Network Intelligence in Action: the DAEMON Perspective

Livia Elena Chatzieleftheriou, Marco Gramaglia, Marco Fiore, Nina Slamnik-Kriještorac, Miguel Camelo, Paola Soto, Evangelos Kosmatos, Andres Garcia-Saavedra, Michele Gucciardo

***Abstract*—The native integration of AI and ML algorithms in the next-generation mobile network architecture will allow for meeting the expectations of 6G. This aspect is targeted by the DAEMON project, which proposed a solution to natively manage Network Intelligence (NI) through novel architectural elements and procedures. In this paper, we discuss how NI solutions based on AI and ML can leverage NI native procedures implemented by the NI Orchestrator to improve their lifecycle management. We also discuss how the architectural procedures can be implemented in practice, using state-of-the-art software components.**

***Index Terms*—Network Intelligence, Network Intelligence Orchestration; Intelligence Plane**

I. Introduction

The roadmap toward 6G networks entails the native integration of Artificial Intelligence (AI) algorithms into the network architecture. This is one of the main outcomes of the DAEMON project[1], that designed a native architecture for the next generation of mobile networks [1]. Thanks to this, details related to Network Intelligence (NI) functions empowered by AI/ML, can now leverage this native architecture and provide advanced features. In [2] we defined the Network Intelligence Orchestrator (NIO), which orchestrates the Network Intelligence Stratum in the network, introducing elements that deal with specific aspects such as MLOps, Knowledge management, and explainability. Thanks to this functionality, the NI solutions that are deployed in the network can integrate their lifecycle management, getting from these common modules relevant information in the same way orchestration and management systems manage traditional network functions.

These procedures have been specified also in [3], defining a clear interaction between elements to fulfill procedures, e.g., conflict resolution, a fundamental tool to coordinate different NI algorithms running at the same time in the network. To facilitate this operation, in [1] we defined the concept of Network Intelligence Service (NIS), which hierarchically builds on Network Intelligence Functions (NIFs) and NIF components (NIF-Cs), the atomic elements of a service that carries a specific operation such as gathering data from a source or enforcing decisions to the network. This approach, which follows the N-MAPE-K [1] framework, offers the maximum feasibility for NI orchestration.

M. Gramaglia is with University Carlos III de Madrid (UC3M). L.E. Chatzieleftheriou, M. Fiore, and M. Gucciardo are with the IMDEA Networks Institute. M. Camelo-Botero, P. Soto, and N. Slamnik-Kriještorac are with University of Antwerp - imec, IDLab. A. Garcia-Saavedra is with NEC Laboratories Europe. E. Kosmatos is with WINGS ICT Solutions.

[1]https://h2020daemon.eu/

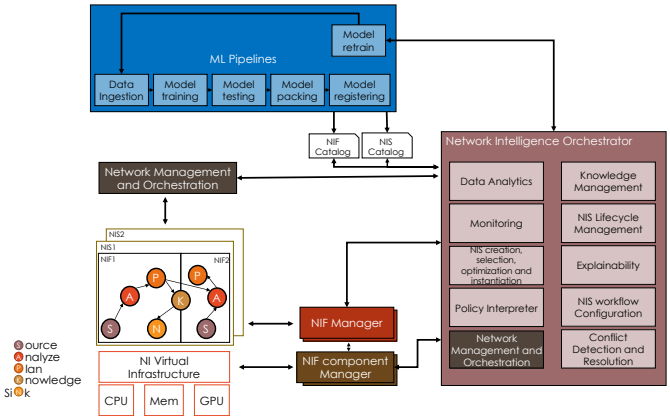

Fig. 1. The Network Intelligence Stratum (NI Stratum) and the functional blocks of the Network Intelligence Orchestrator and ML pipelines [1].

In this paper, we provide a detailed discussion of how specific NI functions (especially the ones related to network access, that have particularly stringent timing requirements) can leverage the procedures set by the NIO. In Section II, we review the procedures and their role in the operation of NI at the edge of the network. We go deeper in SectionIII, discussing how three NIS leverage those procedure to achieve i) Conflict resolution, ii) explainability, and iii) knowledge sharing. Finally, we discuss a reference implementation of the conflict resolution functionality in SectionIV, proving the feasibility of our proposal, before concluding in SectionV.

II. NI Native Architecture and Procedures

The NIFs as discussed in [1] are the fundamental building blocks of the NI Orchestration architecture [2]. Hence, the proper orchestration of such blocks is the preliminary step required for the integration of Network Intelligence, especially when it is deployed very close to the user plane or the edge of the network, that operates in real or near-real-time timescales.

Figure 1 illustrates the key functional blocks defined by the DAEMON architecture, which have been specifically designed to facilitate the seamless integration of NI within network operations. In the following discussion, we will delve into how the solutions use these specific components discussed in [3].

A. ML Pipeline

The ML procedures play a vital role in preparing data-driven models that enable the implementation of the various functionalities envisioned by the project. When dealing with

fast timescale solutions, as the ones we discuss in this paper, this aspect must consider two key characteristics:

- The incoming data rate may be significantly high. For instance, consider a 5G Distributed Unit (DU) at the edge that has to make scheduling decisions based on data coming from several Radio Units (RUs). Thus, it is essential that the training of algorithms, like Reinforcement Learning (RL), occurs independently from the network operations. This ensures the decoupling between the two.
- The registered models need to be sufficiently flexible to adapt to potentially changing conditions, such as those encountered in the radio domain. In this regard, it is crucial to explore the possibility of leveraging continuous training across different scenarios to improve the flexibility of the different models. An example of such a model is the one related to the Machine Learning Radio Resource Scheduling discussed in paper [4], which is constantly monitored by the conflict resolution engine discussed in Section III-A. In this way, it can be replaced according to the changing environmental conditions, such as more congested computing resources.

B. NI Orchestrator

The NIO plays a pivotal role in facilitating the lifecycle management of network intelligence within the DAEMON project. Working in tandem with the NIF Managers and the NIF Component Manager, which handle lower-level factors, such as container management for specific intelligence components, the NIO encompasses several modules that are of paramount importance within the scope of WP3.

1) Analytics and Monitoring: Similar to the ML pipeline discussed earlier, effective monitoring of intelligence deployed at the edge is crucial for ensuring proper system operation. This monitoring task, which is carried out by numerous solutions for the autonomous management of the network, such as the ones described in [3], plays a vital role. By collecting metrics related to the performance of intelligence algorithms (such as precision and accuracy for classification algorithms or errors for forecasting algorithms), the NIO can assess the behavior of NIFs or specific NIF Components (NIF-C). Consequently, appropriate actions can be taken if performance degradation occurs. These actions may include selecting a new model from the catalog or implementing corrective measures.

A possible countermeasure to reduce the effects of suboptimal NI decisions leveraging models without representing the network context is to temporarily disable such action-taking, applying them in a "sandbox", and understanding which input contexts cause such misbehavior. This will help to identify what actions are causing conflicting decisions.

However, achieving real-time monitoring of these operations may be challenging due to the fast decision-making pace of some algorithmic families, e.g. those discussed in Section III.

2) Knowledge Management: The generalization of ML models holds immense significance in dynamic and rapidly changing environments, particularly in the edge networks discussed in [3], allowing them to adapt and perform effectively without needing frequent re-training or extensive model modifications. In these fast-varying environments, where conditions and data distributions can rapidly shift, ML models with strong generalization capabilities excel at handling such variations while consistently providing accurate results. Nevertheless, achieving generalization requires training models over large amounts of input data, which could be difficult depending on the type of network functions (especially for access functions that account for a large number of input variables.

Our ML models are trained to capture the underlying patterns and dynamics of the network, empowering them to make accurate predictions and informed decisions even under previously unseen scenarios. Their avoidance of constant re-training results in significant savings in computational resources and minimizes disruptions to network operations. Thus, developing ML models with robust generalization capabilities [5] becomes a critical factor in enabling efficient and reliable performance within fast-varying environments as the radio access network. In this context, effective knowledge management, including continuous exchange of information between intelligence algorithms running in various parts of the network can significantly enhance the generalization of these models. This collaborative approach to knowledge management enhances the overall network operation by leveraging collective intelligence and improving the generalization capabilities of ML models.

3) Explainability: AI and ML interpretability is a rapidly growing field, gaining significant attention [6]. The interest in explainability lies in evaluating learned models and building trust among their users. However, despite the evaluation of AI and ML methods using specific metrics, a consensus on the definitions of interpretability and explainability is yet to be reached. Various disciplines are actively studying this problem [7]. Currently, it remains unclear which characteristics are essential for the interpretation of a black-box model to generate outputs that are relevant to a specific user or problem [8]. In this paper, we support this view by including an explainability and interpretability module in the NIO design.

As discussed more in detail in [4], we propose the concept of "actionable" Machine Reasoning (MR) [9], which involves making additional decisions based on the ones already taken by AI/ML models. This approach is particularly suitable for managing 5G networks, as this task operates on a different time scale compared to other AI/ML-based operations. This concept is a good fit, e.g., for the O-RAN ecosystem, as it natively applies to the different modules of the system. We leverage these techniques to analyze the behavior of underlying AI/ML rApps/xApps/dApps, and dynamically re-orchestrate resources to meet operator-defined requirements in the context of ML-based radio resource scheduling.

4) Conflict Resolution: In the NI vision defined by the project, multiple NIFs work collaboratively to establish a Network Intelligence Service (NIS) that optimizes and coordinates a suite of NIFs toward a shared objective, such as sustainability. However, coordinating these algorithms may introduce short transients where the decisions made by each algorithm could deviate due to factors like data distribution

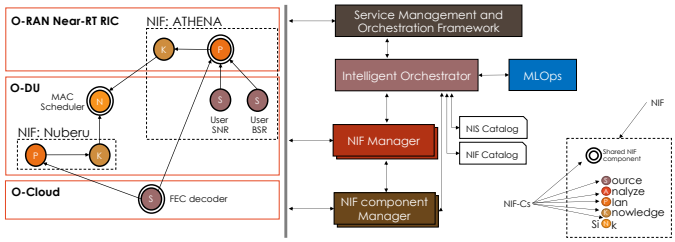

Fig. 2. Integration of Nuberu [10] and ATHENA [4], NIFs related to the Reliable Distributed Unit and the Radio Resource Scheduling, respectively.

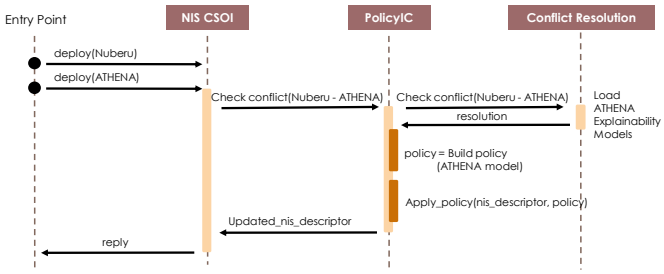

Fig. 3. The conflict resolution procedure for the chosen example.

shifts, inference glitches, or slightly misaligned loss functions during training. In such situations, the NIO plays a crucial role in configuring policies to address conflicts and mitigate potential issues before the system aligns itself with the overall service goal or requires model amendments or re-training.

III. Enforcing NI-native procedures in NI Algorithms

In the subsequent, we discuss how different NI-Native procedures are integrated into the overall DAEMON architecture.

A. Conflict resolution in ATHENA

As defined in [3], the NIO provides a conflict resolution procedure for algorithms that act on the same sink, according to the N-MAPE-K [1] definition. We exemplify this for the Reliable Distributed Unit (DU) solution (Nuberu [10]), and the Radio Resource Scheduling algorithm (ATHENA [4]), which we already used in [2] to exemplify our architectural framework.

Both Nuberu and ATHENA utilize similar data sources, necessitating coordination (see Fig. 2). However, their primary interaction occurs at the MAC scheduler in the DU of 5G base stations, which serves as a common Sink. On the one hand, Nuberu performs two main functions: (i) It predicts the probability of successful decoding of uplink frames, leveraging information from the L1 layer to mark frames as decoded or undecoded before the actual decoding process takes place. (ii) It modulates the number of uplink grants available to the Media Access Control Layer (MAC) scheduler to compensate for decoding operations that may not be accomplished in time due to fluctuations in computing capacity. On the other hand, ATHENA determines the number of Resource Blocks (RB) and the Modulation and Coding Scheme (MCS) to be used at a given point in time, ensuring that the time budget allocated for decoding operations is not exceeded. These two algorithms complement each other well as they both aim to achieve the same goal, with each emphasizing the strengths of the other.

However, conflicts may arise between them in certain situations. For example, Nuberu may reduce decoder capacity due to a previous failure to meet the time deadline, while ATHENA assumes full computing capacity within the system. Since the capacity reduction information is only available after the congestion control algorithm of Nuberu, and not during the scheduling phase, ATHENA cannot anticipate this conflict.

Nevertheless, by leveraging the explainability capabilities of the ML Radio Scheduling algorithm, as discussed in Section III-B and shown in Figure 4,we can observe that, while the RL agent in the dAPP makes decisions that push the boundaries of the reward distribution, such as maximizing overall throughput, there are areas where simpler and "failsafe" decisions can be considered. These decisions become relevant when Nuberu needs to scale down and alleviate congestion at the L1 decoder.

Two possible decisions can be made to reduce the scheduled Transport Block (TB) in the uplink: reducing the number of RBs or reducing the applied MCS. By utilizing the information available in the explainability module of the scheduler, the conflict resolution policy can regenerate the grants based on this new policy until the congestion is resolved, enabling both algorithms to resume normal operation. The conflict resolution policy is formalized as follows and depicted in Figure 3.

1) The ML scheduling algorithm continuously provides explainability measurements to the NIO for periodic policy updates in case of conflicts (i.e., two disjoint MAC scheduling decisions).
2) During each policy update, the conflict policy is injected back into the K module of the ML algorithm, which is responsible for adjusting the initially made decisions.
3) When congestion control is detected, Nuberu computes the maximum TB size that can be scheduled for the next Transmission Time Interval (TTI), as described in [10].
4) This information is communicated to ATHENA, which revises the previously made decision that assumed full capacity, considering the current congestion level.
5) The conflicting policy is used to determine the combination of MCS and RBs that maximizes the reward while adhering to the maximum available TB size constraint.
6) Grants are reformulated based on these constraints.
7) The revised grants are scheduled.

During the conflict resolution operation, no explainability information is reported to the NIO, only information regarding the utilization of the conflict resolution scheme. According to this scheme, both algorithms can operate without issues, reaching their optimal point of operation. However, if congestion persists for an extended period, resource re-orchestration is required, which will be triggered by the NIO using the NIO-MANO (Management and Orchestration) interface.

B. Explainability in ATHENA

One of the blocks of ATHENA [4] is the Machine Reasoning (MR) part, which interprets the ML results and devises (at a slower pace) actionable decisions on the network. The model provides insights into its internal functionality and decision process, which we call explanations. Explanations are categorized into three different classes depending on the question they seek to answer: (i) attributive, answering why, given an input, the model gave a certain outcome, (ii) contrastive, answering why the AI system's outcome is not different, and (iii) actionable, answering how the model's input should be modified so that a certain user-defined desired outcome is achieved. In our study, we investigated methods for all three categories and we adjusted them accordingly to our NN-based actor-critic architecture. However, in this paper, due to space limitations, we will overview only the attributive explanations.

We provide insights touching both the input, e.g., features of a particular example, and the model internals (splitting rules in the case of a decision tree, neurons in the case of Neural Networks (NN), etc.). We provide a comprehensive correlation between the input and the output, which does make sense for an external observer, mainly comprising model-agnostic explanators as feature importance methods. We also use model distillation to produce explanations. Its key concept is to distill a black-box model (e.g. a NN) into an inherently better explainable surrogate model, typically comprising swallow decision trees and linear models, whose parameters (splitting criteria and variable coefficients, respectively) are generally admitted to be better understandable. The distillation process consists of two models, namely the teacher T and the student S. T is the black-box trained model for which the explanations are asked. S is a smaller surrogate model that integrates the knowledge of T while maintaining its accuracy.

ATHENA's model consists of two neural networks: the actor's μ_θ, which provides an approximate solution $\hat{\alpha}$, and critic's Q_ϕ, which assists in refining it to α. We distill the actor's model to a regression tree, which is a decision tree that holds linear models in its leaves (instead of constant approximators). We used response-based knowledge distillation, where S mimics only the T's output layer. During the training process, T (actor) and S produce outputs $\hat{\alpha}_T, \hat{\alpha}_S \in R^2$ respectively. In order to train the tree S, we pass these outputs through the interpolation function $h : R^2 \rightarrow R$. This interpolation function is created using TBS as the target value of combinations (n, m). The outputs $h(\hat{\alpha}_T)$ and $h(\hat{\alpha}_S)$ are used to compute the distillation loss, which in our case is the 1-distance, capturing the difference in yielded throughput between T and S. In fact, we use the teacher's predictions as target values and we train the decision tree S in a supervised way, using the distillation loss as split criteria. We trained with maximum tree depth 3, in order to make it easier to interpret.

In Figure 4, we depict the agent's achieved throughput for combinations of the contextual features. With red dashed-edged rectangles, we paint the borders of the leaves of the trained decision tree. We interpret the splitting thresholds to understand the internal decision process of the actor. The tree has divided the input space into 4 regions, in which the NN-based actor can be approximated by a linear model. These areas have a specific meaning in the context of a vRAN system. For congestion factors $\beta > 0.5$ (Region 4), the actor yields the same throughput independently of the channel conditions. That is, the Central Processing Unit (CPU) is so congested that ATHENA is forced to scale down consistently both the MCS and PRB for all channel conditions. For lower β, instead, the SNR is taken into account, and for values higher than 20 dB, the agent is divided into 2 different models so that the maximum achievable throughput is better approximated.

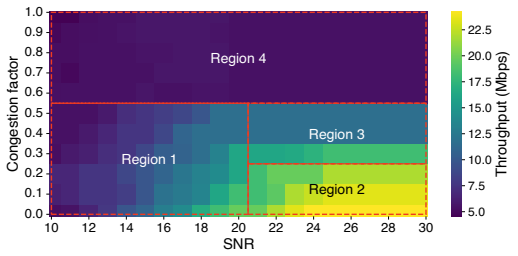

Fig. 4. ATHENA throughput and the regions of the distilled decision tree [4].

C. Conflict resolutions for NIF Re-Orchestration

In this section, we showcase an NIS example consisting of two NIFs, aiming at the creation of optimal deployments of vertical services (i.e., services tailored to specific use cases). The first NIF embodies a federated learning-powered anomaly detection algorithm, enabling anomaly detection at the edge. The second NIF is in charge of service relocation from one edge network unit to another. Both NIFs are designed to address connected mobility scenarios, where users, i.e., vehicles, connect to the optimal service deployments at the network edge to retrieve relevant information about their maneuvering process. Figure 5 illustrates the overall NIS deployment, where both NIFs' components are deployed at the edge or the cloud.

In particular, NIF1 enables anomaly detection by employing the AI model for detecting anomalies. This model is trained locally on individual devices (i.e., edge computing nodes), but at the same time, a central controller that is responsible for retaining a global model and therefore enhancing the local AI model is deployed in the cloud. This approach ensures that data privacy is preserved while the performance of NIF1 is still improved by enabling a collaborative learning process.

To perform service relocation, the NIF2 employs a Multi-Criteria Decision-Making (MCDM) algorithm that collects real-time monitoring data from the edge, resulting in a single decision, i.e., one node for service deployment or destination (in case of relocation). The collected data consists of CPU load, memory load, used storage, and end-to-end latency values. The latency is measured at the client side, reflecting on the performance that is perceived by the end user. The other collected metrics refer to the edge cloud performance and suitability of a specific node for the service deployment. Thus,

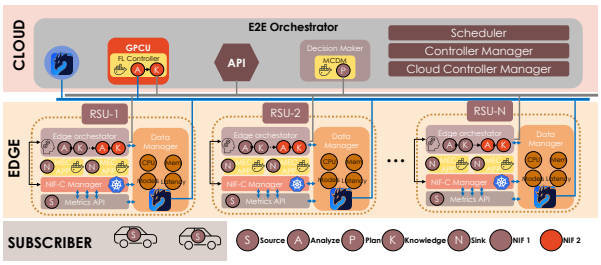

Fig. 5. Federated learning-powered anomaly detection/service relocation NIS.

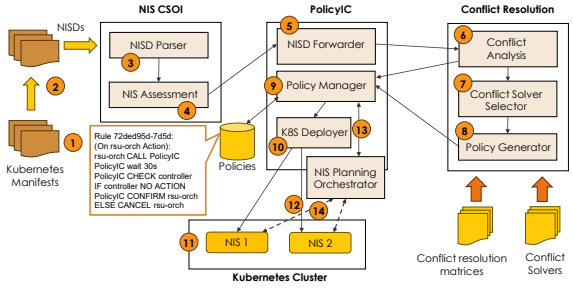

Fig. 6. Conflict resolution procedure.

using the aforementioned MCDM function and data collected in real time, NIF2 dynamically relocates services to ensure both optimal resource utilization and service performance.

Concerning the NIFs-empowered service relocation process, each use case's logic allows for some flexibility in terms of the process implementation. In this case, we define it as a process of relocating stateless services, deploying the correct service instance on the selected edge and terminating all previous instances running on the other edges to improve the edge resource utilization. NIF2's role is to make an optimal decision on the selected edge, taking into account the criteria explained above. Depending on the implementation, the decision can be taken up by the Kubernetes master (e.g., Kubernetes scheduler) to perform the actual service deployment.

This NIS is defined on top of the N-MAPE-K framework, thereby integrating federated learning-powered anomaly detection with service relocation. Given the federated learning capabilities, this NIS ensures optimal and secure anomaly detection across all network edges. By leveraging the methods for collecting monitoring data from the edges, this NIS can make informed decisions about service relocation, maximizing the network's overall performance and responsiveness.

We stress that this NIS is implemented and deployed on the Smart Highway testbed, a real-life environment for testing vehicular services with edge computing capabilities, located in Belgium. It consists of eight Roadside Units (RSUs) that serve as the edge environment for testing and validating vehicular services in 5G and beyond ecosystems. In our case, it serves for validating the effectiveness of the proposed NIS, as it creates a real-world scenario to assess its performance and potential benefits for vehicular services. Figure 5 illustrates the deployment of NIS on the Smart Highway testbed, stretching over multiple RSUs deployed on a 4km long highway segment.

The communication among the NIS components, i.e., NIFs, is implemented using the Eclipse Zenoh data communication framework, which provides a reliable and scalable solution for exchanging data between devices and components within the network. Figure 5 also illustrates this mode of communication, where Zenoh-based data managers are used for collecting metrics related to edge performance (CPU load, memory load, storage), and service performance (end-to-end latency). Observe that the edge performance metrics are used in the anomaly detection (NIF1) as well, while NIF2 also accounts for the latency values to get awareness about the service performance at the user side (vehicle). Finally, taking into account the response from both NIFs, the NIS makes the final decision on the service relocation, i.e., on which network service instance the test vehicle should connect.

The deployment of this NIS results in a service relocation that is sensitive to edge considerations. It strategically determines the optimal timing and location for relocating the service to maintain minimal end-to-end latency consistently. Depending on the type of vertical service (i.e., use case), the set of relevant metrics might change as the priority might be given to another performance optimization target than latency (e.g., uplink throughput for video streaming or lidar data collection from vehicles). Nevertheless, the principles of designing and orchestrating such NISs remain. This decision-making process not only considers edge performance but also evaluates stability by accounting for potential anomalies. Such intelligent services lay the foundation for high-performance deployments of vertical services in upcoming 6G ecosystems.

IV. Implementation and Experimental Evaluation

Section III-C process the prototyping of a set of basic NI Orchestrator functionalities, including NIS/NIF deployment, cooperation, and sharing realized in a Kubernetes environment. This section presents the prototyping of one of the most innovative functionalities provided by the NI Orchestrator: Conflict Resolution. The identification and resolution of conflicts among the deployed NISs is of paramount importance because it ensures stability and smooth operation of the NIP. During the prototyping, the Kubernetes environment was used as the main deployment infrastructure. The following phases of the conflict resolution process were addressed: a) NISD creation and annotation with the DAEMON N-MAPE-K taxonomy; b) initial assessment based on N-MAPE-K types; c) conflict identification, and; d) conflict resolution. We illustrate the process in Figure 6 and explained in this section.

To demonstrate the conflict resolution capabilities of the NIP, two NIS were created and deployed. The first NIS includes a federated learning-powered anomaly detection algorithm (as NIF1 of Section III-C), extended with a basic service resource management capability (e.g. scale in/out on resource under/overutilization). The second NIS employs a service relocation algorithm (as NIF2 of Section III-C). The configuration actions of the above NISs (scale in/out and service relocation) are conflicting because they drive the network