Asynchronous algorithm of an endogenous peer-to-peer electricity market

Alyssia Dong, Thomas Baroche, Roman Le Goff Latimier, Hamid Ben Ahmed

Asynchronous algorithm of an endogenous peer-to-peer electricity market

Dong Alyssia, Baroche Thomas, Le Goff Latimier Roman, Ben Ahmed Hamid
SATIE Laboratory
Ecole Normale Superieure of Rennes
Rennes, France
{alyssia.dong, tbaroche, roman.legoff-latimier, benahmed}@ens-rennes.fr

***Abstract*—As the number of actors in the electricity market increases, peer to peer decentralized markets gain interest despite facing two challenges. First, the amount of exchanged messages increase significantly, w hich m akes t he r esolution d ependent on communication delays, as well as computation delays. Second, the physical limitations of the power network are not taken into account. An asynchronous implementation of an endogenous peer-to-peer market is here introduced to address these difficulties. The proposed algorithm is tested on a 31 agents testcase and a study of the influence o f t he v arious a lgorithm p arameters on the convergence time is performed.**

***Index Terms*—peer-to-peer, market, endogenous, optimal power flow, a synchronous, delays**

I. Introduction

The economic dispatch problem allows the producers and consumers of an electrical network to schedule in advance the production and consumption plan. As the number of distributed energy resources (DERs) grow, it can be argued that a decentralized clearing mechanism is more suited to the problem than a centralized one. Indeed, decentralization allows scalability and data privacy, in exchange of communication between agents. The ADMM based peer-to-peer market clearing algorithm proposed in [1], from which we base our study, also enables agents to express heterogeneous preferences.

Yet, the market clearing alone is not sufficient as it does not take into account the physical limitations of the network. A system operator (SO) needs to verify the physical feasibility of the solution in terms of line congestions and voltage limitations. Involving the SO into the resolution to ensure feasibility is thus mandatory. However, the resolution of the problem still needs to be decentralized. This induces a significant increase of exchanged messages both between peers and with the SO. Therefore, the convergence rate of the peer-to-peer algorithm heavily depends on the state of the communication network and the computation time of the various agents.

The convergence of asynchronous ADMM based consensus algorithms have been studied in [2]–[4], whether it be distributed or fully decentralized. In both cases, convergence of the algorithm was proven under the assumption of either bounded delays, or that the probability of each message to arrive was non null. Moreover, in several papers the asynchronous versions of an optimal power flow (OPF) were studied. In [5], the asynchronous, ADMM based, distributed OPF reduced significantly the convergence time while also increasing the gap of the objective function. In [6], an ADMM based distributed OPF was also studied, but this time on a lateral branching network. Different strategies were investigated to replace the loss of information due to the asynchrony, which lead to fluctuations in the solutions. A Lagrangian relaxation based decentralized OPF was studied in [7], in which individual markets communicate with each other asynchronously. The asynchronous updates lead to the optimal solution with up to 50% time reduction. The convergence of the asynchronous resolution of a multi-time step energy optimization problem in a microgrid, using ADMM is discussed in [8]. The results may deviate from the optimal solution if the communication delays are too high. In all of these decentralized schemes, there is a communication link between agents that are physically connected by a power line, as opposed to our algorithm in which the communication links are independent of the power network topology.

After formulating the endogenous peer-to-peer market in II, we propose in III an asynchronous algorithm to avoid communication or computation latencies. The simulation platform presented in IV is based on a 31 agents testcase. The simulation results discussed in V confirm that the asynchronous algorithm converges towards the same solution as the synchronous one regarding the power dispatch. The impact of various delays and asynchronism parameters are further investigated.

II. Endogenous peer-to-peer market

A. AC regularized peer-to-peer electricity market

The problem addressed in this paper is an optimal power flow based on a peer-to-peer market clearing algorithm described in [1], consisting of N agents part of a community Ω, performing multi-bilateral trades with each other. A system operator (SO) is added here to verify that the AC networks constraints are respected and to induce the market peers to change the solution otherwise. In order to account for the network losses, a new agent is added to the peer-to-peer market, called the *loss provider*. This agent is buying the losses to the producers of the market. We note the extended set of peer-to-peer market agents $\Omega^* = \Omega \cup \{Loss\}$ with

$Loss$ representing the loss provider agent. The AC regularized peer-to-peer electricity market problem is written in (1). The variables p_n (resp. q_n) represents the active (resp. reactive) power injected or consumed by agent $n \in \Omega^*$ in the network. It is positive if the power is actually produced, negative if consumed. The variable p_{nm} represents the active power traded from agent n to agent m. We note ω_n the set of all market partners of agent n.

$$\min_{(p_n)_{n\in\Omega^*},(q_n)_{n\in\Omega^*},\mathbf{P}} \sum_{n\in\Omega^*} c_n(p_n,q_n) + \tilde{\zeta}_{AC}(P,Q) \quad (1a)$$

$$\text{s.t. } \mathbf{P} = \mathbf{P}^T \quad (1b)$$

$$p_n = \sum\nolimits_{m\in\omega_n} p_{nm} \qquad n \in \Omega^* \quad (1c)$$

$$p_n^{min} \le p_n \le p_n^{max} \qquad n \in \Omega^* \quad (1d)$$

$$q_n^{min} \le q_n \le q_n^{max} \qquad n \in \Omega^* \quad (1e)$$

$$p_{Loss} = -\sum\nolimits_{n\in\Omega} p_n \quad (1f)$$

Equation (1a) consists of minimizing the cost function of all the agents in the community, while remaining within the physical constraints. The regularization function $\tilde{\zeta}_{AC}$ is equal to zero if the AC network constraints are respected, and $+\infty$ if they are violated. The bilateral trades constraint (1b) ensures that the trades are symmetrical: $p_{nm} = -p_{mn}$. The total power produced or consumed by agent n is equal to the sum of all its trades, as stated in (1c). Equations (1d)-(1e) account for the physical limitations of the producers and consumers, whilst the network losses are estimated in (1f).

B. ADMM problem resolution

The problem described in (1) is decomposed into smaller problems using the ADMM method, which leads to (2). The mathematical steps leading to these equations are detailed in [1]. Each smaller problem is local to an agent $n \in \Omega^*$. Iteratively, each agent $n \in \Omega^*$ solves its own problem, described in (2a)-(2d), and communicates the latest results with its trading partners $m \in \omega_n$ and the SO. At the same time, the SO performs its own computations described in (2e)-(2g), and in turn communicates the latest results with the market peers in Ω^*. Equations (2h)-(2j) refer to the ADMM dual variables updates performed by market peers and the SO. The superscript k represents the global iteration counter. For readability reasons, the function σ^ρ is introduced in (3). It represents the augmented lagrangian part of the new objective functions, with ρ being the ADMM penalty factor.

$$\begin{aligned}(p_n,q_n,P_n)^{k+1} = \underset{p_n,q_n,P_n=(p_{nm})_{m\in\omega_n}}{\operatorname{argmin}} \; & c_n(p_n,q_n) \quad (2a)\\ & + \sum\nolimits_{m\in\omega_n} \sigma^\rho\left(p_{nm}, \tfrac{p_{nm}^k - p_{mn}^k}{2}, \lambda_{nm}^k\right)\\ & \quad + \sigma^\rho\left(p_n, \tfrac{p_n^{SO,k}+p_n^k}{2}, \eta_n^{p,k}\right)\\ & \quad + \sigma^\rho\left(q_n, \tfrac{q_n^{SO,k}+q_n^k}{2}, \eta_n^{q,k}\right)\\ \text{s.t. } p_n = & \sum\nolimits_{m\in\omega_n} p_{nm} \quad (2b)\\ p_n^{min} \le & \; p_n \le p_n^{max} \quad (2c)\\ q_n^{min} \le & \; q_n \le q_n^{max} \quad (2d)\end{aligned}$$

$$\begin{aligned}(P^{SO},Q^{SO})^{k+1} = \underset{\substack{P^{SO}=(p_n^{SO})_{n\in\Omega},\\ Q^{SO}=(q_n^{SO})_{n\in\Omega}}}{\operatorname{argmin}} \; & \tilde{\zeta}_{AC}(P^{SO},Q^{SO}) \quad (2e)\\ & + \sum\nolimits_{n\in\Omega} \Big[\sigma^\rho\left(\tfrac{p_n^{SO,k}+p_n^k}{2}, p_n^{SO}, \eta_n^{p,k}\right)\\ & \quad + \sigma^\rho\left(\tfrac{q_n^{SO,k}+q_n^k}{2}, q_n^{SO}, \eta_n^{q,k}\right)\\ & + \sigma^\rho\left(p_n^{SO,k} - \tfrac{1}{|\Omega|}\tfrac{p_{Loss}^{SO,k}+p_{Loss}^k}{2}, p_n^{SO} - \tfrac{1}{|\Omega|}p_{Loss}^{SO,k}, \tfrac{1}{|\Omega|}\eta_{Loss}^{p,k}\right)\Big]\\ p_{Loss}^{SO,k+1} = & -\sum\nolimits_{n\in\Omega} p_n^{SO,k+1} \quad (2f)\\ q_{Loss}^{SO,k+1} = & \; q_{Loss}^k \quad (2g)\\ \lambda_{nm}^{k+1} = & \; \lambda_{nm}^k - \rho\left(p_{nm}^{k+1} + p_{mn}^{k+1}\right)/2 \quad (2h)\\ \eta_n^{p,k+1} = & \; \eta_n^{p,k} + \rho\left(p_n^{SO,k+1} - p_n^{k+1}\right)/2 \quad (2i)\\ \eta_n^{q,k+1} = & \; \eta_n^{q,k} + \rho\left(q_n^{SO,k+1} - q_n^{k+1}\right)/2 \quad (2j)\end{aligned}$$

$$\sigma^\rho(x,y,z) = z(y-x) + \rho/2\,(y-x)^2 \quad (3)$$

It should be noted that (2e) is equivalent to performing a classical optimal power flow computation in which the objective function of each agent $n \in \Omega$ is as follows in (4):

$$\begin{aligned}p_n^{SO}, q_n^{SO} \mapsto & \sigma^\rho\left(\tfrac{p_n^{SO,k}+p_n^k}{2}, p_n^{SO}, \eta_n^{p,k}\right) + \sigma^\rho\left(\tfrac{q_n^{SO,k}+q_n^k}{2}, q_n^{SO}, \eta_n^{q,k}\right)\\ & + \sigma^\rho\left(p_n^{SO,k} - \tfrac{1}{|\Omega|}\tfrac{p_{Loss}^{SO,k}+p_{Loss}^k}{2}, p_n^{SO} - \tfrac{1}{|\Omega|}p_{Loss}^{SO,k}, \tfrac{1}{|\Omega|}\eta_{Loss}^{p,k}\right) \quad (4)\end{aligned}$$

The SO performs the OPF near the latest solution of the trading peers $\left((p_n^k)_{n\in\Omega}, (q_n^k)_{n\in\Omega}\right)$ which allows each agent $n \in \Omega$ to keep their cost function private. The losses are then estimated by the SO using (2f).

The algorithm stops once the global primal and dual residuals reach a given value, as stated in (5).

$$\sum\nolimits_{n\in\Omega^*} \epsilon_n^{p,k+1} \le {\epsilon^{p,tol}}^2 \text{ and } \sum\nolimits_{n\in\Omega^*} \epsilon_n^{d,k+1} \le {\epsilon^{d,tol}}^2 \quad (5)$$

The global residual is made up of the sum of the local residuals, which are described in (6) and (7) for the primal and dual local residuals respectively.

$$\begin{aligned}\epsilon_n^{p,k+1} = & \frac{1}{4}\sum\nolimits_{m\in\omega_n}\left(p_{nm}^{k+1} + p_{mn}^{k+1}\right)^2 \quad (6)\\ & + \frac{1}{4}\left(p_n^{SO,k+1} - p_n^{k+1}\right)^2 + \frac{1}{4}\left(q_n^{SO,k+1} - q_n^{k+1}\right)^2\end{aligned}$$

$$\begin{aligned}\epsilon_n^{d,k+1} = & \sum\nolimits_{m\in\omega_n}\left(p_{nm}^{k+1} - p_{nm}^k\right)^2 \quad (7)\\ & + \left(p_n^{SO,k+1} - p_n^{SO,k}\right)^2 + \left(q_n^{SO,k+1} - q_n^{SO,k}\right)^2\end{aligned}$$

C. Communication considerations

The variables $p_n, q_n, (p_{nm})_{m\in\omega_n}, (\lambda_{nm})_{m\in\omega_n}, \eta_n^p$ and η_n^q are local to the peer agent $n \in \Omega^*$. In order to perform its calculations in (2a)-(2d) and (2h)-(2j), agent n also needs to know the latest values of the variables $(p_{mn})_{m\in\omega_n}, (q_{mn})_{m\in\omega_n}$ which are communicated by all its trading partners $m \in \omega_n$ and also p_n^{SO}, q_n^{SO} which are communicated by the SO. Moreover, the variables $(p_n^{SO})_{n\in\Omega^*}, (q_n^{SO})_{n\in\Omega^*}, (\eta_n^p)_{n\in\Omega^*}$ and $(\eta_n^q)_{n\in\Omega^*}$ are local to the SO. In order to perform its calculations described in (2e)-(2g) and (2i)-(2j), the SO also needs to receive messages from the market peers that contain the latest

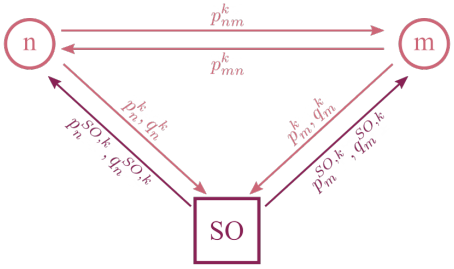

Fig. 1. Diagram of the various data exchanges between peers and between the SO and the peers during the k-th iteration. In this example, agents n and m are the only two market peers, and the SO represents the system operator.

values of the variables $(p_n)_{n\in\Omega^*}, (q_n)_{n\in\Omega^*}$. The various data exchanges are represented in Fig.1 using an example market constituted of two peer agents and the SO.

At the beginning of every iteration, each agent has to wait for the arrival of $l+1$ messages: $l = |\omega_n|$ messages from the other peers and one from the SO. The waiting period depends solely on the last message to arrive, which can be troublesome if one of the agents takes longer to perform its computation or if part of the communication network is experiencing a slowdown due to congestion. As the market size increases, so does $|\omega_n|$, thus these issues might considerably slow down the global convergence time of the algorithm.

III. Asynchronous implementation

As discussed in II-C, the synchronous version of the endogenous peer-to-peer algorithm presented in the previous section is very sensitive to communication and computation delays. To overcome this issue and make the algorithm more resilient to delay variations, an asynchronous implementation of the algorithm is introduced in this section. The asynchronous version is based on waiting for only part of the messages to arrive before continuing to perform the calculations described in (2). Not waiting for the last messages to arrive is going to significatively reduce the waiting period at every iteration. In addition, the asynchronous version allows the SO to perform fewer costly calculations.

In order for the asynchronous implementation to successfully converge, a few modifications need to be done in comparison to the synchronous version.

Firstly, the fact that not all messages are taken into account within the same iteration makes the superscript k, which was the global iteration counter, obsolete. To replace it, we introduce the local iteration counters $(k_n)_{n\in\Omega^*}$ for each of the market peers and k_{SO} for the SO. This counter represents the number of computations performed so far respectively by each agent n or by the SO.

Secondly, as discussed in section II-C, the values $p_n^{SO}, q_n^{SO}, (p_{mn})_{m\in\omega_n}$ received by a peer agent $n \in \Omega^*$ are stored in local copies. These local copies are obviously only updated if a message from either the SO or agent $m \in \omega_n$ arrives to agent n. The same goes for the SO local copies of $(p_n, q_n)_{n\in\Omega^*}$.

Algorithm 1 Market peer $n \in \Omega^*$ asynchronous process

Require: ρ, σ^ρ, ω_n, c_n, p_n^{min}, p_n^{max}, q_n^{min}, q_n^{max}

procedure MarketPeerProcess
  initialize $p_n, q_n, (p_{nm})_{m\in\omega_n}$ ▷ local variables
  initialize $(p_{mn})_{m\in\omega_n}, p_n^{SO}, q_n^{SO}$ ▷ local copies
  initialize $(\lambda_{nm})_{m\in\omega_n}, \eta_n^p, \eta_n^q$ ▷ dual variables
  initialize $k_n, (k_{nm})_{m\in\omega_n}, k_{nSO}$ ▷ local counters
  send p_{nm} to $m \in \omega_n$
  send p_n, q_n to the SO
  repeat
    wait for messages from $\Phi_n^{k_n}$ following (8)
    if SO $\in \Phi_n^{k_n}$ **then**
      update p_n^{SO}, q_n^{SO} from message
      $\eta_n^p \leftarrow$ (2i), $\eta_n^q \leftarrow$ (2j)
    for $m \in \Phi_n^{k_n}$ **do**
      update p_{mn} from message
      $\lambda_{nm} \leftarrow$ (2h)
    $(p_n^{temp}, q_n^{temp}, P_n^{temp}) \leftarrow$ (2a)
    if SO $\in \Phi_n^{k_n}$ **then**
      $p_n \leftarrow p_n^{temp}$, $q_n \leftarrow q_n^{temp}$
      $k_{nSO} \leftarrow k_{nSO} + 1$
      send p_n, q_n to SO
    for $m \in \Phi_n^{k_n}$ **do**
      $p_{nm} \leftarrow p_{nm}^{temp}$
      $k_{nm} \leftarrow k_{nm} + 1$
      send p_{nm} to m
    $k_n \leftarrow k_n + 1$
  until global convergence

Algorithm 2 System operator asynchronous process

Require: ρ, σ^ρ, $\Omega^* = \Omega \cup \{\text{lossProvider}\}$, $\tilde{\zeta}_{AC}$

procedure SystemOperatorProcess
  initialize $(p_n^{SO})_{n\in\Omega^*}, (q_n^{SO})_{n\in\Omega^*}$ ▷ local variables
  initialize $(p_n)_{n\in\Omega^*}, (q_n)_{n\in\Omega^*}$ ▷ local copies
  initialize $(\eta_n^p)_{n\in\Omega^*}, (\eta_n^q)_{n\in\Omega^*}$ ▷ dual variables
  initialize $k_{SO}, (k_{SOn})_{n\in\Omega^*}$ ▷ local counters
  send p_n^{SO}, q_n^{SO} to $n \in \Omega^*$
  repeat
    wait for messages from $\Phi_{SO}^{k_{SO}}$ following (8)
    for $n \in \Phi_{SO}^{k_{SO}}$ **do**
      update p_n, q_n from message
      $\eta_n^p \leftarrow$ (2i), $\eta_n^q \leftarrow$ (2j)
    $(P^{SO,temp}, Q^{SO,temp}) \leftarrow$ (2e)
    for $n \in \Phi_{SO}^{k_{SO}} \setminus \{\text{lossProvider}\}$ **do**
      $p_n^{SO} \leftarrow p_n^{SO,temp}$, $q_n^{SO} \leftarrow q_n^{SO,temp}$
      $k_{SOn} \leftarrow k_{SOn} + 1$
      send p_n^{SO}, q_n^{SO} to n
    if lossProvider $\in \Phi_{SO}^{k_{SO}}$ **then**
      $p_{Loss}^{SO} \leftarrow$ (2f), $q_{Loss}^{SO} \leftarrow$ (2g)
      $k_{SOLoss} = k_{SOLoss} + 1$
      send p_{Loss}, q_{Loss} to lossProvider
    $k_{SO} \leftarrow k_{SO} + 1$
  until global convergence

Thirdly, the local variables don't need to be updated at every local iteration. Let $n \in \Omega^*$ be a market peer. Its local variables p_{nm} and λ_{nm} are updated only if a message from agent m has been received since the last local iteration. In the same way, its local variables p_n, q_n, η_n^p and η_n^q are updated only if a message from the SO has been received since the last local iteration. The same update condition is applied for the SO to the variables p_n^{SO}, q_n^{SO}, η_n^p and η_n^q for any $n \in \Omega^*$.

Lastly, let us define a linkwise iteration counter k_{nm} which represents the number of updates of the variable p_{nm} if $m \in \Omega^*$, and of the variable p_n if $m = \text{SO}$. When agent n receives a message from agent $m \in \omega_n$, the local copy p_{mn} only gets updated if $k_{nm} = k_{mn}$. The same applies to the local copies p_n^{SO} and q_n^{SO}. We define $\Phi_n^{k_n}$ as followed in (8):

$$\Phi_n^{k_n} = \left\{ \; m \in \omega_n \cup \{SO\} \;\middle|\; \begin{array}{c} \text{message received from } m \\ \text{during iteration } k_n \\ \text{and } k_{nm} = k_{mn} \end{array} \right\} \tag{8}$$

The local computation of agent n is triggered as soon as enough messages have been received in compliance with the $k_{nm} = k_{mn}$ constraint. The local iteration k_n can be incremented once $|\Phi_n^{k_n}|$ hits a threshold whose influence will be discussed in V.

The asynchronous version of the decentralized endogenous negotiation mechanism are summarized in Algorithm 1 and Algorithm 2 for the market peer process and the system operator process respectively. We can use the equations as written in (2) so long as any variable with the superscript k is replaced by the latest updated version of the variable.

IV. Simulation platform

The asynchronous implementation of the endogenous peer-to-peer market algorithm has been written in Julia. The open-source code can be found on Gitlab[1]. The various message exchanges have been simulated using a discrete event simulator: SimJulia. This simulation framework allows to add communication and computation delays between each iteration of every agent. The local peer problem written in (2a) being a convex quadratic problem, it is solved using OSQP [9]. As for the SO problem in (2e), it is solved using the PowerModels [10] package in Julia as an AC optimal power flow model with polar bus voltage variables. All simulations were run on a Ryzen 9 3950X 16 core 3.5GHz/4.7GHz processor.

The algorithm is applied on a testcase inspired by the IEEE 39-bus test system, which was adapted in order to make the consumers flexible, as in [1], [11]. This testcase is composed of 21 consumers and 10 producers, which each have a quadratic cost function written as follows:

$$c_n(p_n) = \frac{1}{2} a_n p_n^2 + b_n p_n \tag{9}$$

Every consumer is a trading partner to every producer, and vice versa. This hypothesis allows the final results to be equivalent to that of a traditional optimal power flow.

[1]https://gitlab.com/satie.sete/endogenousp2p

TABLE I
Average and standard deviation of computation delays in Julia

	PowerModels AC-OPF	**OSQP peer**	
		producer	***consumer***
$\mathbb{E}$[computation delay]	75 ms	77 μs	58 μs
$\boldsymbol{\sigma}$(computation delay)	15 ms	22 μs	18 μs

To model the communication delays between two agents, every agent gets randomly assigned a 2-dimensional coordinate. The farther two agents are from each other, the slower the communication delay. The distance between any two agents is defined as the euclidean distance of their coordinates. Within this illustrative testcase, the distance d_{nm} should not only be seen as a representation of the geographical distance. It can also be interpreted as proportional to the number of communication nodes that add latency – e.g. routers.

The communication delay model is posed as a simple deterministic parametric model such that:

$$\Delta T_{nm} = \alpha \, d_{nm} + \beta \tag{10}$$

where ΔT_{nm} represents the communication delay of a message sent by agent n to agent m. The parameters (α, β) represent the state of the communication network. The communication parameter β is the minimal delay of all communications, whereas α accounts for the variations of delays between every communication link. These parameters are set to have delays ranging from 10 ms to 100 ms, which is roughly the delays experienced for an internet connection.

As for the computation delays, the market peer local problem resolution is 1000 times faster than the AC OPF performed by the SO, according to Table I. Only the SO computation delays will therefore be taken into account thereafter. The following results hold as long as the computation delays of every market peer is negligible compared to that of the SO. This hypothesis should be readily validated even if peers have modest computing capabilities, considering the difference in complexity between the two problems.

V. Simulation results

A. Results with communication delays

First of all, let us compare the solutions of the endogenous problem with the reference solution given by Matpower AC OPF in Fig.2. We can observe that the endogenous solutions, whether synchronous or asynchronous, are similar with respect to the active power injections. The same remark can be made for the reactive power injections and for any value of δ_{peers} and δ_{SO}. The substantially small, but observable difference between solutions are due to the various precision errors that are carried over the few hundreds iterations for each local agents.

The main focus of our study is the variations of the convergence time of the algorithm and the number of computations regarding to the asynchronous implementation. A single value $\epsilon^{p,tol^2} = 10^{-3}$ is thus chosen for the stopping criterion on the

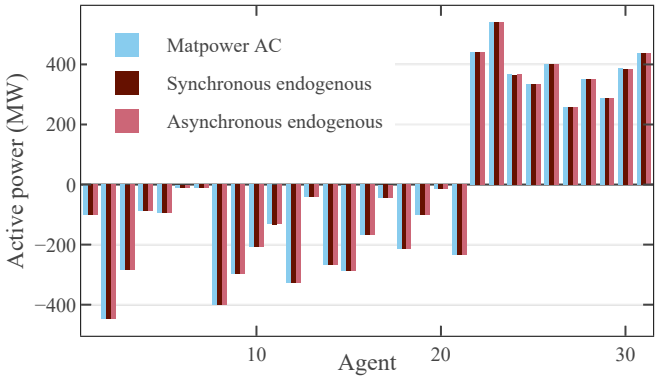

Fig. 2. Power injection comparison between Matpower and endogenous negotiation mechanisms. Asynchronous case with $\delta_{peers} = \delta_{SO} = 0.2$.

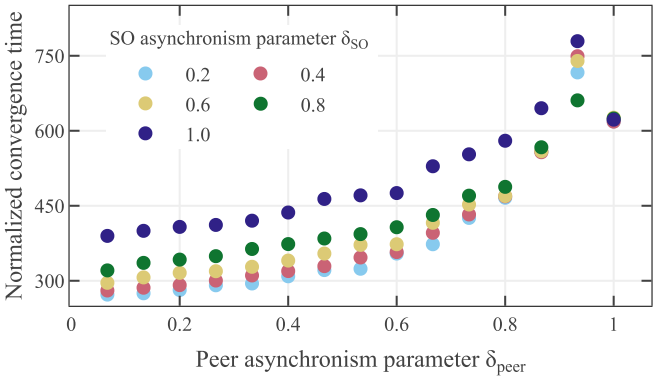

Fig. 3. Convergence time normalized by the average communication delay versus the peer asynchronism parameter δ_{peer} and the SO asynchronism δ_{SO}.

primal residual. Similar results can be performed with different values or a dual residual criterion.

The peer asynchronous parameter δ_{peer} is defined as the minimum proportion of messages that any peer must receive before beginning the local iteration calculations:

$$\delta_{peer} = \frac{|\Phi_n^{k_n}|}{|\omega_n| + 1} \qquad n \in \Omega^* \tag{11}$$

with $|\cdot|$ being the number of elements in the set. If a peer receives as many or more messages as $\lceil \delta_{peer} \cdot (|\omega_n| + 1) \rceil$, with $\lceil \cdot \rceil$ being the ceiling function, it triggers his local update. For the sake of interpretability of the results, all peers are set to the same δ_{peer} value although they might differ in a real implementation. Likewise, the SO asynchronous parameter δ_{SO} is defined as the minimum proportion of messages that the SO must receive before performing the local iteration calculations: $\delta_{SO} = |\Phi_{SO}^{k_{SO}}|/|\Omega^*|$. The case $\delta_{peer} = \delta_{SO} = 1$ corresponds to the synchronous case because every agent and the SO must wait for the maximum possible number of messages.

Fig. 3 shows the normalized convergence time of the asynchronous algorithm for various values of δ_{peer} and δ_{SO}. Here, we only take into account the communication delays, which means that all computation delays are set to zero. The convergence time has been normalized by the average communication delay, which is equal to 60 ms for this case study. We can observe that the convergence time increases with the peer asynchronism parameter δ_{peer}, except when $\delta_{peer} = 1$. The convergence time also increases with the SO asynchronism parameter δ_{SO} as long as δ_{peer} is not close to 1. However, δ_{SO} has a smaller impact on convergence time than δ_{peer}. The configuration that seems to minimize the convergence time is the one that has the smallest values of δ_{peer} and δ_{SO}. This configuration make the agents perform a computation every time a message is received, which significantly increases the number of local peer computations, and of SO computations, which we will define as k_{SO}^*.

As we can observe in Table II, the average number of computations k_{SO}^* increases as the SO asynchronous parameter δ_{SO} decreases. There is a five times increase in the number of SO computations k_{SO}^* for $\delta_{SO} = 0.2$ compared to the synchronous case. This significant increase leads to a large computational workload for SO, which means that its processing time needs to be taken into account throughout the subsequent study.

TABLE II
AVERAGE AND STANDARD DEVIATION OF THE SO'S NUMBER OF COMPUTATIONS k_{SO}^* W.R.T THE SO ASYNCHRONOUS PARAMETER δ_{SO}

δ_{SO}	1.0	0.6	0.2
$\mathbb{E}[k_{SO}^*]$	325	544	1603
$\sigma[k_{SO}^*]$	1.6	14.6	82.9

B. Results with communication and computation delays

In this section, we add various values of computation time for the system operator. On the basis of Table I and II, the SO will not have enough time to compute a full OPF between each received message if his computation time has the same order of magnitude than communication delays. Thus, the real number of SO's computation will probably be smaller the one in Table II. Let us note c_{SO} the *computation to communication ratio*, which is equal to the SO computation time normalized by the average communication delay. Figure 4 shows heatmaps of normalized convergence time with respect to both asynchronous parameters δ_{peer} and δ_{SO}, and for three values of the c_{SO} ratio. The first heatmap, with $c_{SO} = 0$, represents the same case study as the previous section where there were no computation time, for comparison. The second and third heatmaps show $c_{SO} = 1$ and $c_{SO} = 3$ respectively. We can observe that the convergence time is still the lowest for low values of δ_{peer} and δ_{SO}. The peer asynchronism parameter δ_{peer} has a bigger impact on convergence time than the SO asynchronism parameter δ_{SO}, whatever the value of c_{SO}.

Understandably, as the SO computation time increases, so does the convergence time. However, when the computation time is bigger than the average communication delay (e.g. $c_{SO} = 3$), both asynchronous parameters do not have much of an impact on the convergence time. This is due to the fact that almost all messages are arriving at the SO during its

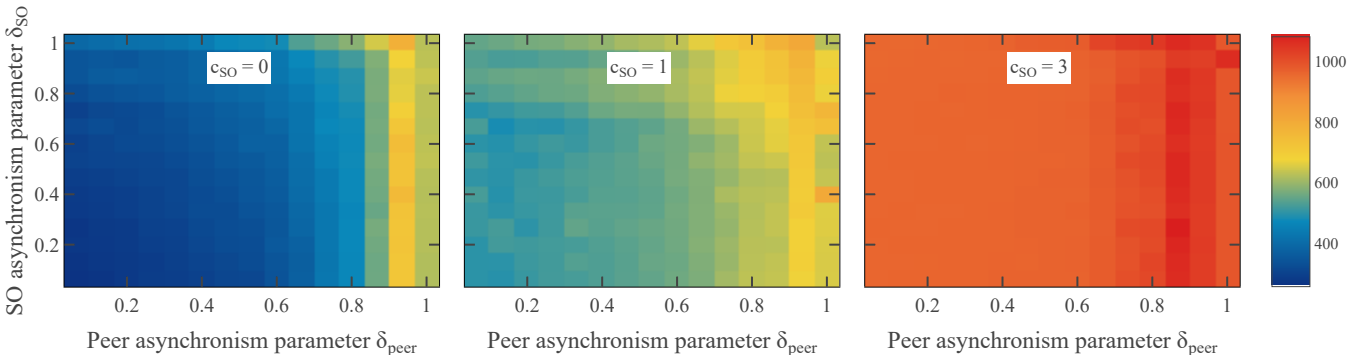

Fig. 4. Heatmaps of normalized convergence time with respect to asynchronous parameters δ_{peer} and δ_{SO} for three values of the SO computation time c_{SO}.

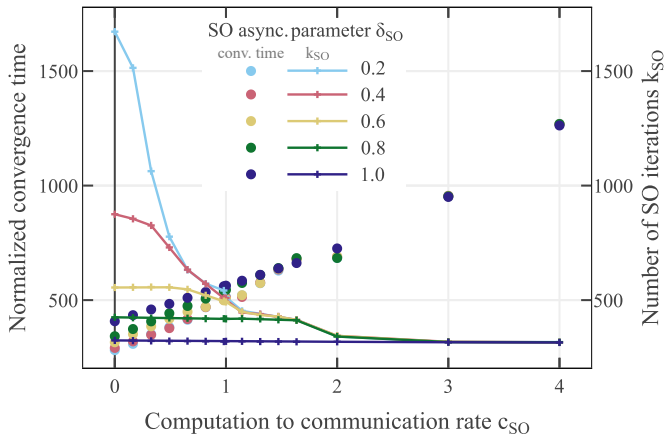

Fig. 5. Convergence time normalized by the average communication delay (markers) and total number of SO iterations k^*_{SO} (lines) w.r.t. the computation to communication ratio c_{SO}, depending on the SO asynchronism parameter δ_{SO}, with $\delta_{peer} = 0.2$.

computation, so the next iteration is equivalent to an almost synchronous update, whatever the value of δ_{SO}.

This phenomenon is also visible in Figure 5, which shows the influence of the computation to communication ratio c_{SO} on the convergence time and the total number of SO iterations k^*_{SO}. We can see that when $c_{SO} < 1$, the SO asynchronism parameter δ_{SO} does have an effect on the convergence time of the algorithm, as a counterpart of a significant increase in the number of SO calculations k^*_{SO}. However, as c_{SO} increases, the SO computation time becomes larger and larger, thus limiting the amount of performed computations and the time reduction between $\delta_{SO} = 0.2$ and $\delta_{SO} = 1.0$. The number of SO iteration decreases towards the value at $\delta_{SO} = 1.0$ as the computation time increases.

VI. Conclusion

The asynchronous implementation of an endogenous peer-to-peer market has been studied on a 31 agents testcase. We verified that the solution of the asynchronous problem was the same as performing an OPF. We showed that the peer asynchronous parameter δ_{peer} had an impact on the convergence time of the algorithm, whereas the SO asynchronous parameter δ_{SO} had an impact on the number of calculations performed by the SO to reach convergence. However, as the computation time of the SO increases, the number of computations that it can perform decreases and all values of δ_{SO} lead to the same performance. The proposed algorithm still calls for further validation, in particular an implementation on an experimental power system. In addition, with the increase of the number of participating agents, the SO computation time is bound to increase too. This makes mandatory to consider decentralizing the SO. Further studies will be done on the asynchronous implementation of this endogenous peer to peer market in the case of a decentralized SO.

References

[1] T. Baroche, "Peer-to-peer electricity markets in power systems," Ph.D. dissertation, École Normale Supérieure de Rennes, 2020.

[2] R. Zhang and J. Kwok, "Asynchronous distributed admm for consensus optimization," in *International conference on machine learning*. PMLR, 2014, pp. 1701–1709.

[3] T.-H. Chang, W.-C. Liao, M. Hong, and X. Wang, "Asynchronous distributed admm for large-scale optimization—part ii: Linear convergence analysis and numerical performance," *IEEE Transactions on Signal Processing*, vol. 64, no. 12, pp. 3131–3144, 2016.

[4] N. Bastianello, R. Carli, L. Schenato, and M. Todescato, "Asynchronous distributed optimization over lossy networks via relaxed admm: Stability and linear convergence," *IEEE Transactions on Automatic Control*, 2020.

[5] J. Guo, G. Hug, and O. Tonguz, "Impact of communication delay on asynchronous distributed optimal power flow using admm," in *2017 IEEE International Conference on Smart Grid Communications (SmartGridComm)*. IEEE, 2017, pp. 177–182.

[6] J. Xu, H. Sun, and C. J. Dent, "Admm-based distributed opf problem meets stochastic communication delay," *IEEE Transactions on Smart Grid*, vol. 10, no. 5, pp. 5046–5056, 2018.

[7] A. Huang, S.-K. Joo, K.-B. Song, J.-H. Kim, and K. Lee, "Asynchronous decentralized method for interconnected electricity markets," *International Journal of Electrical Power & Energy Systems*, vol. 30, no. 4, pp. 283–290, 2008.

[8] M. H. Ullah and J.-D. Park, "Distributed energy optimization in mas-based microgrids using asynchronous admm," in *2019 IEEE Power & Energy Society Innovative Smart Grid Technologies Conference (ISGT)*. IEEE, 2019, pp. 1–5.

[9] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "Osqp: An operator splitting solver for quadratic programs," *Mathematical Programming Computation*, pp. 1–36, 2020.

[10] C. Coffrin, R. Bent, K. Sundar, Y. Ng, and M. Lubin, "Powermodels. jl: An open-source framework for exploring power flow formulations," in *2018 Power Systems Computation Conference (PSCC)*. IEEE, 2018, pp. 1–8.

[11] T. Baroche, P. Pinson, R. L. G. Latimier, and H. B. Ahmed, "Exogenous cost allocation in peer-to-peer electricity markets," *IEEE Transactions on Power Systems*, vol. 34, no. 4, pp. 2553–2564, 2019.