# Management of Sliceable Transponder with NETCONF and YANG

Matteo Dallaglio*, Nicola Sambo*, Filippo Cugini† and Piero Castoldi*
*Scuola Superiore Sant'Anna, Pisa, Italy
Email: m.dallaglio@sssup.it
†CNIT, Pisa, Italy

***Abstract*—NETCONF is emerging as Software Defined Networking (SDN) protocol for the control and management of optical networks. It enables data plane device configuration and access to monitoring information. NETCONF may exploit YANG data model to describe network elements to be controlled/managed. YANG and NETCONF are of interest for operators since they provide a standard way to control and manage network elements, independently from the vendor.**

**This paper presents and demonstrates a management plane based on NETCONF protocol. In particular, a YANG model describing optical transponders supporting slice-ability, variable rate, and monitoring functionalities is proposed. NETCONF experimental demonstrations are carried out to validate the proposed model and to prove the control and management capabilities of these technologies applied to elastic optical networks (EONs).**

***Index Terms*—NETCONF, YANG, Sliceable Transponder, EON, SBVT, slice-ability, elastic.**

## I. Introduction

Data and control plane of optical networks have experienced relevant advances in the recent years. Considering data plane technologies, sliceable transponders are emerging as a technology meeting the requirements of operators to support variable bit-rate, optimizing the spectral efficiency based on the required optical reach — thus supporting multiple modulation formats or forward error correction (FEC) — and also enabling slice-ability (i.e., the capability of generating independent optical flows to be directed toward different paths and destinations) [1]–[4]. Such transponders, thanks to coherent detection, also support monitoring of transmission parameters (e.g., pre-FEC bit error rate —pre-FEC BER) [5]. These technologies aims to increase both the scalability and agility of the transport network, allowing resource optimization and scaling of bandwidth as demands change and increase. However, while data and control planes have experienced such advances, the innovations in the management plane still need improvements [6] to develop management mechanisms to reduce deployment and operational complexity and maximize benefits of EONs capabilities.

Network Configuration Protocol (NETCONF) [7] is emerging as an SDN protocol standardized by the Internet Engineering Task Force (IETF), providing both control (e.g., data plane device configuration) and management functionalities (e.g., access to monitoring information). NETCONF protocol provides mechanisms to install, manipulate, and delete management states and information of network devices. NETCONF may rely on the Yet Another Next Generation (YANG) modelling language [8], [9] to describe network devices in a standard way. YANG and NETCONF are gaining interest for operators and research since they provide a standard way to control and manage network elements, independently from the vendor [10].

NETCONF and YANG have been introduced in several networking scenarios. For example, in [11], authors propose a NETCONF agent for link state monitoring compared with other management technologies such as Simple Network Management Protocol (SNMP). In [12], authors highlight the benefits provided by NETCONF in terms of security and scalability, compared with SNMP and REpresentational State Transfer (REST). In [13], authors uses YANG and NETCONF to manage a 10G-PON. Preliminary works of YANG modelling applied to EONs are mentioned in [14]. However, the YANG model has not been reported, and NETCONF is not fully supported.

All the mentioned works do not address a definition of a YANG model for sliceable transponders in EONs employing NETCONF protocol.

This paper presents and demonstrates a control and management plane based on NETCONF protocol exploiting YANG model. In particular, the proposed YANG model describes transponders based on multi-carrier technology, enabling slice-ability and variable baud rate, bit rate, number of carriers, FEC, and modulation format, and supporting the monitoring of several physical parameters such as chromatic dispersion, BER, Q-factor, and others. The paper presents a management experiment using NETCONF exploiting the proposed YANG model. In particular, the controller performs device state discovery, and monitoring information polling.

## II. YANG and NETCONF

YANG is a data modelling language that can be used to express the structure and semantics of a device information in a vendor-neutral format [8]. Two types of information are present in a YANG model: configuration data and state (operational) data. Configuration data are explicitly set by an external entity on the system. State data reflects the parameters that cannot be set by an external entity (some of these data are fixed by the vendor).

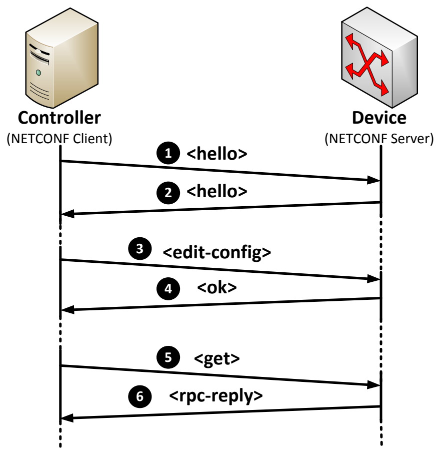


Fig. 1. NETCONF message exchange.

NETCONF is a network configuration and management protocol standardized by the IETF [7]. NETCONF is based on a simple Remote Procedure Call (RPC) layer running over secure transport protocols (typically SSH) to foster interoperability and security. Compared to Simple Network Management Protocol (SNMP), NETCONF has been standardized later, thus it has been designed with additional features to overcome SNMP lacks. Some of the limitation of SNMP have been in RFC 3535 [15] after the IETF workshop held by the Internet Architecture Board on Network Management in 2002. NETCONF provides all the benefit of SNMP as well as many useful features. In particular, NETCONF is transaction based, i.e. when some of the network devices successfully upload the configuration but others fail, NETCONF allows a managed device to rollback to a known-state configuration. Configuration and operational states are well defined and clearly separated. NETCONF supports multiple configuration data stores (candidate, running, startup). Moreover, streaming and playback of event notifications are also supported. The main NETCONF message exchange is illustrated in Fig. 1. Initially, controller (client) and device (server) create a NETCONF session and exchange the list of their capabilities by sending `<hello>` messages (1,2). A capability describes a supported data model. Once the session is opened, the NETCONF peers exchange `<rpc>` and `<rpc-reply>` messages. The `<rpc>` message is used to enclose a NETCONF command sent from the controller (client) to the device (server). The `<get>` command of the `<rpc>` message is used to retrieve the running configuration and state information of the device (3). The `<edit-config>` request is used to write a specific configuration on the device (5) (e.g. to set the maximum transmission unit of a specific Ethernet interface). The `<rpc-reply>` message is sent from the device to the controller in response to an `<rpc>` message. The response data for the given method invoked is encoded as one or more child elements enclosed in the `<rpc-reply>` message.

## III. Transponder Model

In this section, the proposed YANG model for a sliceable transponder with monitoring capabilities is described. The sliceable transponder presented in [2] has been used as a reference architecture to design the YANG model. Standardization guidelines on data modelling suggested by IETF in [16] and by the OpenConfig working group in [17] have been considered. The model has been published online on a public repository [18]. A schematic view of the proposed YANG model describing the transponder is organized in the tree diagram shown in Fig. 2. In particular, a generic transponder is modelled as a list of sub-carrier modules, each one generating or detecting a sub-carrier and a list of connections. In case more than one sub-carrier modules are installed, the Boolean field slice-ability-support is used to indicate whether the transponder supports slice-ability or not [2]. The configuration data of the sub-carrier module are the following: e.g., bit rate, baud rate, modulation, FEC, central frequency, and bandwidth. Some additional fields may appear depending whether the sub-carrier is used in reception (e.g., local oscillator, sampling rate, and analog bandwidth) or in transmission (e.g., the launch power). The state data of the sub-carrier module are the following: supported bit rates, baud rates, modulation formats, and FEC. Then, an additional field is proposed for monitoring capabilities, i.e. listing the parameters that can be monitored by the transponder itself: e.g., input power at the receiver. Regarding the connections, the configuration data are the following: a list of sub-carriers module ids referring to those sub-carriers involved in the connection, and the frequency slot composed by the nominal central frequency (n) and the slot width (m), according to ITU-T [19].

Listing 1 shows the proposed YANG model. In the next sections, the proposed YANG model code will be detailed.

### A. Sub-carrier module

In the model, each sub-carrier module (line 290) has a configurable direction (line 156) which is either transmission (TX) or reception (RX). Several parameters of the sub-carrier module can be configured, e.g. the bit rate (line 159), baud rate (line 163), modulation format (line 167), FEC (line 171), central frequency (line 172), and bandwidth (line 175). In addition, some other parameters can be configured depending whether the sub-carrier is used in reception or in transmission. In reception, those parameters are the local oscillator (line 104), sampling rate (line 107), and analog bandwidth (line 111), while in transmission, the launch power (line 94). The state data is composed by a replica of the configuration data (as explained in section II), plus the supported functionalities and some additional monitoring information. The supported functionality are organized in lists, in particular: list of supported bit rates (line 181), baud rates (line 186), modulation formats (line 191), and FEC (line 85). Then, an additional field is proposed for monitoring functionalities, i.e. listing the parameters that can be monitored by the transponder itself: input power at the receiver (line 117), pre-FEC BER

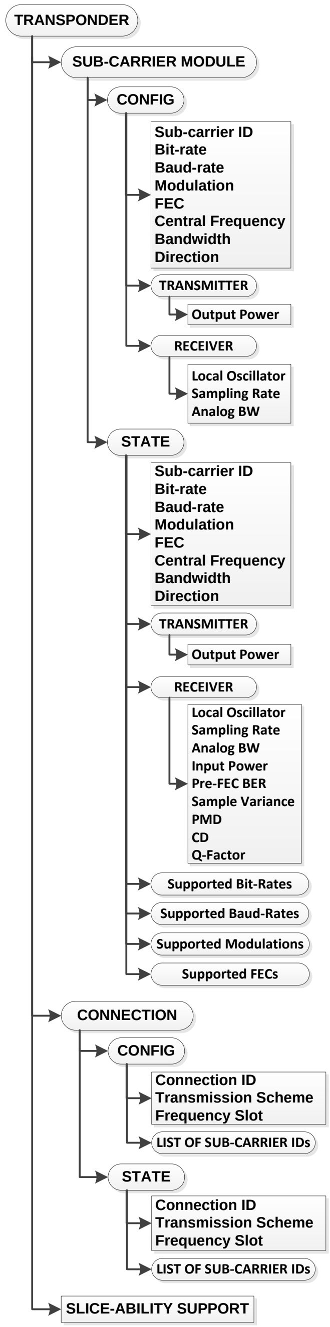


Fig. 2. Transponder YANG Tree

(line 121), polarization mode dispersion (line 134), chromatic dispersion (line 141), and Q-factor (line 147).

### B. Connections

In the model, each connection (line 270) is identified with a unique *connection-id* (line 272), uses a specific *transmission scheme* (e.g. NWDM) (line 232), is supported by one or more sub-carriers specified by the *sub-carrier list* (line 235), and occupies a certain amount of bandwidth specified by the *frequency-slot* (line 243). The sub-carrier list contains the IDs of those sub-carriers involved in the connection while the frequency slot is describes by the nominal central frequency index (n) (line 253) and the slot width (m) (line 257) according to ITU-T [19]. All the connection parameters introduced so far are configuration data. The state data is composed for the connections by just the replica of the configuration data.

Listing 1. Transponder YANG Model

```
1
2 module transponder {
3 namespace "http://sssup.it/transponder";
4 prefix tran;
5  import modulation-formats {
6    prefix mdfrms;
7  }
8  import fec-types {
9    prefix fec;
10 }
11 import ietf-yang-types {
12   prefix yang;
13 }
14
15 typedef transmission-type {
16   type enumeration {
17     enum NWDM;
18     enum O-OFDM;
19     enum TFP; //Time-frequency packing
20   }
21 }
22 typedef direction-type {
23   type enumeration {
24     enum TX;
25     enum RX;
26   }
27 }
28 typedef bit-rate-type {
29   type decimal64 {
30     fraction-digits 3;
31     range "0..max";
32   }
33   units "Gb/s";
34 }
35 typedef baud-rate-type {
36   type decimal64 {
37     fraction-digits 3;
38     range "0..max";
39   }
40   units "Gbaud";
41 }
42 typedef modulation-type {
43   type identityref {
44     base mdfrms:modulation-format;
45   }
46 }
47 typedef fec-type {
48   type identityref {
49     base fec:fec-type;
50   }
51 }
52 typedef frequency-ghz-type {
53   type decimal64 {
54     fraction-digits 8;
55     range "0..max";
56   }
57   units "GHz";
58 }
59
60 grouping fec-config {
61   container fec-in-use {
62     presence "Enables␣FEC";
63     leaf name {
64       type fec-type;
65     }
66     container rate {
67       leaf message-length {
68         type int16 {
69           range "1..max";
70         }
71       }
72       leaf block-length {
73         type int16 {
74           range "1..max";
75         }
76       }
77       must "block-length␣>=␣message-length" {
78         error-message "block-length␣must␣be␣>=␣
              message-length";
79       }
80     }//rate
81   }//fec-in-use
82 }//fec-config
83
84 grouping fec-state {
85   container supported-fec {
86     description "List␣of␣supported␣FEC␣schemes";
87     leaf-list fec {
88       type fec-type;
```

```
    }
  } // supported
} // fec-state

grouping transmitter-config {
  leaf output-power {
    type int16;
    units "dBm";
  }
} // transmitter-config

grouping transmitter-state {
}

grouping receiver-config {
  leaf local-oscillator {
    type frequency-ghz-type;
  }
  leaf sampling-rate {
    type uint32;
    units "GS/s";
  }
  leaf analog-bw {
    type frequency-ghz-type;
  }
} // receiver-config

grouping receiver-state {
  leaf input-power {
    type int16;
    units "dBm";
  }
  leaf pre-fec-ber {
    type decimal64 {
      fraction-digits 18;
      range "0..max";
    }
  }
  leaf sample-variance {
    type decimal64 {
      fraction-digits 18;
      range "0..max";
    }
    reference "http://ieeexplore.ieee.org/xpl/
        articleDetails.jsp?arnumber=7035536";
  }
  leaf pmd {
    type decimal64 {
      fraction-digits 8;
      range "0..max";
    }
    units "ps/(km)^0.5";
  }
  leaf cd {
    type decimal64 {
      fraction-digits 5;
    }
    units "ps/nm/km";
  }
  leaf q-factor{
    type decimal64 {
      fraction-digits 5;
    }
    units "dB";
  }
} // receiver-state

grouping subcarrier-module-config {
  leaf direction {
    type direction-type;
  }
  leaf bit-rate {
    description "The bit-rate in use";
    type bit-rate-type;
  }
  leaf baud-rate {
    description "The baud-rate in use";
    type baud-rate-type;
  }
  leaf modulation {
    description "Modulation format in use";
    type modulation-type;
  }
  uses fec-config;
  leaf central-frequency {
    type frequency-ghz-type;
  }
  leaf bandwidth {
    type frequency-ghz-type;
  }
} // subcarrier-module-config

grouping subcarrier-module-state {
  container supported-bit-rates{
    leaf-list bit-rate {
      type bit-rate-type;
    }
  }
  container supported-baud-rates {
    leaf-list baud-rate {
      type baud-rate-type;
    }
  }
  container supported-modulations {
    leaf-list modulation {
      type modulation-type;
    }
  }
  uses fec-state;
} // subcarrier-module-state

grouping subcarrier-module {
  container config {
    uses subcarrier-module-config;
    container transmitter {
      when "../direction = 'TX'";
      uses transmitter-config;
    }
    container receiver {
      when "../direction = 'RX'";
      uses receiver-config;
    }
  } // config
  container state {
    config false;
    uses subcarrier-module-config;
    uses subcarrier-module-state;
    container transmitter {
      when "../direction = 'TX'";
      uses transmitter-config;
      uses transmitter-state;
    }
    container receiver {
      when "../direction = 'RX'";
      uses receiver-config;
      uses receiver-state;
    }
  } // state
} // subcarrier-module

grouping connection-config {
  leaf connection-id{
    type uint32;
  }
  leaf transmission-scheme {
    type transmission-type;
  }
  list subcarrier {
    key "subcarrier-id";
    leaf subcarrier-id {
      type leafref {
        path "/tran:transponder/tran:subcarrier-module
            /tran:subcarrier-id";
      }
    }
  }
  container frequency-slot {
    reference "draft-ietf-ccamp-flexi-grid-fwk-07";
    leaf nominal-central-frequency-granularity {
      type frequency-ghz-type;
      default 6.25;
    }
    leaf slot-width-granularity {
      type frequency-ghz-type;
      default 12.5;
    }
    leaf n {
      type int16;
      mandatory true;
    }
    leaf m {
      type int16 {
        range "1..max";
      }
      mandatory true;
    }
  }
} // connection-config

grouping connection-state {
}

grouping connections {
  list connection {
    key "connection-id";
    leaf connection-id {
      type leafref {
        path "../config/connection-id";
      }
    }
    container config {
      uses connection-config;
    }
    container state {
      config false;
      uses connection-config;
      uses connection-state;
    }
  } // connection
```

```
  }//connections

  //―――――― MAIN TREE ――――――――――//
  container transponder {
      list subcarrier-module {
        key "subcarrier-id";
        leaf subcarrier-id {
          type uint32;
        }
        uses subcarrier-module;
      }

      leaf slice-ability-support {
        when "count(../subcarrier-module) > 1";
        type boolean;
        config false;
      }

      leaf node-id {
        type uint16;
      }

      leaf add-drop-id {
        type uint16;
      }

      container connections {
        uses connections;
      }
  }
}//transponder
```

## IV. Experimental Demonstration

The controller runs a python implementation of a NETCONF client. The device is emulated using a virtual machine with 4GB RAM, 1 processor at 3.4GHz, and Ubuntu Linux as operating system. The virtual machine runs ConfD, a NETCONF server implementation made by Tail-f (CISCO), and a C program we made to emulate the monitoring capabilities of the device (Fig. 3). Our program communicates with ConfD through specific APIs provided by the latter. The transponder YANG model, as in section III, has been included in the capabilities of both client and server. In the experiment, controller and devices are under the same local area network and connected to the same fast Ethernet switch. The experiment is divided into two parts. First, the controller discovers the device state and functionality. Then, the controller monitors an active connection by issuing NETCONF <get> command.

### A. Transponder discovery

In this part of the experiment, the controller interrogates the device to retrieve its current state (e.g. installed sub-carriers modules, supported transmission parameters). In particular, the controller issues an RPC <get> command asking for the full state information of the transponder (Fig. 1 (3)). Listing 2 shows the content of the message captured with Wireshark.

Listing 2. NETCONF RPC <get> message.

```
<?xml version="1.0" encoding="UTF-8" ?>
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
      message-id="1">
  <get><filter type='xpath' select='/transponder' /></get>
</rpc>
```

The device replies with a <rpc-reply> message containing the full transponder state information (Fig. 1 (4)). Listing 3 shows the content of the captured message (some portions of the message have been skipped by adding "..."). Among all data, it is possible to evince that the transponder has four sub-carrier modules installed. The first sub-carrier module (ID equal to 1) supports the following transmission parameters: four bit rates (112, 124, 224, 248 Gbps), two baud rates (28, 31 Gbaud), two possible modulation formats (dual polarization quadrature phase shift keying —DP-QPSK— and dual polarization 16 quadrature amplitude modulation —DP-16QAM), and two FEC schemes (Low-Density Parity-Check – LDPC, Golay). In particular, 112 and 124 Gbps are associated to DP-QPSK with 28 and 31 Gbaud, respectively, while 224 and 248 Gbps to DP-16QAM with 28 and 31 Gbaud, respectively.

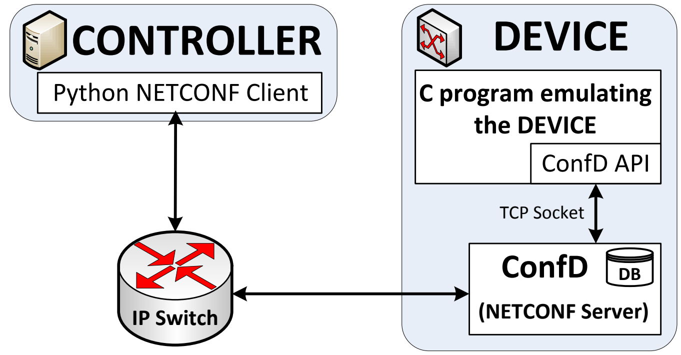


Fig. 3. Experimental setup.

Listing 3. NETCONF RPC reply message with the requested data.

```
<?xml version="1.0" encoding="UTF-8" ?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
      message-id="1">
  <data>
    <transponder xmlns="http://sssup.it/transponder">
      <subcarrier-module>
        <subcarrier-id>1</subcarrier-id>
        <state>
          <supported-bit-rates>
            <bit-rate>112.0</bit-rate>
            <bit-rate>124.0</bit-rate>
            <bit-rate>224.0</bit-rate>
            <bit-rate>248.0</bit-rate>
          </supported-bit-rates>
          <supported-baud-rates>
            <baud-rate>28.0</baud-rate>
            <baud-rate>31.0</baud-rate>
          </supported-baud-rates>
          <supported-modulations>
            <modulation xmlns:mdfrms="http://sssup.it/
                 modulation-formats">mdfrms:dp-qpsk</
                 modulation>
            <modulation xmlns:mdfrms="http://sssup.it/
                 modulation-formats">mdfrms:dp-16qam</
                 modulation>
          </supported-modulations>
          <supported-fec>
            <fec xmlns:fec="http://sssup.it/fec-types">fec:
                 ldpc</fec>
            <fec xmlns:fec="http://sssup.it/fec-types">fec:
                 golay</fec>
          </supported-fec>
        </state>
      </subcarrier-module>
      <subcarrier-module>
        <subcarrier-id>2</subcarrier-id>
        <state>
          <supported-bit-rates>
            <bit-rate>112.0</bit-rate>
            <bit-rate>124.0</bit-rate>
            <bit-rate>224.0</bit-rate>
            <bit-rate>248.0</bit-rate>
          </supported-bit-rates>
          <supported-baud-rates>
            <baud-rate>28.0</baud-rate>
            <baud-rate>31.0</baud-rate>
          </supported-baud-rates>
          <supported-modulations>
            <modulation xmlns:mdfrms="http://sssup.it/
                 modulation-formats">mdfrms:dp-qpsk</
                 modulation>
            <modulation xmlns:mdfrms="http://sssup.it/
                 modulation-formats">mdfrms:dp-16qam</
                 modulation>
          </supported-modulations>
          <supported-fec>
            <fec xmlns:fec="http://sssup.it/fec-types">fec:
                 ldpc</fec>
            <fec xmlns:fec="http://sssup.it/fec-types">fec:
                 golay</fec>
          </supported-fec>
        </state>
      </subcarrier-module>
```

```
        <subcarrier-module>
          <subcarrier-id>3</subcarrier-id>
          :::
        </subcarrier-module>
        <subcarrier-module>
          <subcarrier-id>4</subcarrier-id>
          :::
        </subcarrier-module>
        <slice-ability-support>true</slice-ability-support>
        <node-id>1</node-id>
        <add-drop-id>1</add-drop-id>
        <connections>
        </connections>
      </transponder>
    </data>
</rpc-reply>
```

### *B. Retrieve monitoring information*

In the last part of the experiment, the controller monitors the Q-Factor of a specific connection by periodically issuing a `<get>` command (polling). The considered monitored connection was previously installed and assigned to the first sub-carrier module (sub-carrier id=1) in reception. More in details, it is a $112$ Gbps ($28$ Gbaud) connection, modulated with DP-QPSK modulation and a $14/15$ LDPC code. Listing 4 shows the content of the `<get>` command encapsulated in the RPC message issued by the controller.

Listing 4. NETCONF RPC `<get>` message to retrieve the Q-Factor.

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
    message-id="1">
    <get>
    <filter type='xpath' select=' /transponder/subcarrier-
        module[subcarrier-id=1]/state/receiver/q-factor'/>
    </get>
</rpc>
```

The reply of the device is showed in Listing 5. In particular, the returned message shows that the monitored connection has a Q-Factor value equal to $6dB$.

Listing 5. NETCONF RPC `<rpc-reply>` message containing the Q-Factor.

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
    message-id="1">
  <data>
    <transponder xmlns="http://sssup.it/transponder">
      <subcarrier-module>
        <subcarrier-id>1</subcarrier-id>
        <state>
          <receiver>
            <q-factor>6.0</q-factor>
          </receiver>
        </state>
      </subcarrier-module>
    </transponder>
  </data>
</rpc-reply>
```

During the experiment we measured the time from the instant when the `<get>` command is issued to the instant that the `<rpc-reply>` returns to the controller. This time is around 11 milliseconds. We also measured the time required to the device to process the request, i.e. from the instant when the `<get>` command is received to the instant when the `<rpc-reply>` is sent. This processing time is around 8.5 milliseconds.

## V. Conclusion

This paper proposed and demonstrated a NETCONF protocol including YANG model describing a transponder with monitoring capabilities. The proposed YANG model of a sliceable transponder is fully detailed in the paper. An experiment is successfully conducted considering two use cases: transponder discovery and monitoring of a connection. In both use cases, extremely fast performance has been achieved. NETCONF message captures are provided for `<get>`, and `<rpc-reply>` messages.

## References


[1] M. Jinno, H. Takara, Y. Sone, K. Yonenaga, and A. Hirano, "Multiflow optical transponder for efficient multilayer optical networking," *Communications Magazine, IEEE*, vol. 50, no. 5, pp. 56–65, 2012.

[2] N. Sambo and et al., "Next generation sliceable bandwidth variable transponders," *Communications Magazine, IEEE*, vol. 53, no. 2, 2015.

[3] M. Svaluto Moreolo, J. Fabrega, L. Nadal, F. Vilchez, V. Lopez, and J. Fernandez-Palacios, "Cost-effective data plane solutions based on OFDM technology for flexi-grid metro networks using sliceable bandwidth variable transponders," in *Proc. of ONDM*, May 2014.

[4] A. Napoli and at al., "Novel dac digital pre-emphasis algorithm for next-generation flexible optical transponders," in *Proc. of OFC*, March 2015.

[5] A. Napoli and et al., "Next generation elastic optical networks: The vision of the european research project IDEALIST," *Communications Magazine, IEEE*, vol. 53, no. 2, pp. 152–162, Feb 2015.

[6] A. Martinez, M. Yannuzzi, V. Lopez, D. Lopez, W. Ramirez, R. Serral-Gracia, X. Masip-Bruin, M. Maciejewski, and J. Altmann, "Network management challenges and trends in multi-layer and multi-vendor settings for carrier-grade networks," *Communications Surveys Tutorials, IEEE*, vol. 16, no. 4, 2014.

[7] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "Network configuration protocol (NETCONF)," IETF RFC 6241, June 2011.

[8] M. Bjorklund, "YANG - a data modeling language for the network configuration protocol (NETCONF)," IETF RFC 6020.

[9] J. Vergara and et al., IETF draft-vergara-flexigrid-yang-00, Oct. 2014.

[10] J. Schonwalder, M. Bjorklund, and P. Shafer, "Network configuration management using NETCONF and YANG," *Communications Magazine, IEEE*, vol. 48, no. 9, pp. 166–173, Sept 2010.

[11] D. Loureiro, P. Goncalves, and A. Nogueira, "NETCONF agent for link state monitoring," in *Communications (ICC), 2012 IEEE International Conference on*, June 2012, pp. 6565–6569.

[12] P. da Paz Ferraz Santos, R. Pereira Esteves, and L. Zambenedetti Granville, "Evaluating SNMP, NETCONF, and RESTful web services for router virtualization management," in *Proc. of IFIP/IEEE IM*, May 2015.

[13] A. Berti Sassi, M. Gavidia, E. Leao Fernandes, and M. Ribeiro Nascimento, "Integrated management of 10G-PON network element using NETCONF and OpenFlow," in *Proc. of CNSM*, Nov 2014.

[14] J. Oliveira, M. Siqueira, G. Curiel, A. Hirata, F. van't Hooft, D. Macedo, M. Colazza, and C. Rothenberg, "Experimental testbed of reconfigurable flexgrid optical network with virtualized GMPLS control plane and autonomic controls towards SDN," in *Microwave Optoelectronics Conference (IMOC), 2013 SBMO/IEEE MTT-S International*, Aug 2013, pp. 1–5.

[15] J. Schoenwaelder, "Overview of the 2002 iab network management workshop," IETF RFC 3535.

[16] A. Bierman, "Guidelines for authors and reviewers of yang data model documents," IETF RFC 6087.

[17] in *http://www.openconfig.net*.

[18] in *https://github.com/mattedallo/sssa/blob/master/yang-models/transponder.yang*.

[19] "Draft revised G.694.1 version 1.3," Unpublished ITU-T Study Group 15, Question 6.