Korner *et al. Energy Informatics* 2020, **3**(Suppl 1):13
https://doi.org/10.1186/s42162-020-00116-6

Energy Informatics

RESEARCH Open Access

Impact of communication latency on distributed optimal power flow performance

Clemens Korner[1*], Catalin Gavriluta[1], Filip Pröstl Andrén[1], Marcus Meisel[2,3] and Thilo Sauter[2,4]

From The 9th DACH+ Conference on Energy Informatics
Sierre, Switzerland. 29-30 October 2020

*Correspondence:
clemens.korner@ait.ac.at
[1]AIT Austrian Institute of Technology – Electric Energy Systems, Giefinggasse 2, 1210 Vienna, Austria
Full list of author information is available at the end of the article

Abstract
In this work, we study the performance of a distributed optimal power flow control algorithm with respect to realistic communication quality of service. By making use of a communication network simulator, namely "GNS3", we created a framework that simulates both the controllers involved in the optimal power flow algorithm and the communication between them. Using this platform, we investigate and give insights into the convergence time of the distributed algorithm when applied to the IEEE 13 and IEEE 123 node test feeders. By leveraging the simulation results, we define parameters on the network quality of service, such that the distributed optimal power flow algorithm could be used for secondary or tertiary control.
To deal with the complexity induced by a large number of components involved in these simulations, we present a methodology to automate and streamline the generation and the analysis of simulation scenarios.

Keywords: Alternating direction method of multipliers, Communication quality of service, Distributed generation, Network simulator, Distributed optimal power flow

Introduction

This section provides an overview of the two main topics which are covered by this paper and their related work. The first part is about the ongoing shift from large centralized power generation units to smaller more distributed generation. As a consequence, new decentralized and distributed control strategies have emerged which claim to solve most of the thereby arising problems. In the second part, we will discuss the need for proper communication network simulations for distributed control algorithms to evaluate their practical usability. These simulations are usually manually generated and include cumbersome and error-prone parametrization. The automatized preparation and generation of simulations can reduce the needed work and the time spent debugging.

With the increased penetration of distributed generators based on renewable energy, the number of controllable devices in the electrical grid is increasing (Kraning et al. 2014). In 2017 in the European Union, 85% of the newly installed power generation fell within the category of renewable energy resources (European Environment Agency 2018). This

trend challenges the centralized control methods used in electrical grids nowadays, which have limited scalability. Distributed and decentralized control strategies have been proposed to overcome this shortcoming (Gavriluta et al. 2016). Many of the newly proposed distributed control algorithms are based on the alternating direction method of multipliers (ADMM) which is explained in Boyd et al. (2011). For instance, in Gavriluta et al. (2016); Zhang et al. (2017); Erseghe (2014); Zhang et al. (2019) ADMM-based optimal power flow (OPF) algorithms have been presented. The various approaches solve either the original, nonconvex optimization problem, linearized power flow equations and constraints, or a convex relaxation of the problem. ADMM-based approaches and alternatives such as analytical target cascading and optimality condition decomposition (OCD) can be found in a survey by Molzahn et al. (2017).

However, in most of the literature describing these strategies, the communication layers are either highly simplified or not considered at all. Hence, based on the existing works, one cannot formulate quality of service (QoS) requirements for the communication network, such that the distributed control methods satisfy pre-imposed timing restrictions. Work in this direction could be found, for instance, in Guo et al. (2017) where test grids are partitioned into regions and the behavior of an ADMM- and OCD-based OPF algorithm is analyzed. The communication network is modeled with the OPNET network simulator and the convergence speed of centralized and distributed implementations of the algorithms is simulated. Different background traffic is applied, to investigate the impact of the communication delay. In contrast to this work, we do not introduce additional background traffic, but we emulate different communication QoS with network filters that are applied to the virtual Ethernet cables. Furthermore, in the work of Guo et al., the network is split into multiple regions, each consisting of several buses. In comparison, in our work each bus builds its own region, for instance, the IEEE 13 node test feeder consists of 13 buses and thus 13 regions. In Gavriluta et al. (2020) a cyber-physical framework for distributed control is proposed, where a distributed DC-OPF algorithm is executed on a fleet of single-board computers that are connected to a switch and an OPAL-RT simulator. Traffic control, a program to configure the kernel packet scheduler of Linux, is used to simulate different communication QoS. OPAL-RT simulates the electrical grid in which the controllers are used for secondary control. In comparison, the approach proposed in this paper does not need additional single board computers. However, it could be possible to couple GNS3 – the communication network simulator used in our approach – with such a cyber-physical framework to have a hybrid mixture of physical controllers and controllers simulated. Furthermore, our approach can also be used to evaluate cyber-security related use cases since it models replicas of real communication infrastructure. Theoretical investigations of the convergence behavior of ADMM-based distributed algorithms have been shown in multiple articles (Boyd et al. 2011; Ling et al. 2016; Xu et al. 2018).

To deal with the complexity induced by a large number of components involved in these simulations, we use a methodology to automate and streamline the generation and the analysis of simulation scenarios. For this, a machine-readable format is needed which can store both the topology information of the electrical and the information and communication technology (ICT) network. “Power System Automation Language” (PSAL) is a domain-specific language that provides the flexibility to describe electrical and ICT networks and was designed for rapid prototyping which includes automatic code generation

for simulations (Andrén et al. 2017). However, the current parser is written for the eclipse modeling framework, hence, difficult for other frameworks to use it directly. Therefore, we present in this paper mapping to the JavaScript Object Notation (JSON), which establishes interoperability to various tools and frameworks because JSON libraries exist for most programming languages.

In summary, the contributions of this paper are:

- a simulation framework based on the network simulator GNS3, on which we evaluate the implementation of a distributed OPF algorithm,
- "Power System Automation JSON" (PSAJ), a JSON mapping for PSAL, used to streamline the code generation for simulations, and
- the analysis of an ADMM-based distributed OPF algorithm for the IEEE 13 and IEEE 123 test feeders providing conclusions regarding its usability for secondary and tertiary control, given different communication QoS.

Problem statement

Before advancing to the methodology section, we will briefly discuss the use case under study, namely, distributed OPF.

Classical OPF problems are formulated in a centralized fashion and are solved on a single machine. Different objective functions for different OPF problems exist. In this paper we look into the so-called *power loss problem.* The aim of the power loss problem is the reduction of the active power losses in an electrical grid. This is equivalent to the minimization of injected active power into the grid. The cost function of the power loss problem has the form $\Re(\mathbf{v}^\top \overline{\mathbf{Y}}\overline{\mathbf{v}})$, where $\vec{v}$ is a vector that contains all the voltages for all the buses in the network; $\mathbf{Y}$ is the admittance matrix, the operator $\Re(\ )$ returns the real part of complex numbers, and the line over the voltage vector and admittance matrix means the complex conjugate.

To distribute the power loss problem, the cost function is formulated as a sum of power injections into each bus k, as shown in (1).

$$\Re\left(\mathbf{v}^\top \overline{\mathbf{Y}}\overline{\mathbf{v}}\right) = \sum_k \Re\left(v_{k[k]}\overline{\mathbf{Y}}_k\overline{\mathbf{v}}_k\right) \tag{1}$$

For each bus k, the local voltage vector $\mathbf{v}_k$ contains all the voltages of its adjacent buses and itself and the voltage $v_{k[k]}$ means the voltage of bus k. The admittance matrix $\mathbf{Y}_k$ describes a subgrid that contains only the bus k and its adjacent buses. It must be generated in such a way, that (1) holds. Each term in the cost function can be minimized by a different solver in parallel. This lays the foundation for a distributed OPF formulation. However, the terms are not independent of each other because different local voltage vectors share the same voltages. This is exemplified in Fig. 1, where the buses are visualized as circular nodes with their IDs in the middle. Beside each bus, the local voltage vector and its entries are shown, where the numbers inside the squared brackets in the subscript refer to the IDs of the buses to which the quantities are associated. Bus 2, for instance, has two neighboring buses: bus 1 and bus 3. Therefore, its voltage is present as $v_{2[2]}$ in the vector $\mathbf{v}_2$, furthermore, bus 1 holds the voltage for bus 2 in its local voltage vector $\mathbf{v}_1$ at position $v_{1[2]}$. Meanwhile, at bus 3 the voltage is kept at $v_{3[2]}$ as shown in orange in the figure.

By leveraging ADMM, a fully distributed formulation of the power loss problem can be found which provides a consensus strategy for the shared variables. This distributed OPF

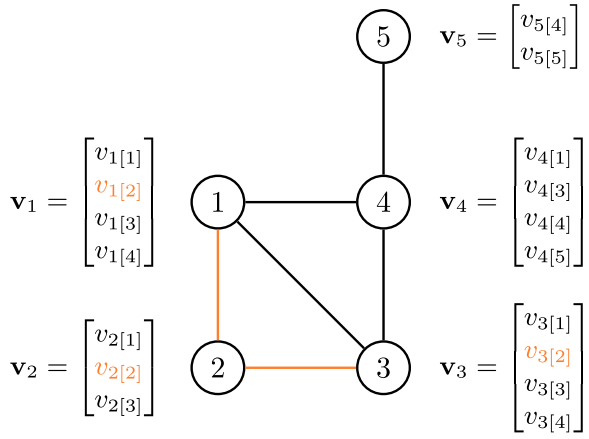

Fig. 1 The figure displays an electrical grid with five buses and their local voltage vectors $\vec{v}_k$. In orange interdependencies of local variables for the voltage of bus 2 are shown

problem needs to be solved by an iterative algorithm which consists of multiple steps per iteration, namely *optimization, consensus,* and *update.*

Optimization The ADMM formulation adds new terms to the objective function as shown in (2). Here, the vectors $\mathbf{z}_k^{[i-1]}$ help in the consensus finding process between the distributed subproblems. This optimization problem can be solved in parallel in each iteration i and for every bus k. A deviation from the consensus of the last iteration gets penalized with the penalty parameter ρ. The formulation of the augmented Lagrangian includes the equality constraint between the local voltages and the consensus variables into the objective function which in turn introduces the dual variables $\boldsymbol{\lambda}_k^{[i-1]}$.

$$\mathbf{v}_k^{[i]} := \arg\min_{\mathbf{v}_k} \Big(\Re\big(v_{k[k]} \overline{\mathbf{Y}}_k \overline{\mathbf{v}}_k\big) + \Re\Big(\boldsymbol{\lambda}_k^{[i-1]\top} \overline{\mathbf{v}}_k\Big) + \frac{\rho}{2} \|\mathbf{v}_k - \mathbf{z}_k^{[i-1]}\|_2^2 \Big) \tag{2}$$

Constraints restrict the minimum and maximum active and reactive power injections into the buses. In each subproblem, the voltages of the central bus and its adjacent buses have lower and upper boundaries.

Consensus After the optimization, a new vector $\boldsymbol{\beta}_k^{[i]}$ is calculated which is a linear combination of the newly obtained voltage vector and scaled dual variables from the last iteration, as shown in (3).

$$\boldsymbol{\beta}_k^{[i]} := \frac{1}{\rho} \boldsymbol{\lambda}_k^{[i-1]} + \mathbf{v}_k^{[i]} \tag{3}$$

Over a communication network, a bus k receives entries $\beta_{l[k]}^{[i]}$ which are associated with its voltage from all its adjacent buses l. On the left side of Fig. 2, an example of this message exchange is given.

Afterwards, a bus builds its consensus $z_{k[k]}^{[i]}$ by averaging the weighted variables $\beta_{l[k]}^{[i]}$, where l stands for the IDs of the central bus and its neighboring buses. Figure 2 shows how the consensus variable $z_{2[2]}^{[i]}$ is built.

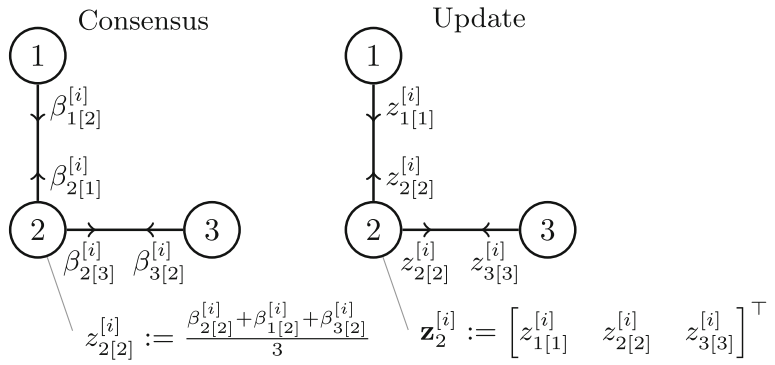

Fig. 2 The left part of the figure shows the communication done in the *consensus* step and how a consensus variable is calculated for the voltage of bus 2. On the right side, the exchange of the previously calculated consensus variables is presented

Update Subsequently, each bus sends its newly calculated consensus variables to all its neighboring buses. With those values, all the missing entries of $\bar{z}^{[i]}_k$ are updated. By using the new voltage vectors and consensus variables, the dual variables are updated in (4). The right side of Fig. 2 gives an example of the message exchange in the update step.

$$\boldsymbol{\lambda}^{[i]}_k := \boldsymbol{\lambda}^{[i-1]} + \rho \left(\mathbf{v}^{[i]}_k - \mathbf{z}^{[i]}_k \right) \tag{4}$$

It should be emsphasized that the nodes need to exchange data in both the consensus and the update step. Thus, the communication infrastructure and its QoS play an important role in how this algorithm performs.

Methodology

This section explains our chosen methodology for the implementation of the distributed power loss problem and the simulation of the communication QoS from bottom up. First, we begin by describing the implementation of the distributed algorithm and its embedding via Docker images into the GNS3 network simulator. The second part explains the need for an approach to automatically generate simulations and how this is done for the GNS3 simulations.

Implementation

Our formulation of the power loss problem, as shown in (2), is a nonconvex quadratically constrained quadratic program (QCQP). Different solutions such as convex relaxations, heuristics, and nonlinear optimization techniques for solving this type of problems exist. For our implementation, we decided to use COBYLA as a solver for the optimization problem. The controllers communicate via remote procedure calls (RPCs) with each other. A server – called stub in the context of RPCs – provides functions that can be called by clients. In particular, we use the library gRPC which uses HTTP/2 for the transmission of the messages and protocol buffers for their encoding.

We containerized the implementation with Docker such that it can be easily deployed on various platforms which include, among others, GNS3. Moreover, the Docker containers need two configuration files that are not included in the image. The first file configures the network interface of the container. In our simulation we use static IP addresses, therefore, the IP address and its corresponding subnet mask are stored in this configuration

file. The second file is a JSON file that contains all the needed parameters for the ADMM optimization. This includes, among others, the data from the electrical subgrid which is considered by the controller and the IDs and the IP addresses of all its adjacent controllers. Furthermore, the penalty parameter ρ and the number of iterations until the controllers stop the iterative algorithm are set.

Network simulator

One of the goals of this work is to measure the performance of the distributed OPF concerning different communication QoS. A major requirement on the network simulator is its support for user applications and their simple integration. GNS3 supports, among others, Docker containers and virtual machines via VirtualBox. This allows the seamless and simple integration of software applications that use standard network interfaces, such as Ethernet, for communication. Furthermore, GNS3 has out-of-the-box support for many network devices, for instance, CISCO routers and their original images. Packet filters can be applied to each network connection in the simulation to control QoS properties. With them, the communication latency and its jitter, and the probability for packet loss, respectively packet corruption can be parametrized.

The implementation of the described distributed OPF, from the "Implementation" section, can be directly used in GNS3, due to the simulator's support of Docker containers. In GNS3 a Docker container acts as a device that can communicate with other network components over an Ethernet interface. Each controller in the simulation is an instance of the same Docker image, which contains the implementation of the distributed algorithm.

Unfortunately, most of the openly accessible electrical grids have only information about their electrical components and do not provide any information about the corresponding ICT network. Due to this lack of data, we decided to interconnect all the controllers with GNS3's builtin layer-two network switches in a simplified network. All the switches are arranged in a daisy-chained topology and are interconnected with Ethernet cables with each other and with the controllers. Ethernet cables are modeled as ideal connections in GNS and network filters are used to define their communication QoS. Every switch, except the switches at the two ends, has two neighboring switches. For the simulation of the communication QoS, network filters are added on all network cables in the simulation, connecting the switches with the controllers. In contrast to this, no network filters are applied to the cables connecting the switches.

Figure 3 shows the GNS3 simulation which is associated with the example grid presented in Fig. 1. For each bus in the electrical grid, a controller is added to the GNS3 simulation. As mentioned before, every controller has two configuration files: one JSON-file, which contains the data needed for the distributed OPF, and another file, which has the configuration for the network interface of the Docker container. The containers are connected with switches and the icons on top of the lines indicate the network filters.

Automation

In this paper, we investigate the proposed distributed optimization algorithm by using the IEEE 13 and IEEE 123 node test feeders as test cases. Particularly, we want to identify the relation between the latencies of the network connections and the time needed for convergence of the algorithm. For each of these scenarios, the simulation has to be modified

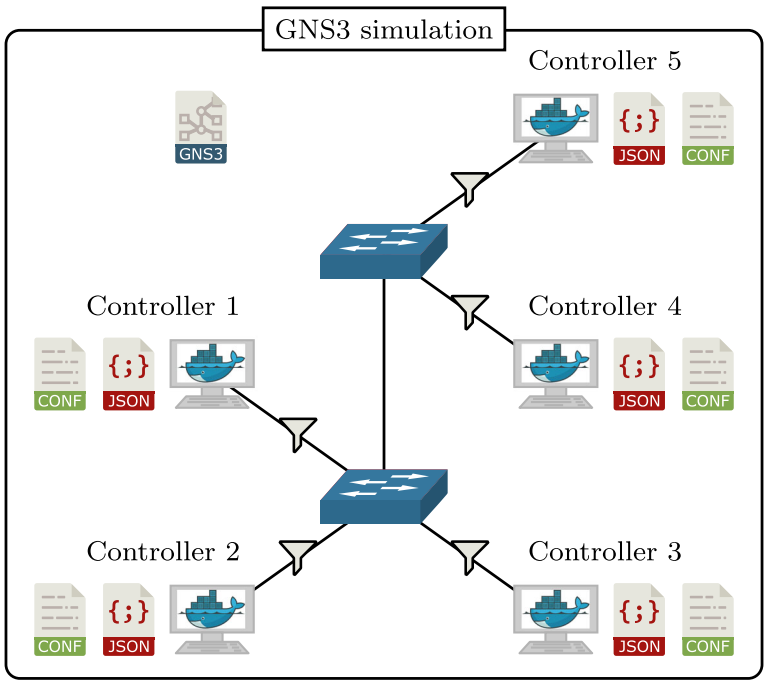

Fig. 3 The diagram displays the structure of a GNS3 simulation with its devices and their configuration files. Every controller is instantiated from the same Docker image which contains the implementation of the distributed OPF. Network filters are applied on all network cables connecting the controllers with the switches

accordingly. This implies, as Fig. 3 indicates, the adjusting of all the network filters and also the modification of all the configuration files of the Docker containers, if parameters of the distributed algorithm are changed. The generation, respectively, adjustment of the configuration files is both cumbersome and error-prone, if performed manually.

Therefore, we decided to store all the data in a format that is machine-readable, extensible, and can be easily customized with scripts. The domain-specific language called "Power System Automation Language" (PSAL) by Pröstl Andrén et al. (Andrén et al. 2017) offers all these features, except the customization with scripts apart from Java, as its parser is written in the Xtext framework for Eclipse. Due to this, we introduce *"Power System Automation JSON"* (PSAJ), which is compatible with PSAL but uses JSON to store its data. Frameworks for the data import and export of JSON files exist for most programming languages. This enables interoperability between PSAJ and various tools and environments.

We envision the usage of PSAJ, as shown in Fig. 4. It consists of three main steps: *serializer, transformer,* and *code generation.* In the first step *serializer,* the input data which may be provided in various formats and multiple files, are parsed and stored in the format specified by PSAJ. Next, in the *transformer* step, the data stored in PSAJ can be transformed and prepared for the simulations. This could be, for instance, the balancing of an unbalanced electrical system or the automatic correction of missing or erroneous data. After all, the needed data has been serialized into PSAJ and has been prepared for the simulations, code for a simulation is generated in the last step *code generation.* These generated files can be directly opened by corresponding simulators. We implemented an exporter for GNS3, which prepares an entire GNS3 simulation and generates

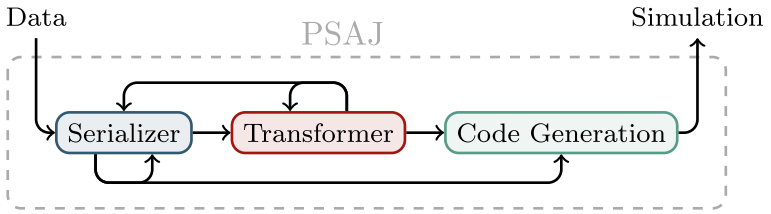

Fig. 4 Our envisioned usage of PSAJ

all the configuration files for the Docker containers. Additionally, Python scripts were written to generate PYPOWER input files. PYPOWER provides power flow (PF) and OPF algorithms which we use to validate the correctness of our proposed distributed OPF.

In PSAJ the top-level datatype has four main keys: `devices_electrical`, `devices_ict`, `connections_electrical`, and `connections_ict`. PSAJ differentiates between electrical and ICT networks which, in turn, are separated into devices and connections. Devices could be, for example, transformers, loads, or network switches, whereas connections are, among others, Ethernet cables or AC lines, used to interconnect two devices.

The openly-accessible data of the two IEEE test feeders are provided in Excel-sheets and Word-documents. Most of the data is in the Excel-Sheets except the properties of the different electrical line types. Their parameters are stored in the Word-documents. Figure 5 shows how we leverage PSAJ for the generation of the simulation files for the IEEE 13 and IEEE 123 node test feeders out of the provided documents. The steps are colored accordingly to the three PSAJ steps, which are shown in Fig. 4.

In the first step, the test feeder data from the Excel-Sheets and the electrical line data is serialized into PSAJ. Both IEEE feeders describe unbalanced three-phase grids. The distributed OPF is designed to solve single-phase problems, therefore, the data stored

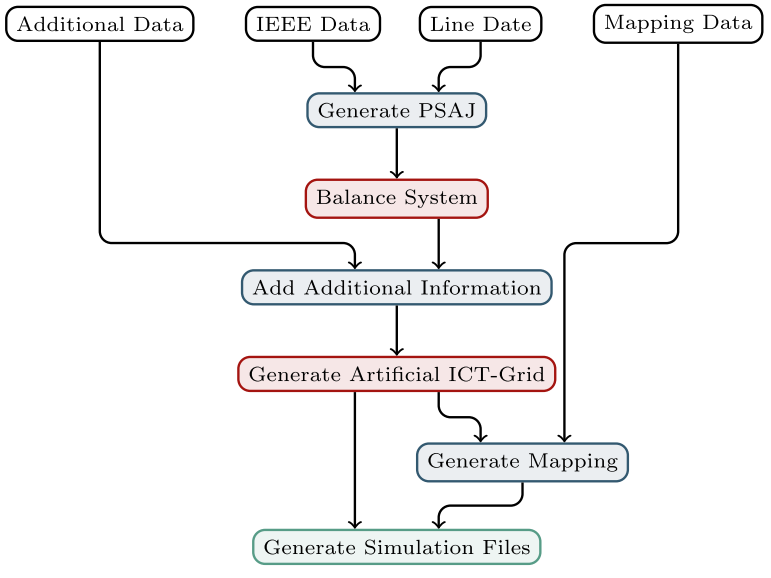

Fig. 5 This graph shows our usage of PSAJ for the generation of simulations for the IEEE 13 and IEEE 123 node test feeders

in PSAJ is transformed into a balanced single-phase system. This is done by using the positive sequence components and ignoring the terms which describe mutual components. Some information is missing in the test feeders, e.g., the breaker resistance. We add default values for these missing data after the balancing of the grid.

Since the IEEE test feeders contain only information about the electrical grid, we generate an artificial ICT network. Each bus gets an associated controller, which again is connected to daisy-chained switches. The data contained in the PSAJ file up to this point is general and contains all the topology of the electrical and ICT network, as well as their properties. However, the current data stored in PSAJ is not sufficient to generate the GNS3 simulation. Besides, the algorithms that should run on the controllers have to be specified. This is done by mapping Docker images to all the controllers which run the control algorithms. In the case of the distributed optimization, all the controllers use the same Docker image which was described earlier in "Implementation" section. Furthermore, the devices must be mapped to components that are available in the respective simulator.

In the final step, the code for the GNS3 simulation and the configuration files for the Docker containers are generated. Additionally, a case file for PYPOWER is generated, which is used for the validation of the distributed algorithm.

Simulation results

In this section, we show the simulation results from the distributed algorithm simulated in GNS3 for IEEE 13 and IEEE 123 node test feeders. Both test feeders model unbalanced distribution grids with a voltage of 4.15 kV. The IEEE 13 node test feeder has 13 buses, whereas the 123 node test feeder is a grid with 129 buses. In the original test feeders, the grids are connected via an on-load tap changer (OLTC) to a transmission grid. Furthermore, no generators exist, except, a slack node which models the power injection from the transmission grid. This is not a proper test case for the distributed OPF, as only a trivial solution for the power injection exists, which does not allow the minimization of active power losses. Therefore, in the IEEE 13 node test feeder, we replaced the loads on the buses 646 and 652 with generators, as shown in Fig. 6, and replaced 30% of the buses (39 buses) of the IEEE 123 node test feeder with generation buses. For both test grids, the distributed algorithm to solve the power loss problem was configured to run 300 iterations. The simulations have been run on a server with Debian 9 Stretch, with twelve cores running at 2.3GHz each, and 32GB RAM.

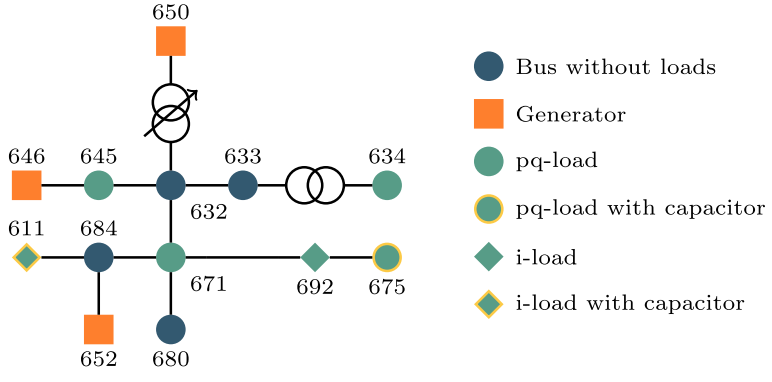

Fig. 6 IEEE 13 node test feeder

Convergence precision

First, we compare the results of the distributed optimization algorithm and their precision with the results obtained from PYPOWER. At the time of usage, PYPOWER did not support constant impedance and constant current loads. Therefore, we replaced these loads with constant power loads which consume the same power at the nominal voltage. To solve the power loss problem in PYPOWER, we set the linear cost coefficients of all the generators to the same value and the offset and quadratic coefficients to zero. For the comparison of the centralized and the distributed OPF, we compare the sum of the injected active powers and reactive powers of the generators.

The relative error for the distributed algorithm for the IEEE 13 node test feeder is 0.16% for the active power and -0.10% for the reactive power. For the larger grid, the IEEE 123 node test feeder, the relative error of the active power is 0.098% and -1.245% for the reactive power. The differences in the results can be due to local minima or approximations in the implementation. However, the solutions are comparable.

Step timing

This part describes insights into the timing of the algorithm on the IEEE 13 node test feeder, with disabled network filters. It represents the maximum achievable performance for the given platform and implementation. The total time until all controllers have finished their optimization is 46.91 s. This optimum scenario and the following results should be seen as upper bound which can be used to estimate the runtime of the distributed algorithm. As shown later, the CPU of the server is a limiting factor in the simulation because it has to run all the Docker containers and the GNS3 simulation in parallel. Therefore, the simulator cannot entirely utilize the fully distributed algorithm and slows down the approach.

Figure 7 shows statistics for all the controllers involved in the simulation. Boxplots are plotted for the controllers in the grid. The labels on the x-axis are the controller IDs. The IDs of the controllers correspond to the IDs of the buses in the IEEE test feeder, e.g., controller 611 is the controller which is associated with bus 611 of the electrical grid. The upper plot shows the time needed for the individual iterations. It is to be noted that the median of the iteration-times is nearly the same and does not depend on the number of buses in the subgrid. In each iteration, each bus depends on the results of its adjacent controllers. A controller can be with maximum one iteration in front or behind its neighbors. This heavily influences the scalability of these ADMM-based distributed algorithms, which involve consensus variables. The slowest participating controller dictates the speed of all controllers and slows them down.

The lower of the two plots of Fig. 7 shows statistics for the time which is needed in each iteration for the optimization without the consensus and update step. From left to right, the medians of optimization times tend to increase. The numbers under the curly brackets on the x-axis reflect the number of simulated buses considered by each controller in its optimization problem. Bus 611, for instance, has only one neighboring bus, whereas bus 671 has four adjacent buses. As can be seen in the plot, a direct dependency exists between the number of buses held by a controller and the time needed for optimization. This makes sense, as each controller k has an optimization problem with $2 \cdot n_k$ optimization variables (real and imaginary voltage parts for all the buses in the subgrid), where n_k is the number of buses in the subgrid. In contrast to the iteration times, the optimization times

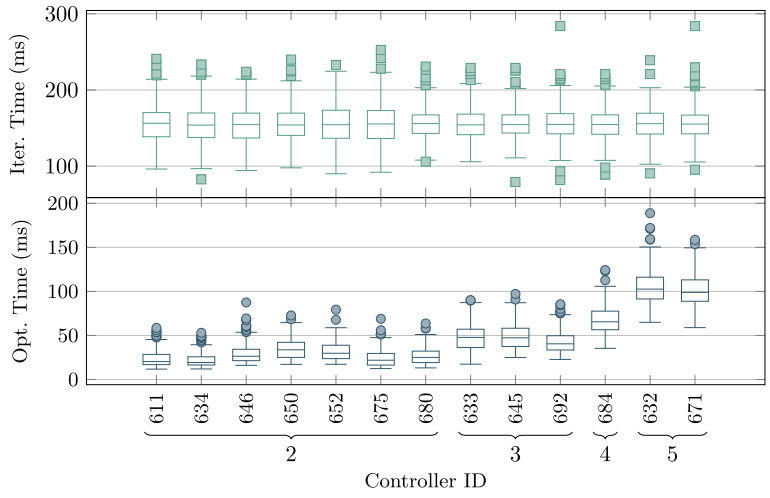

Fig. 7 The two plots show statistics for the time needed for optimization and the time needed for all the steps done in each iteration for all controllers of the IEEE 13 node test feeder

are not dictated by the slowest controller because they can run independently and do not have to wait on the inputs of the neighboring controllers.

Cold-start vs. warm-start

In the results previously shown, a so-called *cold-start* is performed. Thereby, the Lagrangian multipliers $\boldsymbol{\lambda}_k^{[0]}$ are all initialized with $\mathbf{0}$ for each bus k. Moreover, both the voltage vectors $\mathbf{v}_k^{[0]}$ and the consensus variables $\mathbf{z}_k^{[0]}$ are initialized with the value $\mathbf{v}_{\max,k}\angle\frac{\pi}{4}$ prior to the first iteration. In a *warm-start* scenario, in comparison, the results of the voltage and Lagrangian multipliers of a previous run are reused as initial points.

Figure 8 shows the different convergence behaviors of cold- and warm start for a constant pq-load connected to bus 675. In the first scenario, starting with iteration 0, a cold start is performed. The active power setpoint of the load with a precision of ± 0.025 pu is indicated with a rectangle shaded in blue, while the setpoint of the reactive power – which has the same precision – is shown with an orange area. Then, after iteration 300, a load step of 25% is applied for all loads in the IEEE 13 node test feeder. A second load step is performed after iteration 600, in which the loads are decreased to 75% of their initial demand. These two scenarios with the load steps are referred to as scenario 2 and 3. In both, a warm-start is performed because in each case they reuse the results of the previous scenario as a starting point. Indicated with dashed green lines, the iteration indexes C_1, C_2, and C_3 show the time of iterations that are needed in each scenario such that active and reactive power converges within the precision of ± 0.025 pu.

In the first scenario where a cold-start is performed, $C_1 = 203$ iterations are needed until convergence is reached at the shown load. In the case of the warm-starts, convergence is reached after $C_2 = 80$ iterations in the second scenario and $C_3 = 91$ iterations in the third scenario. This is a reduction of -60.59% when comparing the first and second scenario and -55.17% between the first and third scenario.

Table 1 shows the convergence results for all buses of the test feeder. The slowest bus needs 221 iterations in the first scenario until convergence is reached. In the second and

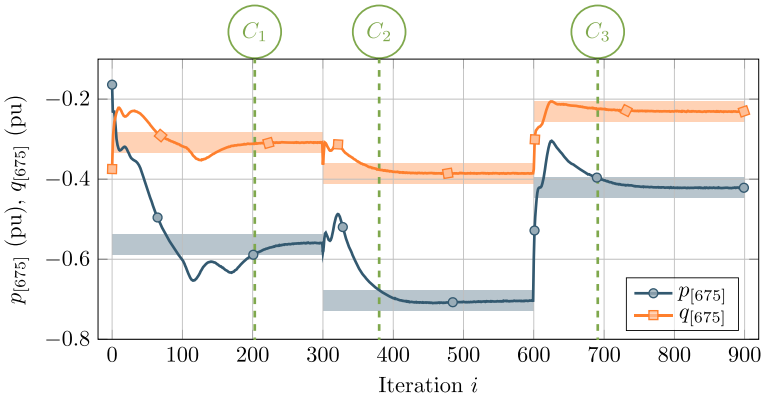

Fig. 8 This plot compares the behavior of cold- and warm-starts. In the first scenario starting with iteration 0, a cold-start is performed. In the other two scenarios, at iteration 300 and 600, load steps occur, while the algorithm is operated in a warm-start fashion, where voltages and Lagrangian multipliers of the previous scenario are reused. The iterations needed for convergence are indicated with dashed lines (green) for the three scenarios

third scenario 93 respectively 119 iterations are needed. These scenarios show the tendency that warm-starts converge faster than cold-starts. Furthermore, warm-starts are more realistic in practice because usually small load steps are performed in the power network. A scenario where the load demand is increased by 25% is already an extreme case. Cold-starts are performed after power outages, however, special procedures to reconnect networks after outages exist.

Relation of latency and runtime

In the following, we vary the filter latencies of the packet filters logarithmically from 0 ms to 80 ms for both IEEE test feeders and compare the results.

Table 2 presents the results for these simulations, where t_{lat} is the configured filter latency. The following metrics have been calculated for all controllers in the simulation, furthermore, the maximum value for each of these timings is displayed in the table. Column t_{f} shows the time needed by the slowest controller to finish the 300 iterations. In column t_{it}, the average time to finish one iteration is listed, while t_{opt} gives information about the average time needed for the optimization step. The last metric t_{comm} shows the average roundtrip latency for the selected filter, which is the time needed from sending a communication package until the acknowledgment is received.

The measured roundtrip latency t_{comm} is approximately four times the filter latency t_{lat} plus an offset. Each packet traverses two network filters during the transmission from

Table 1 This table shows the numbers of iterations which are needed to reach convergence for all buses in the IEEE 13 node test feeder. Three different scenarios are tested, where in scenario 1 a cold-start is performed and a warm-start is done in the second and third scenario

	Controller ID												
Scenario	**611**	**632**	**633**	**634**	**645**	**646**	**650**	**652**	**671**	**675**	**680**	**684**	**692**
1	165	178	117	9	70	8	11	119	221	203	37	218	220
2	62	68	38	1	21	1	1	1	93	80	19	93	93
3	58	76	49	2	29	1	1	66	119	91	5	113	115

Table 2 Statistics for the time needed for convergence t_f, the average iteration time t_f, the average optimization time t_{opt}, and the average roundtrip latency t_f for different network filter latencies t_{lat}

	IEEE 13				IEEE 123			
t_{lat} (ms)	t_f (s)	t_{it} (ms)	t_{opt} (ms)	t_{comm} (ms)	t_f (s)	t_{it} (ms)	t_{opt} (ms)	t_{comm} (ms)
0	46.91	156.37	102.75	3.70	189.36	631.21	366.75	28.70
1	52.50	175.01	96.20	6.99	198.98	663.26	340.30	32.68
2	60.44	201.48	91.15	10.54	198.02	660.07	328.40	31.26
5	95.07	316.91	86.60	23.09	215.91	719.69	302.40	40.83
10	154.79	515.95	77.05	44.79	251.13	837.10	245.35	59.04
20	282.17	940.58	74.80	88.86	364.48	1,214.93	151.60	109.83
50	709.39	2,364.64	77.35	236.27	785.42	2,618.05	97.35	275.77
80	1,164.00	3,879.99	72.65	390.51	1,237.52	4,125.08	87.50	414.13

the sender to the receiver, as well as when the acknowledgment is sent back. The additional offset is caused by the network switches and the overhead of the entire network simulation.

In contrast, the optimization time t_{opt} decreases with increasing filter latencies[1]. This relation can be explained by the fact that the CPU is the main bottleneck for use cases that involve small filter latencies. Increasing the latencies, most of the controllers idle after finishing their optimization problem and wait for arriving messages from adjacent controllers.

In Fig. 9 the relation between the configured filter latency on the x-axis and the runtime on the y-axis is shown in a double logarithmic plot. For small latencies, the time needed for convergence for the IEEE 123 node test feeder is significantly greater than for the IEEE 13 node test feeder. This can be explained by the different controller to CPU core ratios for the two test feeders. For the small feeder, this ratio is nearly one (13 controllers and 12 cores), whereas it is approximately ten (129 controllers and 12 cores) for the IEEE 123 node test feeder which introduces a computational overhead.

At the moment, the typical time frame for secondary control is 30 s after an ongoing frequency deviation (ENTSO-E 2009). None of the simulated scenarios converged within the 30 s time window within which the secondary control has to react. An alternative could be, however, to use a two-step secondary control. First, integral controllers bring the frequency back after a frequency deviation occurred. Afterwards, after the OPF-based second control algorithm converges, the new setpoints can be applied and the grid will be operated optimally from the power losses' perspective.

Moreover, the algorithm must converge, and the generators have to ramp-up within 15 min to comply with the timing constraint of tertiary control. Typically, tertiary control is activated after ongoing frequency deviations which last 15 min or longer, to free activated secondary control reserve (ENTSO-E 2009). This constraint is fulfilled by both test feeders when the roundtrip latency is smaller than ≈300 ms when the time for ramping-up the generators are not considered. It is important to emphasise that this and the following limits on the roundtrip latency have to be interpreted as upper bounds for the convergence of the distributed OPF. Due to the simulation environment and by using a single machine for the simulation, additional time is needed until convergence is reached.

[1] This arises since both the Docker containers and the GNS3 simulation are executed on the same machine, different from a production environment.

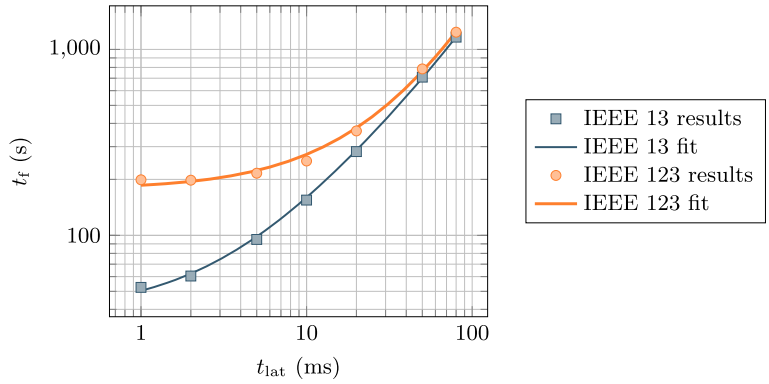

Fig. 9 The plot shows the time t_f needed for convergence with respect to the different communication latencies t_{lat}. Results for the IEEE 13 and IEEE 123 node test feeders are shown

When reserving 10 min for ramping-up the generators, maximum roundtrip latency of ≈70 ms[2] is allowed. Distributed generation based on power electronics, for example, batteries and PV can ramp-up in the sub-second range. However, to allow a smooth transition from centralized generation to distributed generation at the low and medium voltage level, the algorithm must also be able to accommodate centralized solutions based on large generators, for instance, thermal or hydropower plants. Such solutions cannot be automatically activated in all cases, which on one hand makes manual activations necessary and introduces delays of some minutes on the other hand.

Furthermore, it was previously shown, that around 100 iterations are sufficient to reach convergence. For the simulations shown in Fig. 9 cold-starts were performed, and 300 iterations were executed. In a warm-start scenario, only $\frac{1}{3}$ of iterations are needed. This results in a maximum roundtrip latency of ≈210 ms.

Table 3 lists average roundtrip latencies for wireless communication standards. Both, the roundtrip latencies of the 4G and the 5G standard, fall within the time frame of tertiary control and could, therefore, theoretically be used for such a control approach. The 2G, as well as 3G standards, are too slow to be usable for this distributed approach.

Conclusion

In this work, we presented a framework that allows the simulation of distributed control algorithms with a focus on the communication network. This framework is based on the GNS3 network simulator, which supports custom user applications via Docker containers.

To reduce the time needed for the generation and parametrization of the simulations, we introduced PSAJ, a methodology to automate the generation of simulations. This methodology allowed us the reconfiguration of all network cable latencies t_{lat} and penalty parameters ρ of each controller by just changing the parameters centrally. The code for the simulation is then generated automatically.

We used this framework to estimate the runtime behavior of a distributed OPF algorithm. Two use cases were investigated, namely the IEEE 13 and IEEE 123 node test feeders. Network filters were added to some network cables to model different

[2]This value is estimated by piecewise linear interpolations.

Table 3 Average data rate and roundtrip latency of wireless communication standards (Gavriluta et al. 2016)

Generation	Data rate	Roundtrip latency
2G	$100 - 400$ kbit s^{-1}	$300 - 1,000$ ms
3G	$0.5 - 5$ Mbit s^{-1}	$10 - 500$ ms
4G	$1 - 50$ Mbit s^{-1}	< 100 ms
5G	> 1 Gbit s^{-1}	1 ms

communication QoS. The analysis revealed that the distributed OPF under test – with no additional filter latencies – converges in 46.91 s. This value is considered marginally too slow for the current time frame of secondary control. However, the algorithm could be used in a two-step approach for secondary control or as a fast online tertiary control. The maximum, roundtrip delay of the communication network must be smaller than ≈210 ms such that the algorithm converges within the time frame of tertiary control. Typical time frames for tertiary control are 15 min after a frequency deviation. In the simulations 10 min were reserved for the ramp-up of the generators to allow a smooth transition from large centralized generators which need several minutes to ramp-up to distributed generation based on power-electronics which can ramp-up in the sub-second range. Thus, the algorithm had a maximum allowed runtime of 5 min. As shown in this paper, frameworks that have the capabilities to simulate the communication QoS can be successfully used to prove the usability of distributed control strategies for real-world scenarios.

While investigating the effects of deterministic constant communication latencies gives important insights into the performance of the ADMM-based distributed OPF examined in this paper, it might be interesting to study the influences of other network QoS aspects like packet loss and varying latencies. Furthermore, due to lack of available information, the network topology used in this paper was a simplified implementation made of daisy-chained switches. However, the proposed approach allows to model more complex and realistic network topologies. A more realistic model of the ICT layer would provide estimations for the timing of the distributed algorithm which are closer to reality. Moreover, the influences of different network topologies could also be investigated and compared.

About this supplement
This article has been published as part of *Energy Informatics Volume 3 Supplement 1, 2020: Proceedings of the 9th DACH+ Conference on Energy Informatics*. The full contents of the supplement are available online at https://energyinformatics.springeropen.com/articles/supplements/volume-3-supplement-1.

Authors' contributions
All authors contributed to the concept, contributed to the writing of the paper and they have read and approved the final manuscript.

Funding
Publication costs were covered by the DACH+ Energy Informatics Conference Organizers, supported by the Swiss Federal Office of Energy.

Availability of data and materials
The datasets used and/or analyzed during the current study are available from the corresponding author on reasonable request.

Competing interests
The authors declare that they have no competing interests.

Author details
[1]AIT Austrian Institute of Technology – Electric Energy Systems, Giefinggasse 2, 1210 Vienna, Austria. [2]TU Wien – Institute of Computer Technology, Gußhausstraße 27-29, 1040 Vienna, Austria. [3]Sprecher Automation GmbH – Product Management, Ignaz-Köck-Straße 10, 1210 Vienna, Austria. [4]Danube University Krems – Center for Integrated Sensor Systems, Viktor Kaplan-Straße, 2700 Wiener Neustadt, Austria.

Published: 28 October 2020

References

Andrén FP, Strasser TI, Kastner W (2017) Engineering smart grids: Applying model-driven development from use case design to deployment. Energies 10(3)
Boyd S, Parikh N, Chu E, Peleato B, Eckstein J, et al (2011) Distributed optimization and statistical learning via the alternating direction method of multipliers. Found Trends® Mach Learn 3(1):1–122
ENTSO-E (2009) P1-policy 1: Load-frequency control and performance. Technical report, ENTSO-E. Available at https://www.entsoe.eu. Accessed 12 June 2020
Erseghe T (2014) Distributed optimal power flow using admm. IEEE Trans Power Syst 29(5):2370–2380
European Environment Agency (2018) Renewable energy in Europe – 2018. Available at https://www.eea.europa.eu. Accessed 12 June 2020
Gavriluta C, Boudinet C, Kupzog F, Gomez-Exposito A, Caire R (2020) Cyber-physical framework for emulating distributed control systems in smart grids. Int J Electr Power Energy Syst 114:105375
Gavriluta C, Caire R, Gomez-Exposito A, Hadjsaid N (2016) A distributed approach for opf-based secondary control of mtdc systems. IEEE Trans Smart Grid 9(4):2843–2851
Guo J, Hug G, Tonguz OK (2017) On the role of communications plane in distributed optimization of power systems. IEEE Trans Indust Inform 14(7):2903–2913
Kraning M, Chu E, Lavaei J, Boyd S, et al (2014) Dynamic network energy management via proximal message passing. Found Trends® Optim 1(2):73–126
Ling Q, Liu Y, Shi W, Tian Z (2016) Weighted admm for fast decentralized network optimization. IEEE Trans Signal Process 64(22):5930–5942
Molzahn DK, Dörfler F, Sandberg H, Low SH, Chakrabarti S, Baldick R, Lavaei J (2017) A survey of distributed optimization and control algorithms for electric power systems. IEEE Trans Smart Grid 8(6):2941–2962
Xu J, Sun H, Dent CJ (2018) Admm-based distributed opf problem meets stochastic communication delay. IEEE Trans Smart Grid 10(5):5046–5056
Zhang Y, Hong M, Dall'Anese E, Dhople S, Xu Z (2017) Distributed controllers seeking ac optimal power flow solutions using admm. Electr Power Syst Res 9(5):4525–4537
Zhang M, Kar RS, Miao Z, Fan L (2019) New auxiliary variable-based admm for nonconvex ac opf. Electr Power Syst Res 174:105867

Publisher's Note