An Intelligent Optical Telemetry Architecture

Luis Velasco*, Pol González, and Marc Ruiz

Optical Communications Group (GCO), Universitat Politècnica de Catalunya (UPC), Barcelona, Spain
e-mail: luis.velasco@upc.edu

Abstract: A distributed telemetry system is proposed with agents receiving and analyzing data before sending to a centralized manager. Intelligent data aggregation on optical constellations telemetry largely reduces data rate without introducing significant error. © 2023 The Authors

1. Introduction

The benefits of telemetry for optical networking have been shown in the literature and several telemetry architectures have been defined (see, e.g., [1], [2]). In general, telemetry data is collected from *observation points* in the devices and send to a central system running besides the Software Defined Networking (SDN) controller. Although protocols specifically devised for telemetry, like gRPC, compress data, the amount of data that can be collected and the frequency of collection make those architecture not practical. In this paper, we propose a telemetry architecture that supports intelligent data aggregation nearby data collection.

2. Telemetry Architecture

Fig. 1 presents the reference network scenario, where an SDN architecture controls a number of optical nodes, specifically optical transponders (TP) and reconfigurable optical add-drop multiplexers (ROADM), in the data plane. Note that the SDN architecture might include a hierarchy of controllers, including optical line systems and parent SDN controllers. A centralized *telemetry manager* is in charge of receiving, processing and storing telemetry data in a telemetry database (DB). Some data exchange between the SDN control and the telemetry manager is needed, e.g., the telemetry manager needs to access the topology DB describing the optical network topology, as well as the label switched path (LSP) DB describing the optical connections (theses DBs are not shown in Fig. 1). Every node in the data plane is locally managed by a node agent, which translates the control messages received from the related SDN controller into operations in the local node and exports telemetry data collected from observation points (labeled M) enabled the optical nodes.

A detailed architecture of the proposed telemetry system is presented in Fig. 2, where the internal architecture of *telemetry agents* inside node agents and the telemetry manager is shown. Internally, both, the telemetry agent and manager are based on three main components: *i*) a *manager* module configuring and supervising the operation of the rest of the modules; *ii*) a number of modules that include *algorithms*, e.g., data processing, aggregation, etc. and *interfaces*, e.g., gRPC; and *iii*) a *Redis DB* that is used in *publish-subscribe* mode to communicate the different modules among them. This solution provides an agile and reliable environment that simplifies communication, as well as integration of new modules. A gRPC interface is used for the telemetry agents to export telemetry to the telemetry manager, as well for the telemetry manager to tune the behavior of algorithms in the agents.

Let us describe now a typical telemetry workflow valid for a wide range of use cases. The node agent includes modules (denoted data sources) that gather telemetry data from observation points in the optical nodes. Examples include optical spectrum analyzers (OSA) in the ROADMs and data from digital signal processing, e.g., optical constellations, in the TPs. A telemetry adaptor has been developed, so data sources can export collected data to the telemetry system; specifically, the adaptor receives raw data from the data source and generates a structured json object, which is then published in the local Redis DB (labeled 1 in Fig. 2). The periodicity for data collection can be configured within a defined range of values. A number of algorithms can be subscribed to the collected measurements. In this example, let us assume that only one algorithm is subscribed, which processes the measurements locally. Such processing might include doing: *i*) no transformation on the data (*null* algorithm); *ii*) some sort of data aggregation, feature extraction or data compression; or *iii*) some inference (e.g., for degradation detection). The output data (transformed or not) are sent to a gRPC interface module through the Redis DB (not shown in the figure) (2), which conveys the data to the telemetry manager. Because gRPC requires a previous definition of the data to be conveyed, our implementation encodes the received data in base64, which allows generalization of the telemetry data to be conveyed. Note that, although such encoding could largely increase the volume of data to be transported, intelligent data aggregation performed by telemetry agents could reduce such volume to a minimum.

In the telemetry manager, the data are received by a gRPC interface module that publishes them in the local Redis DB, so subscribed algorithms can receive them. The algorithms in the telemetry manager can implement functions related to data aggregation, inference, etc. Once processed, the output data is published in the local Redis DB (4) and can be stored in the telemetry DB (5) and/or be exported to external systems (6). Interestingly, algorithms in the telemetry manager can communicate with those in the telemetry agents using the gRPC interface (7-8). Examples of such communication include parameter tuning, among others.

The research leading to these results has received funding from the H2020 B5G-OPEN (G.A. 101016663) and the MICINN IBON (PID2020-114135RB-I00) projects and from the ICREA Institution.

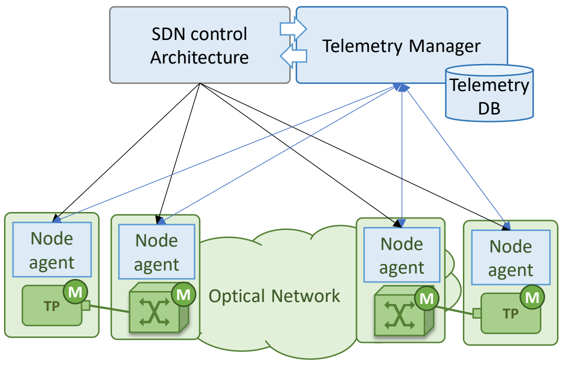

Fig. 1: Overall network architecture

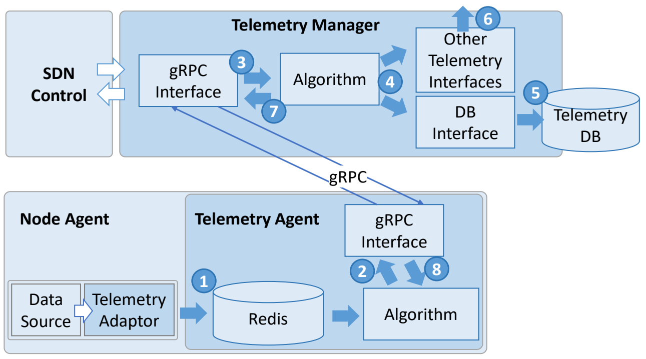

Fig. 2: Proposed telemetry architecture

3. Intelligent Data Aggregation for Optical Constellations Telemetry

We now focus on introducing several techniques to greatly reduce the data volume that needs to be conveyed through the gRPC interface connecting telemetry agents to the manager. In particular, we analyze: *i*) data compression using *autoencoders*; *ii*) supervised *feature extraction*; and *iii*) *data summarization* using the arithmetic mean of a number of observations. For this example, let us assume the case where the observation point is in a TP, which gathers the received optical symbols of a *m*-QAM signal. The related data source then, periodically retrieves a constellation sample *X* (a sequence of *k* IQ symbols as represented in Fig. 3a for a 16-QAM signal) and publish it in the local Redis DB.

Let us start with the use of autoencoders, a type of neural network with two components: the *encoder*, which maps input data into a lower-dimensional *latent* space, and the *decoder*, which gets data in the latent space and reconstructs the original data back. Once trained, the autoencoder takes as input 2×*k* values, i.e., $[x_1^I, x_1^Q, \ldots x_k^I, x_k^Q]$, from the received constellation sample and generates the latent space $Z=[z_1, \ldots, z_L]$, where the size of *Z* is significantly lower than that of *X* (Fig. 3b). In this case, the encoder runs as an algorithm module in the telemetry agent and exchanges *Z* for every input sample *X* with the decoder running in the telemetry manager through the gRPC interface. The algorithm in the telemetry manager uses the decoder to reconstruct the constellation sample and it stores the result in the telemetry DB.

Let us now consider supervised feature extraction. In our previous work in [3], we applied Gaussian Mixture Models (GMM) [4] to characterize each constellation point of an optical constellation sample as a bivariate Gaussian distribution (Fig. 3c). Therefore, each constellation point *i* is characterized by 5 features, the mean position in I and Q axes $[\mu^I,\mu^Q]$, as well as the I and Q variance and symmetric covariance terms that the symbols belonging to the constellation point *i* experience around the mean $[\sigma^I,\sigma^Q,\sigma^{IQ}]$. Therefore, for an *m*-QAM signal, *m**5 features need to be propagated from the telemetry agent to the manager.

With the two previous intelligent data aggregation techniques, telemetry data is propagated from the observation point to the telemetry manager with the same frequency, i.e., every time a new constellation sample is collected from the observation point, a subset of data representing it is generated and conveyed to the telemetry manager. Assuming a high collection frequency, this policy entails large volume of data being conveyed. However, in normal conditions, this is not needed in general. Hence, we could measure variations in the computed features to decide whether a representation of the new sample needs to be sent to the telemetry manager. In case of no significant variations in the features, the telemetry agent can send averaged values of the features with a much lower frequency, thus reducing the volume of telemetry data being conveyed.

4. Illustrative Results

The telemetry agent and the telemetry manager have been implemented in Python and deployed as containers in two different virtual machines. InfluxDB 2.4 implementing the telemetry DB was deployed as a separated container. A data source was developed that emulates constellation samples collection from an observation point in a TP (available in [5]). Each constellation sample *X* includes *k*=2048 symbols from a 16-QAM optical signal. In consequence, the size of each sample *X* is 2×2048×4 = 16,384 bytes (B), assuming that every symbol is represented with two scalars (I and Q) of 32 bits. The telemetry adaptor in the data source publishes samples *X* encoded as a json object. A representation of the json object is shown in Fig. 4 identified with the same label as the related message in Fig. 2 for the sake of clarity.

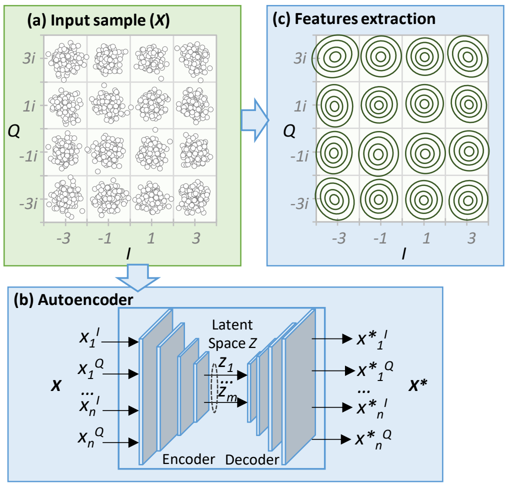

Fig. 3: Constellation sample (a) and use of autoencoders (b) and supervised features extraction (c).

1
```
{
"X": [[-3.08519822419515, 2.98467451952321],
                          :
        [2.96258763128303, 2.97984558299648]]
}
```
2a
```
{
  "Z": ["2.732774257",
                :
        "0.172829926"]
}
```
2b
```
{
 "F": [["-2.971842458","2.971601309","0.004633636",..."0.004507746"],
                                    :
        ["-2.941472597","2.931215828","0.113045212",..."0.121031438"]]
}
```
Fig. 4: JSON objects

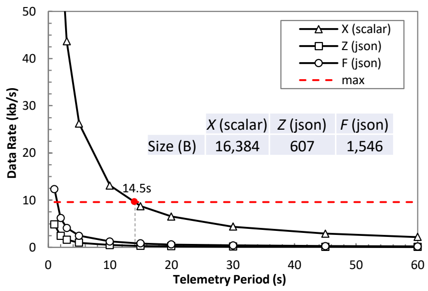

Fig. 5: Telemetry data rate vs period

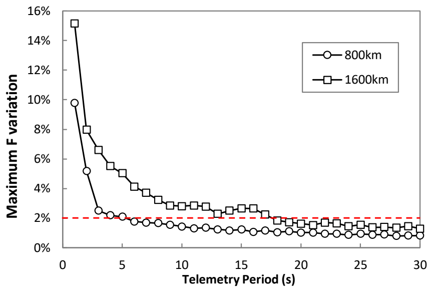

Fig. 6: Aggregation error vs telemetry period

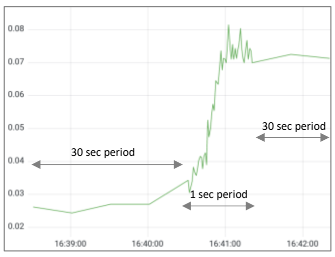

Fig. 7: Data summarization

Algorithms have been implemented in Python and deployed in the telemetry agent and manager for the intelligent data aggregation techniques detailed in Section 3. In case of using autoencoders for data compression, the encoder runs in the telemetry agent, whereas the decoder runs in the telemetry manager. The algorithm in the telemetry agent is subscribed to messages from the data source, so it takes as input a constellation sample *X* and generates the latent space *Z* representing it. We have trained the autoencoder for the maximum compression that produces a reproduction error in the decoder lower than 2%, which results in vectors *Z* of size 32. Such vectors are output as json objects, where each component of the vector is represented as a string with 11 characters, resulting is 455 characters in total for the json object (2a in Fig. 4). The json object is then compressed, so each character uses only 1B and encoded in base64, which results in 607B. When the message arrives at the telemetry manager through the gRPC interface, it is used as input to the decoder in the related algorithm. The decoder generates a sample *X**, which is finally stored in the telemetry DB. In our tests, both, data encoding and decoding took 60ms.

In the case of supervised feature extraction, the algorithm in the telemetry agent applies GMM fitting to every constellation sample *X* received and generates outputs of *m*=16 vectors with 5 features each (denoted *F*). This process, outputs a json object with 1,159 characters (2b in Fig. 4), which is then conveyed through the gRPC interface using 1,545B. To compare the results of features extraction to those from the autoencoder, the algorithm in the telemetry manager, samples each distribution to obtain constellation samples with 2,048 symbols, and stores them in the telemetry DB.

Fig. 5 presents the telemetry data rate when the telemetry period ranges from 1s to 1min for gRPC messages with: *i*) samples *X* using scalar values; *ii*) *Z* vectors with the latent space encoded as json objects; and *iii*) features *F* encoded as json objects. The inset in Fig. 5 summarizes the size of every object. We observe that using constellation samples for the telemetry results in extremely large data rates, which limits the telemetry period. The reason is double; on the one hand, high data rates would require expensive data communication infrastructure dedicated for network telemetry, and on the other, optical devices (in this case, the TP) would need to support them, which might impact its performance. Therefore, assuming a maximum data rate for telemetry collection of 9600 b/s (e.g., for a typical serial interface), the minimum telemetry period for optical constellations would be 14.5s. With such period, the telemetry data rate reduces one order of magnitude to only hundreds of b/s.

Telemetry data rate can be further reduced by implementing data summarization on the extracted features, which can be sent with a larger period in case no significant changes occur. To illustrate this, Fig. 6 shows the relative maximum difference of features *F* when the telemetry period ranges 1-30 s, assuming collection period of 1s. Telemetry data of two lightpaths of 800km and 1600km are shown. In both cases, increasing the telemetry period above 20 s achieves negligible aggregation error under 2%. Based on this, a dynamic telemetry data aggregation has been implemented for the extracted features from optical constellations (Fig. 7). The telemetry agent aggregates feature values and sent them every 30 s if no meaningful variation is detected. However, as soon as some variation is detected, the telemetry agent sends the extracted features as soon as they are processed, i.e., every 1s, so the related algorithm in the telemetry manager can analyze the data with fine granularity. As soon as the variations disappear, the algorithm in the telemetry manager asks the one in the agent to aggregate data again.

We can conclude that having fine grain telemetry for network data analysis at a centralized location results in a large amount of data to be conveyed from the devices. To solve that fact, a distributed telemetry architecture has been proposed, where collected data is analyzed before sending them to the central location. Intelligent data aggregation has been shown for optical constellations telemetry, where aggregation introduces negligible error.

References

[1] L. Velasco *et al.*, "Monitoring and Data Analytics for Optical Networking: Benefits, Architectures, and Use Cases," IEEE Network, 2019.
[2] F. Cugini *et al.*, "Telemetry and AI-based security P4 applications for optical networks," IEEE/OPTICA JOCN, 2022.
[3] M. Ruiz *et al.*, "Deep Learning -based Real-Time Analysis of Lightpath Optical Constellations," IEEE/OPTICA JOCN, 2022.
[4] N. Bouguila and W. Fao, *Mixture Models and Applications*, Springer, 2020.
[5] M. Ruiz *et al.*, "Optical Constellation Analysis (OCATA)," https://doi.org/10.34810/data146, 2022.