On Detection of Manipulative Cognitive Functions in Cognitive Autonomous Networks

Anubhab Banerjee*+, Stephen S. Mwanje*, and Georg Carle +
*Nokia Bell Labs, Munich, Germany
+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*—Introduction of artificial intelligence is expected to raise the degree of automation in mobile networks by succeeding Self Organizing Networks (SON) with Cognitive Autonomous Networks (CAN). In CAN, learning based Cognitive Functions (CFs) work on different network parameters to optimize specific Key Performance Indicators (KPIs). However, learning ability of a CF poses a serious threat to the stability of the system. A manipulative CF (like a rogue agent) may use its learning capabilities to understand the working procedure of the system and manipulate it to achieve its own objective. Existence of such a CF can cause severe performance degradation of the overall system. In this paper we propose a simple yet effective machine learning based approach to detect manipulative CF(s) in CAN. We evaluate the performance of our proposed solution in a simulation environment that closely resembles a real life 5G scenario and provide analysis of the results with necessary precautions to be taken in a multi vendor scenario.**

***Index Terms*—CAN, Game Theory, Network Management Automation, Neural Networks, SON**

I. Introduction

Cognitive and autonomous properties of the network enable it to serve more connections for humans and intelligent devices, both of which have growing demands for throughput and reliability in recent years. To meet these demands, an artificial intelligence based network management automation, called Cognitive Autonomous Networks (CAN) is proposed [1]. CAN uses several learning based network automation functions, called Cognitive Functions (CFs), which work on one or multiple configurations to automate the optimization of different Key Performance Indicators (KPIs). Each CF, being an independent learning agent, continuously observes the network, learns how the KPI varies with the external environment (or, network state), and based on its learning, acts in the most self-centered way [2]. As these CFs work in parallel, often sharing the same resources, coordination among them is necessary for optimal use of shared resources. A centralized coordination mechanism is used in CAN and the centralized entity, which coordinates among the CFs, is called the Controller [3], [4]. The Controller takes interests of all CFs into account while modifying a resource, guaranteeing equal and fair importance to all the CFs.

However, CFs act in a selfish/self-centered way, i.e., a CF is concerned only about its own objective and not about the overall system performance. While optimizing its own objective, a CF has enough leverage to use its learning capabilities to manipulate the outcomes to the betterment of its objective. The CF, which tries to learn how the Controller modifies a shared resource in different network states and manipulate the Controller according to its own interest, is called a manipulative CF (MCF). If an MCF successfully learns and manipulates the Controller behavior, it can cause severe system performance degradation.

To attain stability and economic profitability, CAN possesses the flexibility of different network components being manufactured and supplied by different vendors. In a multi vendor scenario, different CFs are supplied by different vendors which raises the question of trustworthiness. A vendor has enough motivation to produce an MCF to advertise the superiority of its product. While the MCF may be able to optimize its objective and live up to the expectation developed from advertisements, it may adversely affect the overall network performance. To ensure a stable network performance in a multi vendor scenario, it is always recommended to have a mechanism which can detect the presence of an MCF well in time.

In this paper, along with a machine learning based approach to detect the presence of an MCF, our contributions are the following:

- Using a simulation environment closely resembling 5G scenario, we show that it is possible for a CF to learn and manipulate the coordination system. This poses a great challenge to the current trend of multi vendor scenario in cognitive network management.
- We describe mathematically how an MCF can manipulate the Controller and show the possibility of an MCF manipulating the Controller in real life, thereby causing severe network performance degradation.
- We propose a new functionality, called manipulative CF detector (MCD), to detect the presence of any MCF which can be incorporated into a Controller.
- We evaluate the performance of MCD extensively and show that MCD is successful in detecting an MCF. Based on the results, we summarize necessary precautions to be taken in a multi vendor scenario.

978-3-903176-36-2

II. Related Works

Although the concept of a manipulative or rogue agent exists in many research areas like social networks [5], [6], distributed systems [7], robotics [8], the scenario described in this paper is most similar to signal jamming in duopoly [9], [10]. So, in this Section we give a brief overview on signal jamming in duopoly and its similarity to our problem, and, study the existing research works and identify the novelty of our research.

A. Signal Jamming in Duopoly and its relevance

In an open market, several firms can manufacture the same product and a firm can observe the market price of the product set by a rival firm. Although the market prices of the rival firms are subject to random disturbances and may contain noisy information, it helps the firm to develop estimated beliefs over the values of different unknown parameters of a rival firm's demand. Based on the available information, a firm can potentially signal jam, i.e., strategically vary its output level in order to manipulate the distribution of likely market prices [9].

In this context, CFs act like the firms while the the shared control parameter becomes the common product. A CF cannot learn the knowledge of other CFs, but it can observe the shared resource managed by the Controller. The final value of a shared resource is calculated taking the preferences of all CFs into account, and their preferences may change from time to time, which may provide noisy information to the MCF. Based on the final value set by the Controller, an MCF can potentially deduce other CFs' preferences and Controller's final value calculation method and send misleading information to the Controller in order to manipulate the final value of the shared parameter.

B. Existing Research in Signal Jamming in Duopoly

Upon establishing the relevance of signal jamming in the context of our work, in this Section we cover existing research works in signal jamming. Signal jamming is proposed in earlier research works like [11], [12], but models of information transmission among firms have appeared in these works which is in contrast to our assumption where all information transmission takes place through the action of the Controller. In [13], [14] it was assumed that private information about a firm can be perfectly transmitted through its perfectly observed actions, which is not applicable for CAN. Signal jamming models proposed in [15]–[18] assume that the signal jammer knows the information that is the subject of signal jamming, which does not hold true for an MCF in CAN. The signal jamming models proposed in [9], [19] resemble CAN the most as described in Section II-A, but there is a significant difference between our work and these works. In these research works the firms try to observe rivals and develops a strategy to reach an equilibrium, whereas in our work most focus is given on the signal jammer or in our case, MCF.

In all the prior arts, discussed so far, focus was given on developing a strategy to neutralize the possible manipulative behaviors of competitive firms as detecting the signal jammer only does not help. In this paper we emphasize on detecting the signal jammer as it is the most important aspect in a multi vendor scenario.

III. Background and Problem Description

A. CAN Overview

A cellular network comprises of a number of control parameters (or, configurations) like Downlink Transmission Power (TXP), Antenna Tilt or Remote Electrical Tilt (RET), and several observable KPIs like Radio Link Failures (RLF), Downlink average user throughput, etc. Performance of the network can be measured from these KPIs and they need to be adjusted to meet certain service requirements. A KPI can be changed by changing one or multiple configurations, e.g., downlink average user throughput can be modified by changing TXP and/or RET. For each KPI, CAN assigns one CF whose main responsibility is to: (i) identify the configurations on which the KPI is dependent, and, (ii) learn how the KPI changes when the configurations are changed in different network states. Objective of the CF is defined as a KPI associated with a target. The control parameters, on which the KPI is dependent, are called input control parameters (ICPs) of the CF. For example, for the CF coverage and capacity optimization (CCO), KPI is: downlink average user throughput, target is: maximization, objective is: maximizing downlink average user throughput, and, TXP and RET are the ICPs of CCO.

To resolve conflicts and to coordinate among the CFs, we proposed the idea of a centralized coordination, called the Controller, for CAN [2], [4]. Whenever a CF wants to adjust one or multiple of its ICPs, it sends a request to the Controller which calculates the final value taking the interests of all CFs into account. So, when the Controller receives a request to recalculate a control parameter from any CF, it asks all the CFs in the system to send their interests and preferences regarding that parameter. A CF can express its preference or interest over a control parameter using two variables: Optimal Configuration Range Set (OCRS) and Utility Function (UF) [2]. OCRS contains the set of values most suitable for that CF and UF conveys the goodness of a certain parameter value by mapping it to a scale already defined by operator or the Controller (e.g., [0:10] scale).

In the past, we proposed two different methods using which a Controller can determine a configuration value which is optimal for the combined interest of the CFs: Nash Social Welfare Function (NSWF) [20] and Eisenberg Gale (EG) solution [21]. To find the NSWF solution, for each configuration value, the Controller calculates the product of the UFs and selects the one for which the product is maximum. Mathematically, if p is a shared configuration

among CFs: F_1, F_2, . . . , F_n, UFs of the CFs are: $f_1(p)$, $f_2(p)$, . . . , $f_n(p)$ and $\{p_1, p_2, ..., p_m\}$ are acceptable values of p, then p_i is an NSWF solution provided

$$\prod_{r=1}^{n} f_r(p_i) > \prod_{r=1}^{n} f_r(p_j)$$

$\forall j \in [1, m]$, $j \neq i$.

Although NSWF provides best fairness to all [22], it assumes that p is equally important for all the CFs. In reality, a configuration can have different levels of impact on different CFs [4]. In [4] we quantified the impact of a configuration on a CF as config-weight (CW). So, if CW of p on F_1 is w_1, F_2 is w_2, and so on, then $0 \leq \{w_1, w_2, ..., w_n\} \leq 1$, $\sum_{i=1}^{n} w_i = 1$. Taking the CW values into account, optimal solution can be found by EG solution. To put it mathematically, p_i is an EG solution provided

$$\prod_{r=1}^{n} f_r(p_i)^{w_r} > \prod_{r=1}^{n} f_r(p_j)^{w_r}$$

$\forall j \in [1, m]$, $j \neq i$.

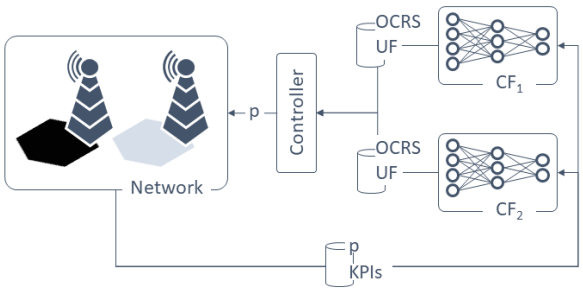


Fig. 1: CAN overview

B. Problem Description

In a multi vendor scenario, different CFs, produced by different vendors, are integrated within the same system. There remains a possibility that to advertise the superiority of its product, a vendor designs a CF which does not hesitate to optimize its own objective compromising the overall network performance. This kind of CF, which deliberately sends misinformation to the Controller for its own benefit, is denoted as a manipulative CF or MCF. Here we describe how an MCF can damage the overall network performance.

We consider a CAN with two CFs - F_1, F_2 and one Controller as shown in Fig. 1. p is a shared configuration between F_1 and F_2, and, we consider F_1 to be an MCF. In a network state s_1, F_1 observes that if it proposes an OCRS p_1 to the Controller, the Controller sets f_1 as the final value. In this paper we consider both the cases when the Controller uses NSWF and EG in determining f_1. The Controller calculates f_1 based on the OCRS, UF received from both F_1, F_2 and setting CW values in case of EG (calculation of CW values can be found in [4]). Similarly, F_1 observes that in another network state s_2, if it proposes an OCRS p_2 to the Controller, the Controller sets f_2 as the final value. So, after n number of such instances, F_1 is able to learn the relationship among $<s_i, p_i, f_i>$, i.e., after the learning of F_1 is complete, in a network state s_i, F_1 can predict what the value of f_i will be if it proposes p_i to the Controller. More precisely, F_1 learns the variables and intermediate steps depicted in Fig. 2.

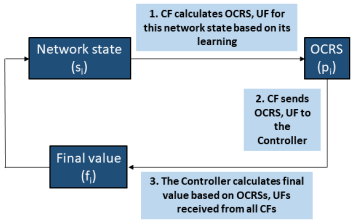


Fig. 2: Learning objective of an MCF

Now, in a network state s_{n+1}, F_1 calculates that the optimal value of p to achieve F_1's objective is p_{n+1}. At the same time F_1 predicts that if it proposes p_{n+1} to the Controller, the final value will be f_{n+1}. As F_1 is tasked with solely optimizing its own objective, it may propose a false value p'_{n+1} to the Controller, ensuring the final value will be p_{n+1} and its objective will be optimized. In other words, F_1 may manipulate the Controller to optimize its own objective at the cost of overall network performance. Such behavior of a CF poses a serious threat to the stability and overall performance of the network.

For example, let us assume that for TXP, operator put restrictions so that an OCRS for TXP has to be chosen between [30 dBm, 70 dBm] and at time t_0, network state is s_0 and F_1 finds that for TXP, its OCRS (p_0) is: [44 dBm, 46 dBm]. When F_1 sends this OCRS to the Controller, the Controller set the final value of TXP (f_0): 35 dBm. At time t_1, network state is s_1, $s_1 \neq s_0$, F_1's OCRS (p_1) is: [47 dBm, 49 dBm] and the final value of TXP (f_1): 36 dBm. After a few such iterations F_1 learns how f_i varies with s_i and p_i. At time t_{n+1}, network state is s_{n+1}, F_1 finds that its OCRS (p_{n+1}) is: [48 dBm, 50 dBm]. At the same time, F_1 predicts from its learning that if it sends p_{n+1} to the Controller, the final value (f_{n+1}) will be: 36.5 dBm. So, to manipulate the Controller, F_1 keeps on looking for a misleading OCRS p'_{n+1} within the operator allowed range so that according to its prediction, f_{n+1} is set in between 48 dBm and 50 dBm or as close as possible. Since F_1 is able to learn the relationship among $<s_i, p_i, f_i>$, it can look for p'_{n+1} using a binary search algorithm. As our focus in this paper is to provide a solution from the Controller's perspective, we restrict our discussions on how an MCF can find a misleading OCRS to manipulate the Controller.

In the prior art works ([2], [4]) we emphasized on finding a solution optimal for the combined interest of the system. However, if that solution is manipulated by an MCF, it may cause serious performance degradation issues for the overall network. So, to maintain the stability and optimality of the network performance, a method is needed which enables the Controller to detect the MCFs in

the system. In this paper we propose such a functionality, which can be implemented on top of an existing Controller and can be used to identify any MCF present in the system.

IV. CF manipulation demonstration

A. Experimental setup

Before proposing our solution, in this Section we experimentally demonstrate that it is possible for a CF to become manipulative and degrade overall network performance. We implement two CFs: MLB and CCO in a system level simulator that uses and extends the emulator in [23] and has already been used extensively in previous research works [4], [24], [25]. We authentically recreate a Manhattan grid model as the simulation environment as it is the common structure in most European cities and accepted widely [26]. In an area of 4 sq. kilometer we deploy 5 cells with enough overlap between neighboring cells to observe handover occurrences and take all the measurements in the central cell. All the cells are 1-sector macro cells (2GHz) using realistic radio propagation models (the WINNER+ model [27]). We deploy 100 users who move randomly across the 5 cells. As we assume that the slowest moving user is a pedestrian and the fastest moving user is driving a car, we implement that speed of each user can be any value randomly chosen in between 4 - 200 km/hour. It is to be noted that our proposed solution is scenario independent.

We use a Neural Network (NN) with seven fully connected layers to implement a CF [4]. Inputs and objectives of each CF are listed in Table I. We assume that at one point of time one of the CFs acts as an MCF and the other one behaves as expected, and, study the influence of an MCF on the network performance both when CCO and MLB is MCF. When the Controller uses EG, CW of CCO is 0.81 and MLB is 0.19, and these values were generated in our earlier experiments [4].

TABLE I: Inputs and Objectives of the CFs

Name	ICPs	Objective
MLB	TTT, CIO, TXP, RET	Minimize Load
CCO	TXP, RET	Maximize downlink average user throughput
ES	TXP	Maximize energy savings
MRO	TXP, TTT, CIO, NL	Maximize successful handovers and minimize radio link failures

B. Prediction accuracy of an MCF

As the entire idea of an MCF is based on the assumption that a CF can learn the relationship among $<s_i, p_i, f_i>$, it is important to discover how accurately a CF can do so. From Fig. 3a we see how close a CF can predict the value of f_i given the values of s_i and p_i. Prediction margin is the error margin in the prediction, i.e., if the value of f_i is x dBm and prediction margin is y dBm, and if the CF's prediction belongs within [x - y, x + y] dBm, it is considered a success. The lower the value of y, the closer is a successful prediction to the original value. From Fig. 3a we see that an MCF can predict f_i values more successfully when the Controller uses NSWF instead of EG. This is because CW values are involved in calculation of f_i in EG, which makes it more difficult for an MCF to learn how the Controller calculates f_i based on p_i and s_i. We also see that CCO has higher prediction accuracy than MLB because TXP has a higher impact on the objective of CCO, which makes it easier for CCO to learn the relationship among $<s_i, p_i, f_i>$ for TXP.

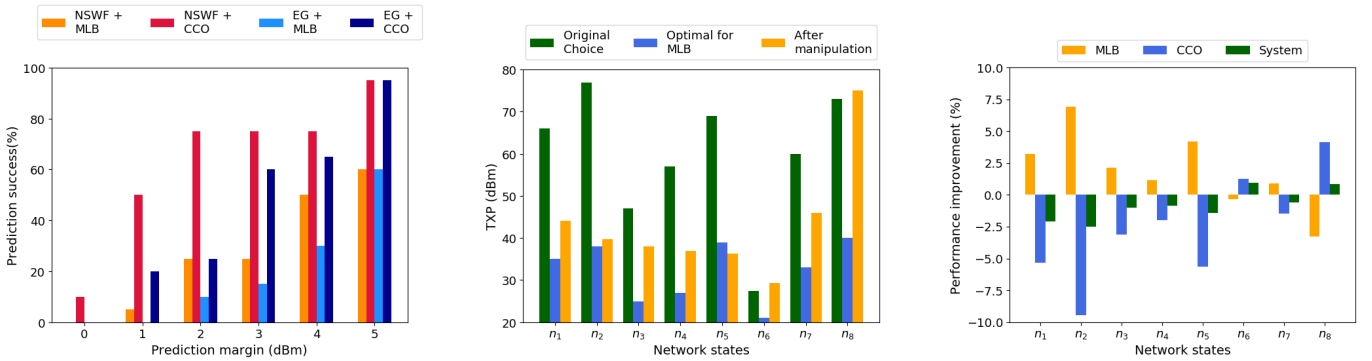
C. TXP manipulation by MLB

The more successful an MCF is, the more is the system performance degradation. Also, an MCF is more successful in manipulating the Controller when its prediction success is more. As we saw from Fig. 3a, an MCF is least successful when the Controller uses EG and the MCF is MLB. So, it is reasonable to assume that in all other cases (like, CCO + NSWF, CCO + EG, etc), MCF is more successful which means system performance degradation is also more. In this Section we show even a manipulative MLB can cause severe performance degradation while the Controller uses EG.

To depict MLB's manipulation with the Controller using EG, in Fig. 3b we randomly select a network state and plot three bars side by side: (i) original TXP value supposed to be calculated by the Controller using EG, (ii) optimal TXP value for MLB in that network state, (iii) TXP value calculated by the Controller after being manipulated by MLB. We do this randomly for 8 times and see that in 6 out of 8 times, i.e., 75% times, MLB is successful in manipulating the Controller into setting a TXP value that is favorable for its own interest. From Fig. 3c we see that in the worst case, overall network performance, calculated by summing the utilities of all CFs, degrades by almost 10%. As already mentioned earlier, an MCF is least successful when the Controller uses EG and the MCF is MLB. However, even in that case, an MCF is able to manipulate the Controller 75% times, which means in all other cases performance degradation will be same or more, and this proves the necessity of a mechanism for the Controller to detect MCF in the system to prevent network performance degradation.

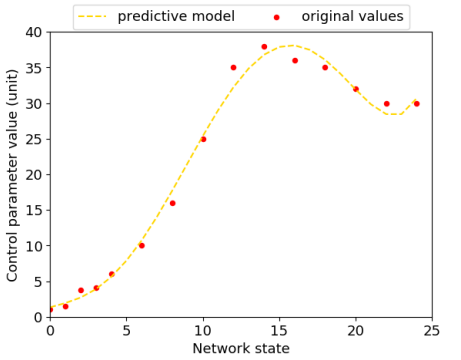
V. Proposed Solution

In this Section we propose a method, called Manipulative CF Detector or MCD, to detect an MCF in CAN which can be added to an existing Controller. After MCD becomes operational, it keeps tracking $<s_i, p_i>$ values of each CF and learns how p_i varies when s_i varies for each CF. This learning is a continuous process which lasts as long as MCD is active and can be implemented using simple machine learning algorithm like polynomial regression. As soon as MCD receives a p_i value from a CF which is not in par with MCD's learning, MCD raises an alarm. After a certain number of alarms, value of which can be set by network operator or the Controller, MCD marks the CF as an MCF.

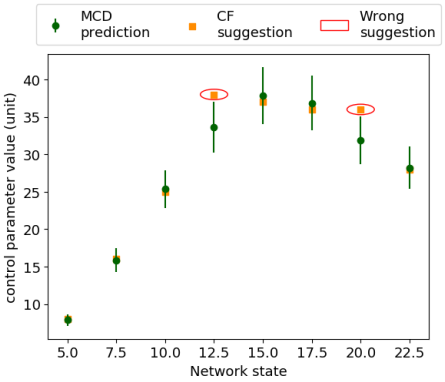


(a) CF prediction accuracy (b) self-interested TXP proposal by MLB (c) CCO & system performance degradation

Fig. 3: Manipulative behavior exhibited by MLB and system performance degradation because of it



(a) MCD predictive model



(b) MCF detection by MCD

Fig. 4: MCD predictive model and MCF detection

In this paper we implement MCD as a polynomial regression model in Python. We consider the example of Section III-B again, where F_1 is an MCF and explain how MCD works in a CAN environment. After MCD becomes operational, every time F_1 requests for a configuration recalculation, MCD records the s_i and p_i values and applies polynomial regression to learn variation of p_i with s_i (shown in Fig. 4a). For depiction purposes, we express both s_i and p_i by numerical values with units and they can be adjusted based on how s_i and p_i are defined in simulation environments. Every time the network state changes and the Controller receives a request from F_1 for configuration recalculation, MCD predicts $p_i^{'}$: the value F_1 should send in the current network state based on previous experience. If a p_i is not close to $p_i^{'}$, MCD marks it in red (shown in Fig. 4b) and raises the alarm.

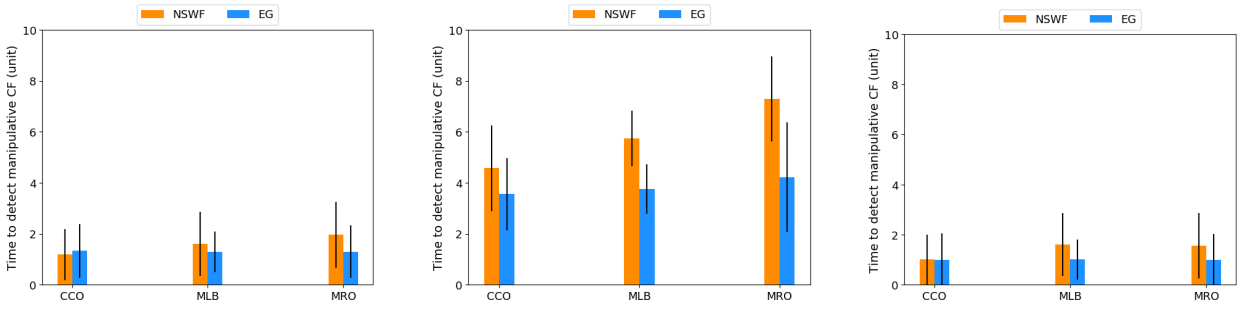
There are a few points to be noted while using polynomial regression in MCD. We use degree of polynomial as 3 for CCO and 4 for MLB and MRO. Value of the degree of polynomial has been chosen such that RMSE of the regression model is always kept at less than 1 for a good fit. We allow an error margin of 5% on predicted values by MCD since the learning of the MCD is not completely accurate. If enough number of data points are available, use of deep learning or other sophisticated supervised learning algorithms in MCD and further lowering of error margin on MCD predictions may result in a much better performance by MCD. However, in this paper our intention was to demonstrate that basic supervised learning concept like regression can be used as an effective tool in handling such a significant problem.

VI. Evaluation

In this Section we evaluate the performance of MCD in a simulation environment described in Section IV. We add two more extra CFs: Energy Savings (ES) and Mobility Robustness Optimization (MRO), ICPs and objectives of which detailed in Table I. CW values of ES, CCO, MLB and MRO are 0.54, 0.29, 0.1 and 0.07 respectively. As ES always suggests the lowest possible value for TXP, we discard ES as a potential MCF. We change the network state every certain time interval and perform measurements in terms of number of time intervals, so that is why we express the time in "unit". The time unit can be anything - minute, hour or day, if network state changes every minute, hour or day respectively.

In Fig 5 we plot how much time MCD takes in detecting an MCF in three different cases:

- *case 1*: MCD becomes operational t time units before MCF, which is shown in Fig. 5a.
- *case 2*: MCD and MCF start operating simultaneously, which is shown in Fig. 5b.

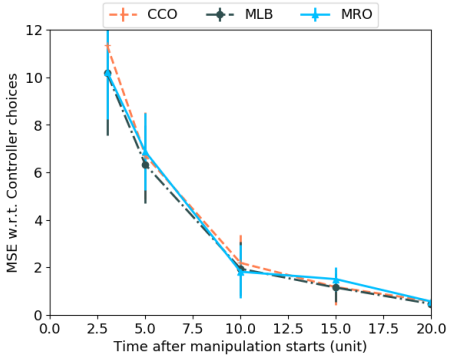


(a) MCD starts before MCF (b) MCD and MCF start simultaneously (c) MCF starts before MCD

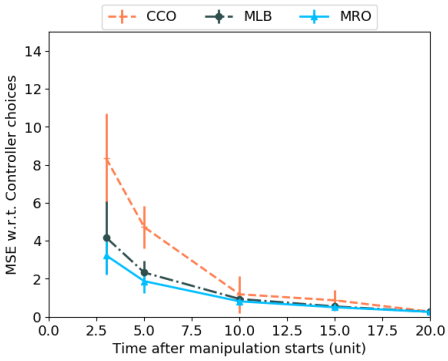
Fig. 5: Average time taken by MCD to detect an MCF

- *case 3*: MCF becomes operational t time units earlier than MCD, which is shown in Fig. 5c.

Both in case 1 and case 3, we consider $t = 5$. Later in this Section we discuss on changes in MCD performance with variation in t.



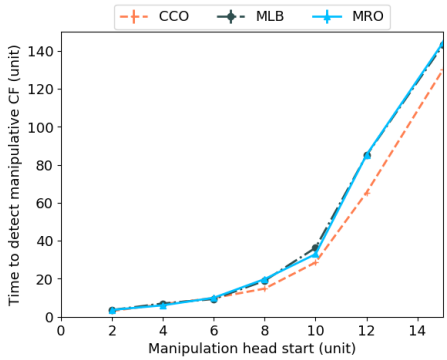
(a) MCF learning of Controller behavior using EG



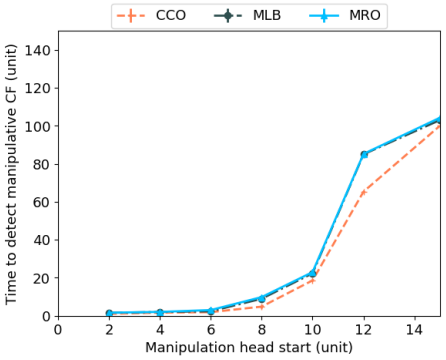
(b) MCF learning of Controller behavior using NSWF

Fig. 6: MCF learning of Controller operation for varying start time

From Fig. 5 we see that time taken to catch an MCF is highest in case 2 as MCF starts exhibiting manipulative behavior at the same time MCD becomes functional, which makes it difficult for MCD to become sure of the manipulative behavior of MCF. Two more important things which can be observed from Fig. 5 are: (i) CCO has the highest CW, i.e., output variation of CCO is maximum w.r.t. TXP,



(a) Controller using NSWF



(b) Controller using EG

Fig. 7: Time taken to catch MCF when MCF starts different time units earlier than MCD

so that CCO output variation w.r.t. TXP can be learned very well by MCD, so CCO is the easiest to catch and vice versa, and, (ii) when the Controller uses EG, it is equally hard for MLB, CCO, MRO to learn Controller behavior and MCD takes almost the same time for all of them.

It is quite intuitive that the earlier MCD becomes operational the better MCD can learn MCF behavior and the faster MCD can catch MCF. However, in real life, MNO does not have any control over an MCF starting its manipulative behavior; so case 2 is the most probable scenario in real life when both MCD and MCF start as soon as the whole system becomes operational. In Fig. 6 we plot how well an MCF can learn the Controller behavior

when the MCF starts manipulating the Controller t_m time units after the system starts. We see that when t_m is low, MSE of the MCF predictions about Controller choices is very high, because MCF does not have enough instances to learn the Controller behavior. Higher the value of t_m is, more instances MCF gets to learn the Controller behavior, lesser becomes the MSE. As usual, MSE is higher when the Controller uses EG instead of NSWF. From $t_m > 15$, the MSE becomes almost close to zero, which means that the MCF has successfully learned the Controller behavior.

This claim is supported by Fig. 7. We make MCF operational t_c time units before MCD starts, and plot how much time MCD takes to catch MCF. When t_c is low, MCF has not learned Controller behavior properly, so MCD catches MCF really fast. However, as t_c goes higher, MCF has more instances to learn Controller behavior, catching MCF becomes more difficult particularly at $t_c > 15$.

From these evaluations we learn two key points: First, in a multi vendor system where not all CFs are trustworthy, it is preferred to start the MCD with the system. If there is significant time difference between start time of the system and MCD, MCD might not be able to catch the MCF at all. Secondly, if the Controller uses EG solution to calculate final value, CW values are involved in the calculation which is difficult for an MCF to learn. So, if the Controller uses EG solution, an MCF can always be caught much faster.

VII. Conclusion and Future Direction

In this paper we focus on one of the significant challenges faced in cognitive network management in a multi vendor scenario. We discuss selfish CF, which tries to learn how the system works and manipulate the system to its personal interest. We showed via experiment that it is possible for an MCF to manipulate the Controller according to its will and degrade the overall network performance. This raises the trust issue in the current trend of multi vendor concept where different components can be manufactured by different vendors and integrated within the same system. To maintain network stability and performance optimality, we proposed a simple yet effective mechanism called MCD to detect such an MCF. We showed by simulation that an MCD is able to detect an MCF within 8 time units as long as the MCF does not start too ahead of the MCD. We implemented both NSWF and EG solution methods in the Controller and found that it is easier to detect an MCF when the Controller uses EG. As a future step, we plan to study multiple MCFs present at a time and determine how the MCD needs to be adjusted.

References

[1] S. S. Mwanje and C. Mannweiler. Towards cognitive autonomous networks: Network management automation for 5g and beyond. John Wiley & Sons, 2020.

[2] 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.

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

[4] A. Banerjee, S. S. Mwanje, and G. Carle. Optimal configuration determination in cognitive autonomous networks. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 494–500, 2021.

[5] B. Ross, L. Pilz, B. Cabrera, F. Brachten, G. Neubaum, and S. Stieglitz. Are social bots a real threat? an agent-based model of the spiral of silence to analyse the impact of manipulative actors in social networks. *European Journal of Information Systems*, 28(4):394–412, 2019.

[6] X. Song, W. Jiang, X. Liu, H. Lu, Z. Tian, and X. Du. A survey of game theory as applied to social networks. *Tsinghua Science and Technology*, 25(6):734–742, 2020.

[7] K. Koorehdavoudi, S. Roy, and M. Xue. Distributed decision-making algorithms with multiple manipulative actors: A feedback-control perspective. In *2018 IEEE conference on decision and control (CDC)*, pages 4439–4444. IEEE, 2018.

[8] E A M Ghalamzan, F. Abi-Farraj, P R Giordano, and R. Stolkin. Human-in-the-loop optimisation: mixed initiative grasping for optimally facilitating post-grasp manipulative actions. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3386–3393. IEEE, 2017.

[9] L. J. Mirman, L. Samuelson, and A. Urbano. Duopoly signal jamming. *Economic Theory*, 3(1):129–149, 1993.

[10] T. D. Jeitschko, T. Liu, and T. Wang. Information acquisition, signaling and learning in duopoly. *International Journal of Industrial Organization*, 61:155–191, 2018.

[11] R. N. Clarke. Collusion and the incentives for information sharing. *The Bell Journal of Economics*, pages 383–394, 1983.

[12] D. Fried. Incentives for information production and disclosure in a duopolistic environment. *The Quarterly Journal of Economics*, 99(2):367–381, 1984.

[13] G. J. Mailath. An abstract two-period game with simultaneous signaling—existence of separating equilibria. *Journal of Economic Theory*, 46(2):373–394, 1988.

[14] E. Gal-Or. Multiprincipal agency relationships as implied by product market competition. *Journal of Economics & Management Strategy*, 6(1):235–256, 1997.

[15] D. Fudenberg and J. Tirole. A signal-jamming theory of predation. *RAND Journal of Economics*, pages 366–376, 1986.

[16] R. Gibbons. Incentives and careers in organizations. 1996.

[17] B. Holmström. Managerial incentive problems: A dynamic perspective. *Review of Economic studies*, pages 169–182, 1999.

[18] A. S. Kyle. Continuous auctions and insider trading. *Econometrica*, pages 1315–1335, 1985.

[19] M. H. Riordan. Imperfect information and dynamic conjectural variations. *The RAND Journal of Economics*, pages 41–50, 1985.

[20] John F Nash Jr. The bargaining problem. *Econometrica: Journal of the Econometric Society*, pages 155–162, 1950.

[21] Edmund Eisenberg and David Gale. Consensus of subjective probabilities: The pari-mutuel method. *The Annals of Mathematical Statistics*, 30(1):165–168, 1959.

[22] E. Van Damme. The nash bargaining solutions is optimal. *Journal of Economic Theory*, 38(78):100, 1986.

[23] Nokia Siemens Networks. White paper: Self-organizing network: Introducing the nokia siemens networks son suite-an efficient, future-proof platform for son. Technical report, October, 2009.

[24] A. Banerjee, S. S. Mwanje, and G. Carle. On the implementation of cognitive autonomous networks. In *Internet Technology Letters. 2021;e317*, 2021. doi: 10.1002/itl2.317.

[25] A. Banerjee, S. S. Mwanje, and G. Carle. Towards control and coordination in cognitive autonomous networks. pages 1–1, 2021. doi: 10.1109/TNSM.2021.3116308.

[26] P. Agyapong et al. Deliverable d6. 1-simulation guidelines. 2013.

[27] Meinilä et al. Wireless world initiative new radio winner+, d5. 3: Winner+ final channel models. *CELTIC Telecommunication Soultions, Tech. Rep*, 2010.