the lower level detailing the specific hardware components.

In the case of an IPoWDM OpenSDK white box, the platform provides the number and type of ports (e.g., form factor, number of electrical lanes). Then, each transceiver inserted in a port is managed according to the specific type of pluggable module. For example, the 400ZR+ module is specified according to CMIS/C-CMIS specifications. Its SDK API includes get/set of transmission parameters including frequency, output power and operational mode as well as other state parameters. In addition, with OpenSDK, the libraries for managing the 400ZR+ modules need also to be provided by the OpenSDK white box Vendor. This includes for example the mechanisms for initializing and managing the EEPROM pages (e.g., Versatile Diagnostics Monitoring - VDM, configuration, and status), the Module State Machine (MSM) for initialization, and the Data Path State Machine (DPSM) for standard configuration. Similarly, OpenSDK can be applied to ROADMs and amplifiers. The ROADM platform provides the list of components (i.e., SRG add/drop, degree, amplifiers) and the internal connectivity to enable cross connections. Then, each component is managed according to its specific type (e.g., get/set cross-connections among internal/external ports of a degree component). Similarly, amplifiers are described via their traditional specific config and state parameters (e.g., power, tilt, etc), but managed by a standard SDK node and not through standard SBI from the Controller. In our implementation, OpenSDK API relies on the gNMI technology.

3. OpenSDK experimental demonstration

Fig. 2a shows the considered network testbed. It consists of three optical fiber spans of 80km each, interconnecting a pair of IPoWDM white boxes with 400ZR+ transceivers running SONiC OS. A software module, including the SDN Agent and a smart Application, is deployed as docker container in each of the white boxes. The App performs intelligent data aggregation and correlation. As an example, it is able to retrieve and analyze RX parameters from optical signals that are partially sharing one of the links, as A-B and C-D in the figure. Fig. 2b shows the Wireshark capture of the messages exchanged by the IPoWDM SDN Agent with both the SDN Controller and the underlying OpenSDK implementation. Message n. 63 shows the request from the Controller to the Agent to perform the configuration of the pluggable module and then apply a smart correlation between pre-FEC BER of signals A-B and C-D. The request is made through a proprietary REST interface. The SDN agent communicates via gRPC/gNMI procedures with the OpenSDK module, implemented relying on the OpenConfig YANG models version 1.9.0. In particular, it configures the pluggable parameters, by exploiting the *Set* RPC (message 71, exploded in the bottom part of the figure). Then, the same SDN agent waits that the card becomes active, by using the *Get* RPC (messages 92, 309, 467, 483, 563). At this point, the App performs the subscription to the pair of underlying transceivers to retrieve pre-FEC BER values (message 639). The subscription details are exploded in the bottom part of Fig. 2b. Finally, the telemetry stream is activated for both connections. As shown in Fig. 2c, in the Grafana dashboard attached to the smart SDN agent module, a soft failure affects Ethernet16 pluggable (i.e.,



```
No.   Time    Source         Destination    Protocol   Length Info
   63 3.745…  SDNController  SDNAgent       HTTP          303 PUT /SmartSDNAgent/ConfigurePort/3/Ethernet16 HTTP/1.1
   71 3.752…  SDNAgent       OSDKServer     GRPC          586 POST /gnmiTelemetry.gnmiTelemetry/Set, (PROTOBUF) gnmiTelemetry.SetRequest        SET
   77 3.754…  OSDKServer     SDNAgent       GRPC          219 200 OK
   92 3.756…  SDNAgent       OSDKServer     GRPC          396 POST /gnmiTelemetry.gnmiTelemetry/Get, (PROTOBUF) gnmiTelemetry.GetRequest        GET
  309 6.761…  OSDKServer     SDNAgent       GRPC          288 200 OK, (PROTOBUF) gnmiTelemetry.GetResponse
  325 6.763…  SDNAgent       OSDKServer     GRPC          396 POST /gnmiTelemetry.gnmiTelemetry/Get, (PROTOBUF) gnmiTelemetry.GetRequest        GET
  467 9.767…  OSDKServer     SDNAgent       GRPC          288 200 OK, (PROTOBUF) gnmiTelemetry.GetResponse
  483 9.769…  SDNAgent       OSDKServer     GRPC          396 POST /gnmiTelemetry.gnmiTelemetry/Get, (PROTOBUF) gnmiTelemetry.GetRequest        GET
  547 12.77…  OSDKServer     SDNAgent       GRPC          286 200 OK, (PROTOBUF) gnmiTelemetry.GetResponse
  563 12.77…  SDNAgent       OSDKServer     GRPC          396 POST /gnmiTelemetry.gnmiTelemetry/Get, (PROTOBUF) gnmiTelemetry.GetRequest        GET
  621 15.78…  OSDKServer     SDNAgent       GRPC          286 200 OK, (PROTOBUF) gnmiTelemetry.GetResponse
  639 15.78…  SDNAgent       OSDKServer     GRPC          515 POST /gnmiTelemetry.gnmiTelemetry/Subscribe, (PROTOBUF) gnmiTelemetry.SubscribeRequest SUBSCIRBE
  644 15.78…  OSDKServer     SDNAgent       GRPC          378 200 OK, (PROTOBUF) gnmiTelemetry.SubscribeResponse
  654 15.78…  SDNAgent       SDNController  HTTP/JSON     199 HTTP/1.0 200 OK , JavaScript Object Notation (application/json)
  667 15.79…  OSDKServer     SDNAgent       GRPC          505 POST /gnmiTelemetry.gnmiReply/StreamData, (PROTOBUF) gnmiTelemetry.TelemetryData
  716 18.79…  OSDKServer     SDNAgent       GRPC          237 DATA[1] (GRPC) (PROTOBUF) gnmiTelemetry.TelemetryData
  764 21.79…  OSDKServer     SDNAgent       GRPC          237 DATA[1] (GRPC) (PROTOBUF) gnmiTelemetry.TelemetryData
  807 24.79…  OSDKServer     SDNAgent       GRPC          236 DATA[1] (GRPC) (PROTOBUF) gnmiTelemetry.TelemetryData
  854 27.80…  OSDKServer     SDNAgent       GRPC          236 DATA[1] (GRPC) (PROTOBUF) gnmiTelemetry.TelemetryData
                                                     ==============
 1310 51.82…  OSDKServer     SDNAgent       GRPC          237 DATA[1] (GRPC) (PROTOBUF) gnmiTelemetry.TelemetryData
 1381 54.82…  OSDKServer     SDNAgent       GRPC          236 DATA[1] (GRPC) (PROTOBUF) gnmiTelemetry.TelemetryData
 1442 57.82…  OSDKServer     SDNAgent       GRPC          237 DATA[1] (GRPC) (PROTOBUF) gnmiTelemetry.TelemetryData
 1500 60.83…  OSDKServer     SDNAgent       GRPC          237 DATA[1] (GRPC) (PROTOBUF) gnmiTelemetry.TelemetryData
 1560 63.83…  OSDKServer     SDNAgent       GRPC          235 DATA[1] (GRPC) (PROTOBUF) gnmiTelemetry.TelemetryData
 1779 66.83…  OSDKServer     SDNAgent       GRPC          237 DATA[1] (GRPC) (PROTOBUF) gnmiTelemetry.TelemetryData
 1911 69.84…  OSDKServer     SDNAgent       GRPC          237 DATA[1] (GRPC) (PROTOBUF) gnmiTelemetry.TelemetryData
 1919 69.84…  SDNAgent       SDNController  HTTP/JSON     184 POST /SDNController/NetworkEvents HTTP/1.1 , JavaScript Object Notation (application/json)
 2008 72.84…  OSDKServer     SDNAgent       GRPC          237 DATA[1] (GRPC) (PROTOBUF) gnmiTelemetry.TelemetryData
 2069 75.84…  OSDKServer     SDNAgent       GRPC          237 DATA[1] (GRPC) (PROTOBUF) gnmiTelemetry.TelemetryData
```

```
v Protocol Buffers: /gnmiTelemetry.gnmiTelemetry/Set,request                    SET
  v Message: gnmiTelemetry.SetRequest
      [Message Name: gnmiTelemetry.SetRequest]
    v kv: (41 bytes)
      v Message: gnmiTelemetry.KeyValue
          [Message Name: gnmiTelemetry.KeyValue]
        > Field(1): key = /component/Ethernet16/ethernet/fec/
        > Field(3): str_value = RS (string)
        Value Length: 41
        Value: 0a232f636f6d706f6e656e742f45746865726e657431362f65746865726e65742f666563…
    v kv: (67 bytes)
      v Message: gnmiTelemetry.KeyValue
          [Message Name: gnmiTelemetry.KeyValue]
        > Field(1): key = /component/Ethernet16/optical-channel/config/frequency
        > Field(3): str_value = 193200250 (string)
    v kv: (66 bytes)
      v Message: gnmiTelemetry.KeyValue
          [Message Name: gnmiTelemetry.KeyValue]
        > Field(1): key = /component/Ethernet16/optical-channel/config/operational-mode
        > Field(3): str_value = 7 (string)
    v kv: (58 bytes)
      v Message: gnmiTelemetry.KeyValue
          [Message Name: gnmiTelemetry.KeyValue]
        > Field(1): key = /component/Ethernet16/optical-channel/config/tx-power
        > Field(3): str_value = 0 (string)
```

```
v Protocol Buffers: /gnmiTelemetry.gnmiTelemetry/Subscribe,request
  v Message: gnmiTelemetry.SubscribeRequest                                     SUBSCRIBE
      [Message Name: gnmiTelemetry.SubscribeRequest]
    > Field(1): name = 12 (string)
    > Field(3): sample_interval = 3000 (uint32)
    > Field(4): duration = 90000 (uint32)
    > Field(5): collectors  (message)
    v Field(6): resources  (message)
      v resources: (65 bytes)
        v Message: gnmiTelemetry.Resource
            [Message Name: gnmiTelemetry.Resource]
          > Field(1): path = /component/Ethernet16/optical-channel/state/pre-fec-ber/instant
    v Field(6): resources  (message)
      v resources: (65 bytes)
        v Message: gnmiTelemetry.Resource
            [Message Name: gnmiTelemetry.Resource]
          > Field(1): path = /component/Ethernet24/optical-channel/state/pre-fec-ber/instant
```

```
v Protocol Buffers: /gnmiTelemetry.gnmiTelemetry/Get,request                    GET
  v Message: gnmiTelemetry.GetRequest
      [Message Name: gnmiTelemetry.GetRequest]
    > Field(1): paths = /component/Ethernet16/ethernet/state/admin-state
```

Fig. 2. (a): testbed; (b) Wireshark captures at white box 2; (c) Pre-FEC BER telemetry monitors.

box D receiver in Fig. 2a). The App detects a relative deviation of pre-FEC BER, before it reaches critical levels. Message 1919 shows the alarm notification sent to the Controller, including the indication that can significantly facilitate the failure localization. No other data or messages have to be sent to the Controller.

The same experiment has been repeated using different HW (i.e., 100G transponders) from a different Vendor and with different OS (still supporting micro-services). Only adaptation of the key string in the OpenSDK API has been performed, while the very same smart App has been successfully operated over the new harwdare with no modifications. Using traditional SDN-based NETCONF/YANG disaggregated approaches, no smart APPs could have been installed and controlled at the node level. As a consequence, smart data aggregation and correlation could practically occur only at the Controller level, which could easily be overloaded by excessive data. Furthermore, alarms are triggered by the crossing of individual threshold, and not upon deviations among co-monitored channels. Additional examples of smart operations requiring a combination of local and centralized intelligence include effective power equalization techniques running in ROADMs or optimized amplifier gain/tilt control running at the amplifier level. Indeed, smart control and management operations performed by propriety optical line system (OLS) are typically enabled by advanced control features rather than by different commands to underlying coherent pluggables (nowadays all compliant to C-CMIS) or WSSes and amplifiers.

4. Conclusions

This paper introduced OpenSDK as innovative approach to enable disaggregation in optical networks, providing standard, micro-service-based control of underlying optical hardware without requiring standard NETCONF/YANG communication with the SDN Controller. The proposed OpenSDK solution was implemented and validated on transponders and IPoWDM white boxes, showing the effective capability to enforce intelligent data aggregation and correlation at the white box level.

Acknowledgment. This work is supported by the EU ALLEGRO Project (grant agreement No. 101070009) and by the MIUR PRIN2022 ZeTON Project (GA 2022YA59ZJ).

References

1. E. Riccardi *et al.*, "An operator view on the introduction of white boxes into optical networks," JLT (2018).
2. A. Sgambelluri, A. Giorgetti, D. Scano, F. Cugini, and F. Paolucci, "OpenConfig and OpenROADM automation of operational modes in disaggregated optical networks," IEEE Access **8**, 190094–190107 (2020).
3. G. Borraccini *et al.*, "Experimental demonstration of partially disaggregated optical network control using the physical layer digital twin," IEEE Trans. on Netw. Serv. Manag. **20**, 2343–2355 (2023).
4. Z. Wang, D. Kilper, and T. Chen, "An Open EDFA Gain Spectrum Dataset and Its Applications in Data-driven EDFA Gain Modeling," in *Optica Open,* (2023).
5. L. Gifre *et al.*, "Experimental demonstration of transport network slicing with SLA using the teraflowsdn controller," in *ECOC,* (2022).
6. W. Zheng *et al.*, "Sonic-based network operating system for open whitebox optical transport," in *ECOC,* (2022).
7. A. Giorgetti *et al.*, "Enabling hierarchical control of coherent pluggable transceivers in sonic packet–optical nodes," J. Opt. Commun. Netw. **15**, 163–173 (2023).
8. V. Lopez *et al.*, "Enabling fully programmable transponder white boxes [invited]," J. Opt. Commun. Netw. (2020).
9. L. Velasco, P. González, and M. Ruiz, "An intelligent optical telemetry architecture," in *OFC,* (2023).