RAN Cognitive Controller

Anubhab Banerjee\textsuperscript{1,2}, Stephen S. Mwanje\textsuperscript{1}, and Georg Carle \textsuperscript{2}

\textsuperscript{1}Nokia Bell Labs, Munich, Germany
\textsuperscript{2}Dept. of Informatics, Technical University of Munich, Germany
Email:anubhab.banerjee@tum.de, stephen.mwanje@nokia-bell-labs.com, carle@net.in.tum.de

Abstract—Cognitive Autonomous Networks (CAN) deploys learning based Cognitive Functions (CF) instead of conventional rule-based SON Functions (SF) as Network Automation Functions (NAF) to increase the system autonomy. These CFs work in parallel sharing the same resources which give rise to conflicts among them which cannot be resolved using conventional rule based approach. Our main target is to design a Controller which can resolve any type of conflicts among the CFs in a dynamic way.

Index Terms—Network management and automation

I. PROBLEM DESCRIPTION

A typical network is characterized by many control parameters and Key Performance Indicators (KPI). For example, a gNB has several control parameters like Transmission Power (TXP), Antenna Tilt (RET) or Antenna Gain, and, multiple observable Key Performance Indicators (KPIs) like downlink throughput, Radio Link Failures (RLF). In CAN, each CF focuses on optimizing one or more KPIs as its objective. The CF continuously observes and learns the variation of its objective w.r.t. changes in the control parameters or the external environments, and based on learning, it determines the values of control parameters for which its objective is optimal in a certain state.

In 5G environment can change very rapidly, and the CFs may detect frequent changes in their respective optimal control parameter values. If each CF changes the parameters according to its own will, stability of the system might be compromised. This is why we assume, for sake of stability issue, that only the Controller changes the control parameters. Without going into details, we can abstract a CAN as shown in Fig. 1 in a hierarchical overview where Controller stays one levels above the CFs and makes necessary changes in the control parameters \cite{1}.

When a control parameter is shared among multiple CFs, and when each CF wants to change the value of the parameter to different degrees of extent, conflicts among them arise. CFs exhibit three types of conflicts among them - (i) Configuration conflict (while sharing a control parameter or KPI simultaneously), (ii) Measurement conflict (while influencing other CF’s output by own action) and (iii) Characteristic conflict (logical dependency) \cite{2}. As these conflicts cannot be resolved using conventional rule based approach, some new approach is required. Our goal is to design a Controller which can resolve any type of conflicts among these learning agents and determine the optimal configuration.

II. PROPOSED SOLUTION

We develop a Controller with the following properties -

\begin{itemize}
  \item It can be used to determine the optimal values for all configurations in CAN.
  \item It calculates configuration of the system in a way which optimal for the combined interest of all the CFs.
  \item Its calculation mechanism is very generic and dynamic to be used in real life.
\end{itemize}

In this Section we give a brief overview on how the CFs with the Controller work in CAN and how they exchange necessary information.

A CF always sends two pieces of information to the Controller whenever required - (i) optimal-config-range set and (ii) utility function. As each CF is a learning agent itself, it can generate its most favorable configuration set based on
its learning. This set is called optimal-config-range set with the structure: \([\text{min\_value}, \text{max\_value}]\). For any parameter value which lies in between \text{min\_value} and \text{max\_value}, the objective of the CF always lies within a certain percentage of its maximum objective value. Also, In CAN, different CFs have different objectives with different dimensions. For the Controller to understand and compare between different CFs’ outputs, it is recommended to convert the outputs in some identical predefined scale. Example of such a scale is \([0:10]\), where 0 means the lowest and 10 means the highest achievable value. To convert its output to this scale, each CF generates a function called utility function.

A CF is trained with some real life or simulator generated dataset before it becomes operational so that it knows its optimal-config-range set and utility function beforehand. The system starts with some preloaded configuration determined by the Mobile Network Operator (MNO) based on previous experience. After the system becomes operational, each CF starts observing its output and begins learning from it. After every certain time interval, the CF calculates the optimal-config-range set for a configuration and compares it with the optimal-config-range set calculated in the last cycle. If these two sets are identical, the CF continues its learning until the next cycle. Otherwise, if there is a change between these two sets, it sends a request to the Controller to recalculate that configuration. After the Controller receives a request for configuration recalculation from a CF, it sends a message to all CFs in the system, asking for the necessary information. After the Controller receives all the information, it calculates the optimal configuration as described in the next paragraph and makes necessary changes in the network. This whole end-to-end workflow is depicted in Fig. 2.

The optimal value is calculated using Nash Social Welfare Function (NSWF) \cite{3} which is defined as the product of the individual agent utilities for a particular resource allocation. The reason behind using NSWF in a resource allocation problem is the solution provided by NSWF balances efficiency and fairness, and it is sensitive to change in overall welfare \cite{4}. The fairness in allocation can be observed from the lowest difference in utilities obtained.

III. IMPLEMENTATION AND CONCLUSION

In this section we discuss how the optimal configurations are calculated in a real life scenario based on a system level simulator called SEASON II which has already been used extensively in previous research works like \cite{5}. The environment of the simulations is an authentic recreation of a small part of the city of Hamburg with realistic radio propagation models (WINNER+). In an area of 4 sq. kilometer we deploy 5 cells. We place 100 static users across all cells randomly for 20 times and collects data.

We assume a CAN deployed at the central cell with two CFs - Mobile Load Balancing (MLB) and Capacity and Coverage Optimization (CCO). MLB tries to reduce the load in the cell and CCO tries to maximize the coverage and capacity of the cell. Both of them use TXP as their one of the input parameters and have a conflict over it, so in this paper we show how the conflict is resolved and optimal TXP is determined. We use two neural networks (NN) to implement these two CFs. Each CF has 5 fully connected layers, and each layer has 50 nodes with MSE as the loss function and Adaptive learning rate optimizer. We collect data from 20 different scenarios and use them for training the NNs, so that after the training is done, each CF can predict the output corresponding to a set of input configurations. For each CF we use a linear utility scale - for an output value \(v\), the corresponding utility value is \(\frac{v}{\text{max}v}\), where \text{max}v is the maximum obtained output.

From Fig. 3 we see that utilities of MLB and CCO are maximum for different TXP values. However, when TXP = 66 dBm, their respective utility values are 0.998 and 0.986, and the product of their utilities is maximum. So, according to NSWF, this value is the optimal TXP for the combined interest of both MLB and CCO i.e., none of them have their maximum utility at this TXP value, but it is a good balance for two of them. However, problem with this approach is it always assumes the shared parameter has equal impact on both the CFs, which may not be always the case in reality. As a future direction from here, we plan to incorporate the impact of each parameter on CF so that the optimal solution can be determined more precisely.

REFERENCES

\[1\] A. Banerjee, S. S. Mwanje, and G. Carle. Game theoretic conflict resolution mechanism in cognitive autonomous networks. In 2020 International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS), pages 1–8. IEEE, 2020.

\[2\] A. Banerjee, S. S. Mwanje, and G. Carle. On the necessity and design of coordination mechanism for cognitive autonomous networks. arXiv preprint arXiv:2001.07031, 2020.

\[3\] J. F. Nash Jr. The bargaining problem. Econometrica: Journal of the econometric society, pages 155–162, 1950.

\[4\] A. Banerjee and C. M. Machuca. Game theoretic pricing policy in online media delivery platforms. arXiv:1912.06434, 2019.

\[5\] S. S. Mwanje, M. Kajó, S. Majumdar, and G. Carle. Environment modeling and abstraction of network states for cognitive functions. In NOMS 2020–2020 IEEE/IFIP Network Operations and Management Symposium, pages 1–8. IEEE, 2020.