CHALMERS
UNIVERSITY OF TECHNOLOGY

Microservice-Based Unsupervised Anomaly Detection Loop for Optical Networks

Microservice-Based Unsupervised Anomaly Detection Loop for Optical Networks

Carlos Natalino[1], Carlos Manso[2], Lluis Gifre[2], Raul Muñoz[2], Ricard Vilalta[2], Marija Furdek[1], Paolo Monti[1]

1. Electrical Engineering Department, Chalmers University of Technology, Gothenburg, Sweden.
2. Centre Tecnologic de Telecomunicacions de Catalunya (CTTC/CERCA), Spain.
carlos.natalino@chalmers.se

Abstract: Unsupervised learning (UL) is a technique to detect previously unseen anomalies without needing labeled datasets. We propose the integration of a scalable UL-based inference component in the monitoring loop of an SDN-controlled optical network.

1. Introduction

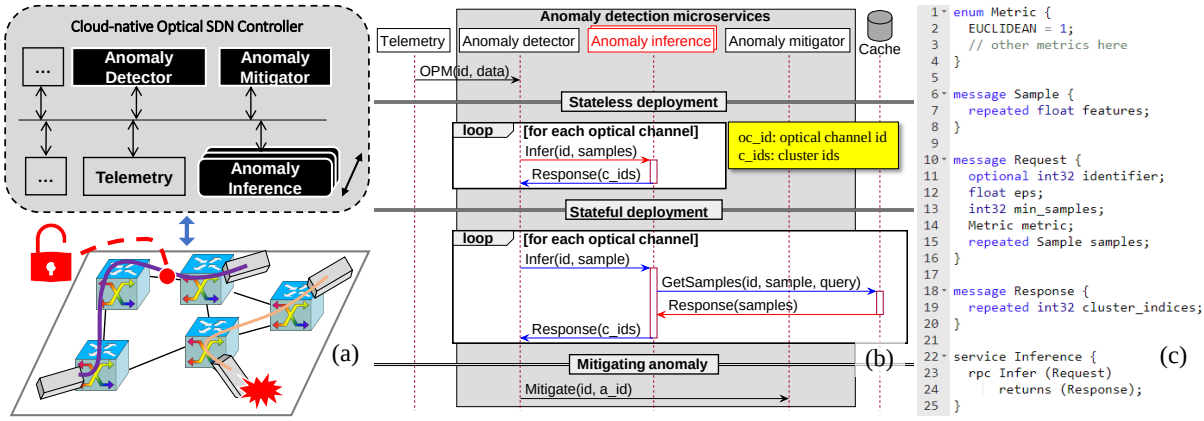
Machine Learning (ML) models are expected to enable the automation of several operations in optical networks. Thanks to Optical Performance Monitoring (OPM) and network telemetry [1], more data are available, making ML the perfect candidate to support provisioning, monitoring, maintenance, and quality assurance of Optical Channels (OChs) [2]. Among them, ML-based Anomaly Detection (AD) is used to identify deviations from normal operating conditions caused, for example, by performance degradation in optical devices, misconfiguration, or even malicious activities [3–5]. Unsupervised Learning (UL) models are a promising candidate for the AD task. They can detect anomalies using only current OPM data, without the need of models trained over previously collected labeled datasets. One challenge with UL is scalability, since AD needs to be periodically performed over all the OChs in the network. Executing UL models for several OChs over short monitoring intervals can become impractical if models are not implemented and deployed appropriately. The work in [6] shows how a microservice-based architecture improves the efficiency of the monitoring task in optical networks. However, the authors considered only the application of Supervised Learning (SL) models, which have resource-intensive training but lightweight inference. UL models, on the other hand, do not require training, but have resource-intensive inference due to the need to process tens or hundreds of samples.

Leveraging the microservice-based architecture presented in [7], this work presents two deployment options, i.e., *stateless* and *stateful*, for AD using UL models. For the stateless option, each inference request includes all samples to be processed, and the inference component does not store any samples between requests. For the stateful option, the inference component keeps a cache with the samples processed so far. As a result, an inference request needs to include only the latest (i.e., newest) samples to be processed. The paper assesses the pros and cons of both deployment options, coming to the conclusion that the stateless one is the most appropriate in terms of scalability for the anomaly detection use case. Performance of a stateless anomaly inference component based on the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm is then validated in an experimental setting showing the ability to scale the monitoring to up to 960 OChs while maintaining a suitable response time (i.e., between 5 and 20 ms).

2. Microservice-Based Architecture for Unsupervised Anomaly Detection in Optical Networks

This work considers the microservice-based optical Software-Defined Networking (SDN) controller proposed in [7]. Within the controller we propose 3 new components (Fig. 1a) for detecting anomalies in the OChs established over the physical infrastructure. The *anomaly detector* performs anomaly cognition, i.e., it obtains OPM data, interacts with the anomaly inference, and consolidates the inference results. The *anomaly inference* hosts the AD model; it receives the raw OPM samples, executes the model over them, and returns the model result. Finally, the *anomaly mitigator* computes, upon detection an anomaly, a solution to mitigate the effect of the anomaly.

Detecting anomalies using UL requires the processing of a given number of samples. This number defines the accuracy of the inference results. In this work, we assume that the anomaly inference component requires W samples to perform AD. The anomaly detection loop works as follows. The telemetry component collects OPM samples from the device and/or the OCh being monitored every T time units. The anomaly detector queries the inference component every D time units, with $D \geq T$. Two options are available when deploying the anomaly inference component. With a *stateless* option, each request from the anomaly detector contains all W samples to be processed. With a *stateful* option, an extra cache component is deployed in the controller, which stores the latest W samples associated with a particular device or OCh. As a result, a request to the anomaly inference by




```
enum Metric {
  EUCLIDEAN = 1;
  // other metrics here
}

message Sample {
  repeated float features;
}

message Request {
  optional int32 identifier;
  float eps;
  int32 min_samples;
  Metric metric;
  repeated Sample samples;
}

message Response {
  repeated int32 cluster_indices;
}

service Inference {
  rpc Infer (Request)
      returns (Response);
}
```

Fig. 1. (a) Anomaly detection components; (b) Messages exchanged among the proposed microservices; and (c) Messages and services exposed by the anomaly inference component.

the anomaly detector contains only the latest $n=\lceil D/T \rceil$ OPM samples, while the inference component obtains the remaining and most recent $W-n$ samples from the cache. Regardless of the deployment option (i.e., stateless or stateful) a request to the anomaly inference results with an indication whether or not each one of the W samples was detected as an anomaly. This is done by assigning a cluster ID value to each sample.

Fig. 1b shows the message exchange of the anomaly detection loop. First, the anomaly detector obtains the OPM data from the telemetry component. In case of a stateless deployment, the anomaly detector queries the anomaly inference component with a message including all of the W samples. In case of a stateful deployment, the anomaly detector queries the anomaly inference component with a message including only n samples, while the anomaly inference component retrieves the remaining $W-n$ OPM samples from the cache. Regardless of the deployment option, the anomaly inference returns the cluster ID values for all W samples. If an anomaly is detected, the anomaly detector queries the anomaly mitigator, specifying the ID of the affected device or OCh and the anomaly ID value. It becomes clear that W impacts not only the accuracy of the AD model, but also impacts the size of the anomaly inference and cache messages. Depending on the deployment option (i.e., stateless or stateful) we need to carefully tune the value of W. Fig. 1c shows the *gRPC*-like (gRPC Remote Procedure Call) data model used to define the anomaly inference microservice, the related messages, and their respective parameters [8], inspired by the Scikit-Learn Application Programming Interface (API) design [9]. The figure is specific for the DBSCAN algorithm, but the model can be adapted to other anomaly detection algorithms. With the DBSCAN algorithm, we need to specify the metric used to compute the distance between each sample pair (we use the Euclidean distance as an example in Fig. 1c). Each OPM sample is defined in terms of floating point numbers that represent the values of each feature. A detection request includes the ε (*eps*) and *MinPts* (*min_samples*) parameters of the DBSCAN algorithm, as well as the distance metric to be used and the set of samples to be processed. It may also include an identifier of the source of the OPM sample (i.e., the device or OCh ID). A detection response contains as many integer numbers as there are samples, representing the cluster ID of each sample. In our implementation, we use -1 to indicate an anomaly, while normal clusters are identified by positive integers.

When comparing the stateless and stateful deployment options, the following observations can be made. In a stateless deployment, the inference component is simpler and does not have to manage the interface with the cache. Not having to deploy and maintain an extra cache also simplifies the architecture of the microservice-based SDN controller. It also eliminates the need to query the cache for data at every inference. Therefore, in the performance assessment work described in the next section we consider a stateless deployment only.

3. Performance Assessment

In this section, we focus on the anomaly inference component and its interaction with the anomaly detector. We are interested in evaluating the impact of the number of samples W and the number of monitored OChs on the system performance. We implemented the components from Fig. 1b using gRPC and the Rust programming language, and deployed them over a Kubernetes cluster, which is responsible for monitoring and scaling the components as their resource utilization varies. We enabled a Linkerd network mesh to perform load balancing among the anomaly inference replicas [10].

We evaluate first how many samples are necessary to achieve the desired anomaly detection accuracy. For this assessment, we use the physical layer security dataset described in [5] with 33 features collected from a coherent transceiver for every sample. Besides the normal operating conditions, the dataset contains 3 different attack strategies, each one with two intensities, for a total of six different attack scenarios. We build a detection request containing 30 samples from one of these attack scenarios (chosen at random), and vary the number of samples

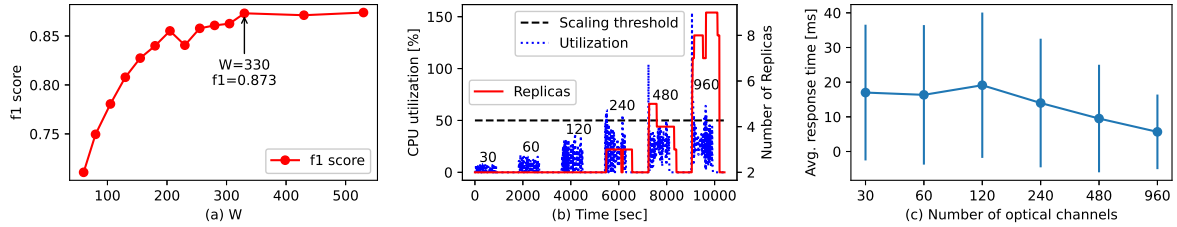


Fig. 2. (a) Accuracy of the AD model; (b) CPU utilization and number of replicas over time (insets represent the number of active OChs); (c) Average response time and its standard deviation when varying the number of active OChs.

for normal operating conditions between 30 and 500. Fig. 2a shows the accuracy of the AD model averaged over 50 experiments as a function of W. The accuracy is measured using the *f1 score* which ranges from 0 to 1, with 1 representing the highest accuracy. Accuracy increases with up to 330 samples, while it slightly degrades after that. This is why for the scalability performance assessments we assume W=330. We vary the load of the inference component by changing the number of actively monitored OChs. The anomaly inference component was configured to keep a minimum of two replicas, each with guaranteed 300 mCPUs and able to reach up to 500 mCPUs, and scale whenever its average CPU utilization over all replicas exceeds 50% of the guaranteed CPUs. For a particular number of OChs, we generate detection requests every D=30 seconds during 15 minutes. Between tests, we stop the anomaly detection requests for 10 minutes to allow the system to return to the initial state.

Three properties of the inference component are of interest: *(i)* CPU utilization, *(ii)* number of replicas, and *(iii)* response time. Fig. 2b shows the relative CPU utilization (the % of CPUs used over the total CPUs guaranteed to the component) and the number of replicas over time. Two replicas are sufficient to support the anomaly inference for up to 120 OChs. For a higher number of OChs, we need more replicas. We also note that when the CPU utilization increases beyond 50%, the component is quickly scaled (i.e., a new replica is instantiated) and the CPU utilization stabilizes below 50%. We also monitor the average response time and its standard deviation experienced by the anomaly detector after each detection request as a function of the number of OChs (Fig. 2c). Thanks to the scaling of the anomaly inference component, the response time does not increase with the number of monitored OChs. The average varies between 20 and 5 ms, which can be considered a very good response time given the complexity of the DBSCAN algorithm. When compared to the response time reported in [6] for an SL model, i.e., an Artificial Neural Network (ANN) with a response time of 1 ms, performing a similar ML task, our UL model is $5-20x$ slower. Nevertheless, the response time is suitable for a 30-seconds monitoring interval with a significant number of monitored OChs.

4. Conclusions

In this work, we proposed a scalable anomaly detection loop assisted by UL models suitable for optical network infrastructures controlled by microservice-based SDN controllers.After considering pros and cons of both stateless and stateful deployment options, we use the former for a performance assessment analysis. The obtained results highlight the relation between accuracy and the number of samples processed during each inference. Moreover, our analysis shows that by properly scaling the number of replicas, it is possible to maintain a relatively low response time regardless of the number of monitored OChs.

Acknowledgements: Work partially supported by the EC H2020 TeraFlow (101015857), Spanish AURORAS (RTI2018-099178-I00) and Vetenskapsrådet (2019-05008). Upon publication, the source code of this work will be available at https://github.com/carlosnatalino/2022-OFC-unsupervised-learning.

References

1. F. Paolucci *et al.*, JLT **36**, 3142–3149 (2018). DOI: 10.1109/JLT.2018.2795345.
2. D. Rafique *et al.*, JLT **36**, 1443–1450 (2018). DOI: 10.1109/JLT.2017.2781540.
3. S. Varughese *et al.*, in *Proc. of OFC,* (2019), p. W2A.46.
4. X. Chen *et al.*, JLT **37**, 1742–1749 (2019). DOI: 10.1109/JLT.2019.2902487.
5. C. Natalino *et al.*, J. Light. Technol. **37**, 4173–4182 (2019). DOI: 10.1109/JLT.2019.2923558.
6. C. Natalino *et al.*, in *Proc. of ECOC,* (2021), p. We3E.2.
7. R. Vilalta *et al.*, in *Proc. of ECOC,* (2019), p. Tu.3.E.2. DOI: 10.1049/cp.2019.0874.
8. "gRPC remote procedure call," https://grpc.io/. Accessed: 2021-10-19.
9. L. Buitinck *et al.*, in *ECML PKDD Workshop,* (2013), pp. 108–122. ArXiv 1309.0238.
10. "Linkerd service mesh," https://linkerd.io/. Accessed: 2021-10-19.