# Optical Network Telemetry with Streaming Mechanisms using Transport API and Kafka

R. Vilalta(1), R. Casellas(1), R. Martínez(1), R. Muñoz(1), A. González-Muñiz(2), J.P. Fernández-Palacios(2)

(1) Centre Tecnològic de Telecomunicacions de Catalunya (CTTC/CERCA), ricard.vilalta@cttc.es
(2) Telefónica I+D, Global CTO Unit

***Abstract*** *We present a streaming mechanism for optical networks based on the Kafka architecture and protocols, to efficiently distribute state and network updates following the upcoming ONF Transport API streaming implementation agreement. The proposed mechanism is validated and experimentally evaluated.*

## Introduction

In the last decade, the need for improving control and management protocols in optical Software Defined Networks (SDN) has driven innovation and novel mechanisms for controlling and monitoring optical networks[1]. Several monitoring and telemetry protocols, such as gRPC, gNMI, or websockets, have been proposed to control and manage network equipment, with the final intention of providing full network automation.

Large optical monitoring of datasets is possible when combining them with massive cloud/edge computational and storage resources.[2] presents a complete telemetry service architecture including both control plane and monitoring/management plane modules with dedicated focus on disaggregated optical networks.

Apache Kafka is an open-source event streaming platform[3]. It provides a publish/subscribe event bus that is called Kafka broker. It combines three key capabilities: a) to publish (write) and subscribe to (read) streams of events, including continuous import/export of your data from other systems; b) to store streams of events durably and reliably ; and c) to process streams of events as they occur or retrospectively. This method is known as compacted log and it allows to gain and maintain alignment with current state. The client can achieve eventual consistency, with no need to request a complete context, by simply subscribing to the necessary streams.

The introduction of a Kafka broker for the messaging of telemetry data was first presented in[4]. The authors demonstrate optical network threshold-based streaming and verification for open DWDM systems, where through a closed-loop system, the SDN Controller can act upon the observed data and determine the possible actions to be triggered on the network elements.

Current Optical Line Systems (OLS) are being designed and developed in order to support disaggregated optical networks to provision Network Media Channels (NMC)[5]. Open Networking Foundation (ONF) Transport API(TAPI)[6] allows the dynamic control and monitoring of optical networks. Through this API, a client application can request and modify current network status.

With the objective of increasing the ability to provide network telemetry information based on information streams, ONF Transport API has been extended to provide this functionality[7]. Compared to simpler notification-based current solutions (e.g. using Yang push notifications) a Kafka solution provides: i) flexibility, the discovery allows the client to use various log strategies and stream connection protocols; ii) flow control, streaming provides engineered flow such that peak load is averaged using mechanisms such as back-pressure and/or selective pruning of detail; iii) reliability, the log-record-header provides information allowing subsequent alignment of the notifications guaranteeing delivery of each log record); iv) Consistency, since delivery guarantee ensures consistency of the client with the view presented by the provider allowing the client to reach eventual consistency v) High scalability and relatively low latency.

This paper presents an implementation of ONF Transport API streaming mechanism using Kafka broker, with the SDN controller acting as event producer and enabling clients to retrieve one or more streams, synchronize state and be informed of network changes, alarms and notifications. The authors experimentally validate the proposed mechanism in an SDN OLS controller and evaluate its performance in terms of latency towards

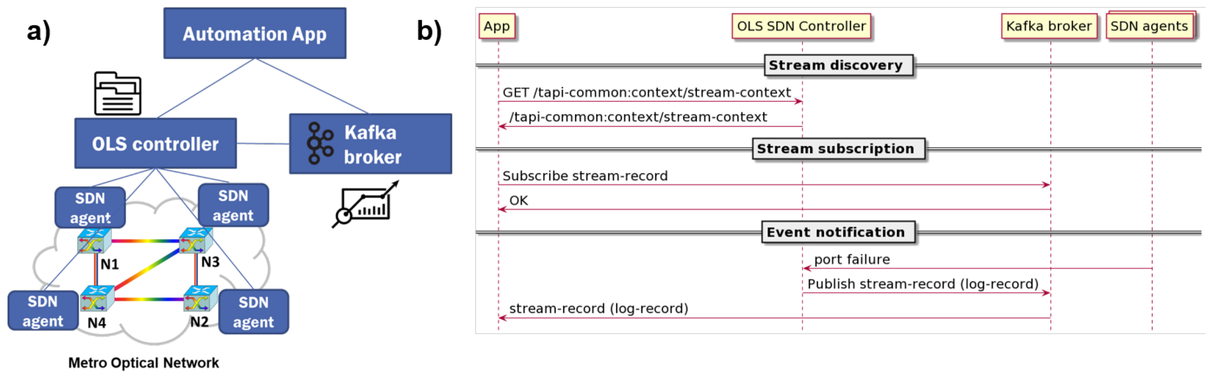


**Fig. 1:** a) Proposed architecture using automation application, Kafka broker, and OLS controller; and b) Sequence diagram of the proposed streaming use cases.

subscribed applications. Different types of updates are evaluated, including TAPI topology elements such as links and nodes as well as connectivity services and connections.

## Proposed Architecture

Fig. 1.a shows the proposed architecture to incorporate a Kafka broker for streaming telemetry. Firstly, an OLS controller is responsible for control and management of a disaggregated optical network consisting of several network elements, which are controlled using an SDN agent. The OLS controller exports its Transport API streaming capabilities through its RESTCONF server. Then, a Kafka broker is deployed in order to provide the necessary publish/subscribe mechanism for the data streams. Finally, a client application is deployed (consumer application) dedicated to analytics and network automation.

The application is able to retrieve the available data stream information from OLS controller, which refers to published data streams to the Kafka broker, so it can request a subscription to a data stream towards the Kafka broker. When a new event is generated (for example a link failure or a change in the status of optical spectrum), it is published by the OLS controller to the Kafka broker and distributed to the subscribed application. The (simplified) workflow is depicted in Fig. 1.b and is detailed in the following subsections: a) Stream discovery; b) Stream subscription; and c) Event notification.

Stream discovery The augmentation of the TAPI context with the so called stream-context allows the client to determine what specific stream connections are supported and available. The analysis of the stream-context offers the ability to identify and use various log strategies and stream connection protocols. The interface can offer many streams for a context. A variety of connection protocol, content, record strategy and storage strategy combinations might be offered. The available-streams allows the provider to report the streams that are currently available including their attributes such as connection-address, stream-state, supported-stream-type or connection-protocol.

Stream subscription In this use case, the automation application uses the provided endpoint address (i.e., Kafka broker) and method to connect (i.e., Kafka message brokering, using topic TAPI). Upon connection, both the application and the Kafka broker are synchronised and the application buffers the log events as appropriate. At the end of this operation the application will be well aligned at the head of the stream.

Event notification When a new event occurs, (e.g., detected and notified to the OLS controller through the SDN agent present in the Network Elements or generated by the OLS controller after a network state change caused by the triggering of a connectivity service from Operations Support System and/or Business Support System), the OLS controller publishes the corresponding log-record using the TAPI topic in the Kafka broker. Each log-record includes a header and a body. The log-record-header provides information common to all records, such as the tapi-context, a token as an identifier of the record allowing subsequent alignment, the event timestamp or the record type (such as update, delete or tombstone records).

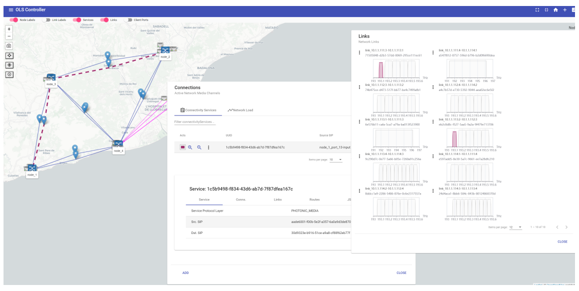
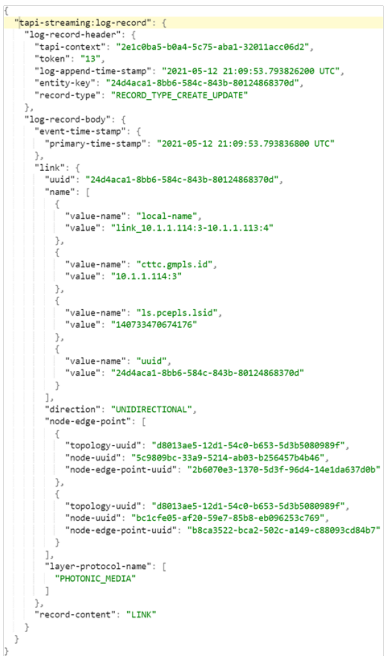


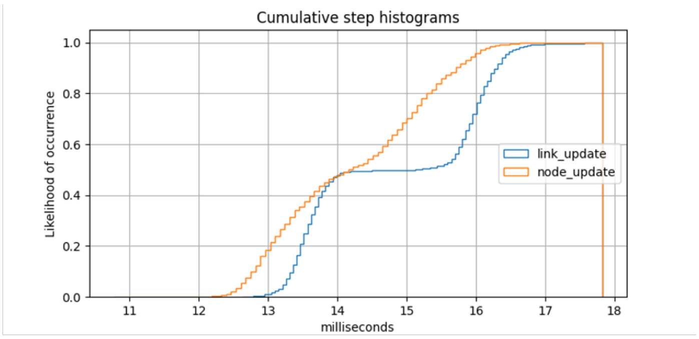
**Fig. 2:** OLS link/node status with active connectivity services

a)

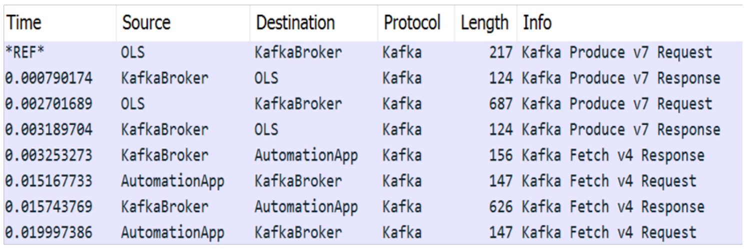


```
{
  "tapi-streaming:log-record": {
    "log-record-header": {
      "tapi-context": "2e1c0ba5-b0a4-5c75-aba1-32011acc06d2",
      "token": "13",
      "log-append-time-stamp": "2021-05-12 21:09:53.793826200 UTC",
      "entity-key": "24d4aca1-8bb6-584c-843b-80124868370d",
      "record-type": "RECORD_TYPE_CREATE_UPDATE"
    },
    "log-record-body": {
      "event-time-stamp": {
        "primary-time-stamp": "2021-05-12 21:09:53.793836800 UTC"
      },
      "link": {
        "uuid": "24d4aca1-8bb6-584c-843b-80124868370d",
        "name": [
          {
            "value-name": "local-name",
            "value": "link_10.1.1.114:3-10.1.1.113:4"
          },
          {
            "value-name": "cttc.gmpls.id",
            "value": "10.1.1.114:3"
          },
          {
            "value-name": "ls.pcepls.lsid",
            "value": "140733470674176"
          },
          {
            "value-name": "uuid",
            "value": "24d4aca1-8bb6-584c-843b-80124868370d"
          }
        ],
        "direction": "UNIDIRECTIONAL",
        "node-edge-point": [
          {
            "topology-uuid": "d8013ae5-12d1-54c0-b653-5d3b5080989f",
            "node-uuid": "5c9809bc-33a9-5214-ab03-b256457b4b46",
            "node-edge-point-uuid": "2b6070e3-1370-5d3f-96d4-14e1da637d0b"
          },
          {
            "topology-uuid": "d8013ae5-12d1-54c0-b653-5d3b5080989f",
            "node-uuid": "bc1cfe05-af20-59e7-85b8-eb096253c769",
            "node-edge-point-uuid": "b8ca3522-bca2-502c-a149-c88093cd84b7"
          }
        ],
        "layer-protocol-name": [
          "PHOTONIC_MEDIA"
        ]
      },
      "record-content": "LINK"
    }
  }
}
```

b)



c)



| Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|
| *REF* | OLS | KafkaBroker | Kafka | 217 | Kafka Produce v7 Request |
| 0.000790174 | KafkaBroker | OLS | Kafka | 124 | Kafka Produce v7 Response |
| 0.002701689 | OLS | KafkaBroker | Kafka | 687 | Kafka Produce v7 Request |
| 0.003189704 | KafkaBroker | OLS | Kafka | 124 | Kafka Produce v7 Response |
| 0.003253273 | KafkaBroker | AutomationApp | Kafka | 156 | Kafka Fetch v4 Response |
| 0.015167733 | AutomationApp | KafkaBroker | Kafka | 147 | Kafka Fetch v4 Request |
| 0.015743769 | KafkaBroker | AutomationApp | Kafka | 626 | Kafka Fetch v4 Response |
| 0.019997386 | AutomationApp | KafkaBroker | Kafka | 147 | Kafka Fetch v4 Request |

**Fig. 3:** a) Streaming log record example; and b) latency (in ms) for node and link updates.

## Experimental evaluation

In this section, we present the experimental evaluation of the proposed streaming architecture based using an OLS controller[5] and the ADRENALINE testbed. It has a hybrid fixed/flexi-grid DWDM core network with 4 white-box ROADM/OXC nodes, sliceable-bandwidth variable transceivers (S-BVTs) and 5 bidirectional amplified optical links of up to 150 km (610 km of G-652 and G.655 SMF total). Links have 100GHz CS, 50 GHz CS, or flexi-grid using commercially wavelength WSS. OLS controller (Fig. 2) provides ONF Transport API 2.1.3[6] photonic media layer. The network element agents implement the front-end towards the OLS with the OpenROADM device model or a REST interface with JSON.

A python automation application has been developed to subscribe to specific data streams and a Apache Kafka broker has been deployed. The automation application is able to received the log-records and analyse theirs content.

Fig. 3.a shows a generated log-record due to a link update. The header includes important information on the record-type, which in this example is of create/update, and on the tapi-context. In the body, it can be observed the timestamp of the event (this will later be used to measure the latency). It also includes the identifier of the record content (i.e., link), and the content itself.

Fig. 3.b shows the measured latency of two different event updates, link and node updates. To measure the latency, the timestamp available on the log-record body is used, while running the automation application in a synchronized server using Network Time Protocol (NTP). It can be observed that node update latency mean is of 15ms, while link update latency mean is of 15ms. Depending on the number of described information, we observe different latencies both for link and node updates. This will relate to the used network topology. The introduced measured latencies are not significant in comparison to typical connectivity provisioning times. The WireShark depicted in Fig. 3.c shows the message exchange between the OLS, the Kafka broker and the automation application. It can be observed the Kafka produce between OLS and Kafka broker, when an event is notified. Later, Kafka broker distributes a Kafka Fetch message towards automation application.

## Conclusions

This paper has presented a novel streaming mechanism based on ONF Transport API streaming and Kafka broker usage. Through event updates, context can be reconstructed by an external application. Latency measured has been demonstrated to be insignificant in comparison with connectivity provisioning times.

## Acknowledgements

Work partially supported by the EC H2020 TeraFlow (101015857) and Spanish AURORAS (RTI2018-099178-I00). The authors thank the ONF TAPI working group and Nigel Davis for feedback and clarifications on the use of the telemetry yang model and the streaming framework.

## References

[1] R. Vilalta, C. Manso, N. Yoshikane, R. Casellas, R. Martínez, T. Tsuritani, I. Morita, and R. Muñoz, "Experimental evaluation of control and monitoring protocols for optical sdn networks and equipment [invited tutorial]", *Journal of Optical Communications and Networking (JOCN)*, vol. 13, no. 8, pp. D1–D12, Aug. 2021.

[2] F. Paolucci and A. Sgambelluri, "Telemetry in disaggregated optical networks", in *2020 International Conference on Optical Network Design and Modeling (ONDM)*, IEEE, 2020, pp. 1–3.

[3] J. Kreps, N. Narkhede, J. Rao, *et al.*, "Kafka: A distributed messaging system for log processing", in *Proceedings of the NetDB*, vol. 11, 2011, pp. 1–7.

[4] A. Sadasivarao, S. Syed, D. Panda, P. Gomes, R. Rao, J. Buset, L. Paraschis, J. Brar, and K. Raj, "Demonstration of extensible threshold-based streaming telemetry for open dwdm analytics and verification", in *2020 Optical Fiber Communications Conference and Exhibition (OFC)*, IEEE, 2020, pp. 1–3.

[5] R. Casellas, F. J. Vılchez, L. Rodrıguez, R. Vilalta, J. M. Fàbrega, R. Martınez, L. Nadal, M. S. Moreolo, and R. Muñoz, "An ols controller for hybrid fixed/flexi grid disaggregated networks with open interfaces", in *2020 Optical Fiber Communications Conference and Exhibition (OFC)*, IEEE, 2020, pp. 1–3.

[6] A. Mayoral-López-de-Lerma, N. Davis, and A. Mazzini (editors), "Tapi v2.1.3 reference implementation agreement tr-547, v1.0", ONF, 2020.

[7] N. Davis (editor), "Tapi v2.1.3 reference implementation agreement, tr-548, streaming (draft)", ONF, 2021.