Gossip protocol approach for a decentralized energy market with OPC UA client-server communication

Josef Schindler[1], Asmaa Tellabi[2] and Karl Waedt[3]

Abstract: Gossiping is a well-researched protocol that enables decentralized information sharing. Being comparable to viruses spreading in a biological population, such concepts of data sharing are also called epidemic protocol. Without wanting to be impious with respect to recent pandemics, we propose its usage to facilitate a peer-to-peer (P2P) market for sharing energy between flexible loads or generation units, respectively. Gossip algorithms have been proposed several times in the context of power sharing in transmission grids. Our main contribution is the integration of such scenario with OPC UA. Comprising security by design, good interoperability attributes, several, well-maintained stack implementations and a widespread usage in automation, it reveals to be an outstanding framework for the proposed use case that will be explained in the first sections. After describing underlying physical models and the setup scenario, we will compare the results of the scenario that was conducted on non-OPC UA modules and an OPC UA implementation. Mostly, the performance is questioned at the comparison, still some beneficial concepts of OPC UA can be highlighted in the conclusion: Security controls can be added to the system at the Application Layer where Attribute Based Access Control (ABAC) can be performed, which allows a fine granularity of privileges expressed for subjects (agents in the gossiping algorithms) and objects (energy related assets) via semi-formal security policies. Additionally, UA Discovery service allows for plug and play availability. Concluding, a framework for a very efficient large-area algorithm is presented here to be researched in further work.

Keywords: gossip, peer-to-peer, decentralized energy market, OPC UA

1 Introduction

In this section, we provide basic information and a literature overview, first on the implemented gossip protocol or algorithms in general (section 0), second on the used communication protocol, namely OPC UA (section 1.2). Third a literature overview is given on energy markets, with a strong focus on former use-cases of gossiping algorithms (section 1.3). In section 2 the consensus finding between two peers (section 2.1), as well as the formal description of the cost functions (section 2.2) for both – in section 3 described – implementations is given. Their test results are compared in section 4, before a short conclusion with an outlook is given in section 5.

[1] Friedrich-Alexander-Universität Erlangen-Nürnberg, Faculty of Engineering, Institute of Electrical Energy, Systems Cauerstr. 4, Erlangen, 91058, josef.s.schindler@fau.de
[2] University of Siegen, Faculty of Science and Engineering, Chair for Data Communication Systems, Hölderlinstraße 3, Siegen, 57068, asmaa.tellabi@uni-siegen.de
[3] Framatome GmbH, Erlangen, Germany, karl.waedt@framatome.com

1.1 Gossip Protocol

Gossip algorithms are part of the wide class of distributed algorithms. They were created based on the concept of epidemic. The creation of such a class was a founded on the spreading of virus among populations, based on the idea that one source can infect more entities which subsequently become sources themselves [Kr11]. For computer systems, the notion of viruses is replaced by information that should be exchanged with other nodes in a network. During each communication round, nodes choose and exchange information with one or more of their surrounding nodes. Before the next round begins, the information is managed locally and modified according to measurements taken locally in order to modify the information value that has to be exchanged [KH15].

Fig. 1 depicts the basic concept of the information spreading inside a simple network. The nodes with the information are shown in orange. In each round they pick a random neighbour to hand the information over to him (compare vectors). The more the information is spread, the more it is likely to address a node that already has the information, as seen, when the vectors are pointing to an orange node.

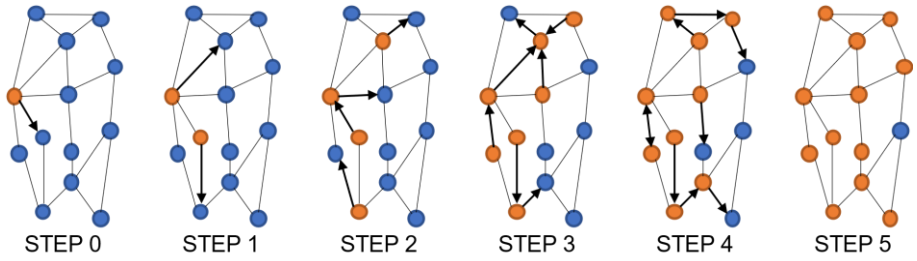

Fig. 1: Basic concept of gossiping algorithms with 5 rounds of information spreading

Gossip protocols are well-known in the research field especially in the computer science areas. They are a promising method that can be used to manage technical issues that occur in distributed systems since they are easy to implement and are able to detect and react to failures rapidly. In a basic gossip protocol implementation, each node chooses a partner randomly to send its current observed state [Al07]. Recently, more research on gossip protocols have been conducted especially for information processing in sensor networks [Di10]. In many distributed computer systems, such as cloud computing and peer-to-peer (P2P) computing systems, the rapidity and robustness characteristics that gossip protocols provide as well as their simple concepts and the absence of central management make this protocol an interesting solution for future implementations. Gossip algorithms are useful for distributed systems for the following reasons [Je11]:

- Creating new protocols: gossip protocols are easy to implement, quick and robust. These characteristics are useful for sharing and collecting information as well as for their processing. These tasks are essential for distributed systems.
- Advanced research about security attacks: With the growing use of the Internet, cybersecurity attacks are becoming more and more sophisticated. Usually, advanced malicious attacks use infected computers that are grouped into networks known as botnets, which are capable of performing advanced and coordinated cyber-attacks. Gossip protocols are used to understand and simulate such attacks in order to find more advanced and performing countermeasures.

At first, gossip algorithms were applied for the quick duplication of database updates. Conversely, they also tend to be seen as a potential application for data collection and the processing of information in sensor networks. After that, they were used for solving peer sampling issues, group and topology management, fault detection and other rising technical issues in communication. On the other hand, they can be implemented in other areas where information has to be diffused in distributed computation systems. This was the motivation behind new research dedicated to the implementation of gossip algorithms in future power systems and smart grids [KH15].

They are based on a periodic exchange of information, during each communication round, nodes communicate with other peer nodes. A communication topology that is based on the gossip algorithm provides several functions like information exchange, collection of information and topology construction [BUV12]. Some of the key characteristics of the gossip algorithms are the following [Kr11]:

- The communication is done in pairs only, so one node communicates with only one other node at a defined time and several pairs can exchange information simultaneously.
- The gossiping partner is selected randomly.
- Before sharing the information, every node has only knowledge about the local system state.
- Nodes that are available in the system run on a similar protocol.
- The transmission and processing capacity per communication round is limited.
- Such networks are scalable and are integrated in dynamic models. More details about gossip algorithms properties can be found in [Kr11].

Three types of gossip algorithms – random, broadcast and geography – are presented in [Mi18] and their behaviour is sketched in Fig. 2.

In random gossip protocols, a node chooses randomly a direct neighboured node to share information. Whereas, geography gossiping is not limited to one hop. In the third type, the broadcast gossip, a node spreads its information to all of its neighbours. Fig. 2 shows the

node with the information to be spread in orange, all possible targets in green and the path to one (random, broadcast) or several (broadcast) targets with vectors. The approach in this paper uses geography gossip style, i.e. each node can request any other node of the network (compare section 3)

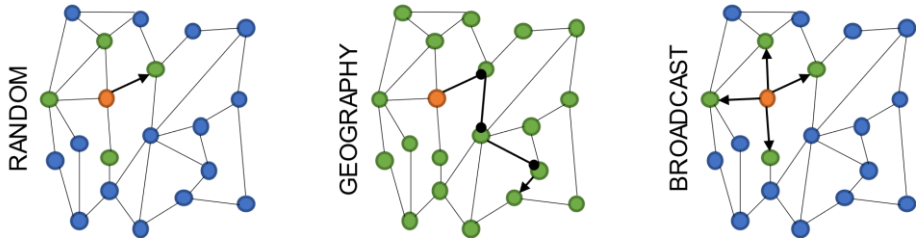

Fig. 2: Different types of gossiping protocols - random, geography and broadcast

1.2 OPC UA

OPC UA is a standard used for exchanging data between industrial systems. In 2011, OPC UA has been standardized by IEC as IEC 62541, which guarantees a better understanding and precision in the specification and an increasing acceptance and implementation within the industry. In addition, OPC UA takes into consideration the particular requirements related to each industry needs within its specifications and uses its extensible information modelling framework to create new industry specific information models. OPC UA provides information modelling that includes security features built in the design, it offers access rights controls, as well as the communication stack to deliver plug-and-play, machine-to-machine communication for plants and factories [SP20].

In [OP18], it is expected that the usage of OPC technologies will increase nearly by 45 percent annually within the next five years. This increase of usage can be explained by the integration of OPC UA technology within smart automation devices and into new market segments like traffic signal systems in Germany. OPC UA can grow exponentially outside its main market segment, which is the automation market in different other markets, counting building automation, medical, telecom, data centres, warehouse, and transportation [OP18].

Message Queue Telemetry Transport (MQTT) is another promising communication protocol that is founded on a subscription-publishing concept, where publishers send messages to a server which is responsible of forwarding messages to subscribers in order to avoid point-to-point connections between subscribers and publishers. The subscribers do not necessarily need to know from where the information came, and which entity is subscribed. MQTT is created to be integrated in low-cost devices with limited resources, which have low-bandwidth networks and high latency [MU20]. The Constrained Application Protocol (CoAP) is a protocol dedicated to IoT architectures that is defined in RFC 7252. CoAP is a protocol with low overhead that is dedicated to resources constrained devices, such as microcontrollers, and networks. This protocol is used in M2M

data exchange and has multiple similarities to HTTP [DZ20]. OPC UA is a wide architecture where the communication protocol is only a small part of it. An application that used OPC UA can see all network nodes, methods, and data structures. As stated before, MQTT and CoAP were designed to be lightweight and for smaller systems, which is not the case in industrial systems. Therefore, OPC UA was chosen for this implementation. OPC UA architecture already includes services such as UA Discovery, Pub/Sub, it also integrates security services in its specification and design.

A major challenge of Industry 4.0 and the industrial Internet of Things (IIoT) is to provide a secure, standardized data exchange between devices, machines, and services, coming from diverse industries [Mu20a]. In April 2015, the Reference Architecture Model for Industry 4.0 (RAMI 4.0) has recommended the IEC 62541 standard related to OPC UA [Fr13] as the only communication stack to implement the communication services. One of the requirements needed to integrate OPC UA inside the industrial 4.0 communication layers is a network based on the Internet Protocol (IP). The data exchange in OPC UA is based on 2 different mechanisms:

- A client-server model in which OPC UA clients use the services an OPC UA Server is providing.
- A publisher-subscriber model in which an OPC UA Server makes some information available to a certain number of receivers.

The OPC UA framework is suitable for small devices as well as to technologies used at the IT level such as the cloud computing. In 2012, the Fraunhofer Institute Lemgo reduced the size of the OPC UA Server to a 10 kB footprint, so it can be implemented into small size sensors. Framatome GmbH is one of the multiple companies who choose to integrate the OPC UA Server into the sensor of monitoring devices for valves as well as their electric actuators. Framatome GmbH uses this solution in the nuclear industry to monitor distributed critical systems (without connection to the internet). For data reliability, built-in security features and interoperability aspects forming the essential parts behind the success of OPC UA technology see [SP20].

Since 2008, Beckhoff and Siemens supported the integration of the OPC UA standard and its implementation into their products. Currently, nearly all Programmable Logic Controller, visualization and Manufacturing Execution System-manufacturer support this standard. The OPC UA specifications were published as multipart standards since 2015 and the OPC communication stack was available as an open Source library for evaluation purposes in 2016. For the professional integration of OPC UA into specific industrial use-cases, some companies provide software toolkits and consulting services for platform and product specific implementations [SP20].

Besides commercial stacks, there are various open source implementations that are compatible with a different range of OS, written in different programming languages and offering a different level of functional coverage of OPC UA via services and features. In [Mu20a] and [Mu20b], authors provided a comparison of the most famous open-source

OPC UA implementations, namely, open62541, node-opcua, UA-.NETStandard and FreeOPCUA. One of them, the FreeOPCUA library is used to implement OPC UA clients and servers in this paper.

A major basis for the Industry 4.0 model is interoperability between previous systems with newer technologies [Sc19]. Industrial communication protocols have been developed since the 1980s and currently OPC UA is considered as the main communication protocol that is used for monitoring and controlling purposed in industrial processes [OP10]. OPC UA can be implemented in order to connect industrial controllers in factories to the Internet.

OPC UA can be integrated on platforms that are running on multiple General Purposes Operating Systems (GPOS) such as Windows and Linux. The OPC UA protocol specification is based on the concepts of Address Space Model [OP18] and the services. The Address Spaces specify how objects between servers and clients are going to be represented. An OPC UA Object contains some variables and methods that are implemented as nodes which belong to different node classes inside the Address Model.

1.3 Energy markets

Energy production has developed throughout the last decades, especially in Germany, where windmills and photovoltaic power plants have a share of about 50% nowadays compared to 5% in the beginning of this millennium. The entry of such volatile energy sources makes the market more fluctuating and decentralized. To overcome the higher demand of control due to the rising number of stakeholders in the market system, many hierarchical approaches have been proposed, e.g. smart-grids, smart-neighbourhoods, virtual power plants.

A concept of cross-commodity sharing such as electricity or heat, is proposed in the DECENT project. Herein, a centralized market on a neighbourhood level uses merit order on a 15 minute-base to facilitate the commodity-sharing. The consensus is then stored using Blockchain technology, so that the agreed trades are reliable and not negligible.

A notable number of scientists has also considered gossiping algorithms for the use in, and for the control and measure of smart grids. Gossiping algorithms were introduced first by [Kr11] for information dissemination in power systems. Furthermore, the authors state the requirements on ICT infrastructure to enable actual implementations. In [Cr17], a gossip-based scheme is used to control and to facilitate a distributed demand response in a virtual microgrid, i.e. the university campus. Checks on the applicability of flow updating for a large number of network nodes is done with a P2P network simulator. Also, [EAD16] proposes a P2P control and gossiping communication for the similar purpose of keeping the voltage of a smart grid within the boundaries. Surplus on that approach is seen in keeping all control local, i.e. leaving a central controller out, which increases the fault tolerance.

In [KH15] and [KH16], the authors propose a gossip-based approach to detect and resolve voltage violations, and to manage demand and thus power flows on radial distribution grids, respectively. Authors in [Le16] and [Mi18] have also used gossip algorithms for power sharing purposes. In two papers [Wa19] and [Wa20], R. Wang et al. first propose a gossip-based algorithm for the economic dispatch in a power grid with transmission losses and secondly research random communication link failures within the same setup. By modelling the system as a Bernoulli network in [Wa20], they try to overcome the communication link failures and make the algorithm able to solve the mathematical problem, i.e. the economic dispatch.

During the last years, even the car manufacturer Tesla launched its own so-called "Autobidder" software. Several forecasting mechanisms and a dispatch optimization are performed to autonomously monetizes battery assets. According the statement [TE20], the software is scalable from a single household prosumer up to an the utility level. Such large company working in that domain shows a valid potential in that market section.

In this paper, we will present an approach where we have merged the DECENT idea of having a local market, with the discussed solutions for power sharing using gossip protocols. The result will be an algorithm that finds the market consensus on a neighbourhood-level. Furthermore, we compare an OPC UA stack with a basic implementation to measure their performances and to evaluate beneficial concepts of this communication standard.

2 Consensus finding

The system consists of several flexible prosumers, which are all connected electrically and communicate with each other. In this paper, only the communication network was implemented, with the assumption that the power network is a copper plate with no limitations and packet losses. Additionally, all participants have the same target – to assure the common welfare by producing power at the least overall- or community-cost.

Therefore, every flexible prosumer or entity has its own cost function, i.e. at which price can it produce which amount of energy. The entity runs a server and is continuously listening to requests from other entities to find their common optimum setpoint. If no request is upcoming at the server, the entity tries to request another, randomly chosen entity for a settlement from time to time. Therefore, it sets up an own client.

2.1 Entity-to-entity consensus

The overall process for consensus finding is depicted in Fig. 3 for entity A which sends a request to entity B. To avoid issues with parallel truths, the entity – which is represented by Client A – requests to block its server (see step 1 & 2). If Server A were blocked due to another request the process would stop immediately.

Next, Client A reaches out to Server B to authenticate and establish a connection (see steps 3 to 7). Again, the process stops here if Server B were blocked or if the authentication failed. After a connection is established, Client A sends the cost function details to Server B (step 8). At step 9, Server B calculates the optimum setpoints and sets its own, new power production. The new setpoint A is resent to entity A, communicated to Server A and confirmation about that finalizes the process in steps 10, 11 and 12, respectively.

The optimization calculations in step 9 rely on polynomials, which can be noted in the standard form and a form showing the polynomial roots

$$c(p) = \sum_{i=0}^{n} a_i \cdot p^i = b_n \cdot \prod_{i=0}^{n-1}(p - b_i). \quad (1)$$

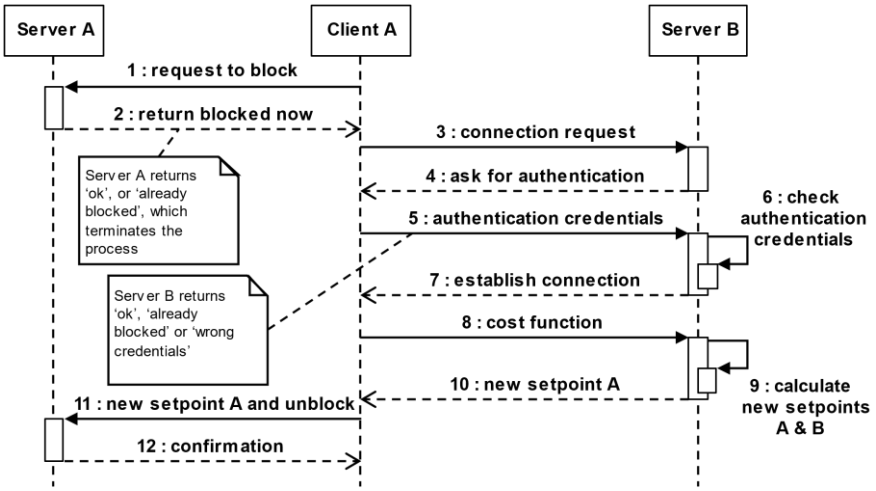

Fig. 3: Communication between OPC UA Server and Client based on the gossiping algorithm

Hence, all necessary coefficients a_i, the setpoint p_0 and limitations p_{min} & p_{max} of entity A are sent in step 8 and will be named $c_A(p)$, whereas the cost function of entity B is named $c_B(p)$. To gather comparable functions, not the absolute power output p, but the relative change of the setpoint Δp is considered for both polynomials. Therefore, both functions are shifted by p_0:

$$b_i' = \begin{cases} b_i - p_0, & i < n \\ b_i, & i = n, \end{cases} \quad (2)$$

Returning the shifted functions $c_{A,s}(p)$ and $c_{B,s}(p)$. If entity B raises its power output, entity A would need to lower its output to satisfy the system's demands. Therefore, entity A's polynomial is mirrored at the why axis

$$c_{A,s,m}(\Delta p) = c_{A,s}(-\Delta p). \quad (3)$$

Afterwards, both functions can be added together:

$$c(\Delta x) = c_{A,sh,m}(\Delta p) + c_{B,sh}(\Delta p) \quad (4)$$

To receive an overall function whose optimum can be found. Differentiation of the polynomial from equation (4) returns all possible optima for Δp, from which all values within the boundaries $[p_{min}, p_{max}]$ and the boundaries themselves need to be considered.

2.2 Manifestation of the cost functions

The cost functions resemble the marginal cost to produce a certain amount of power and are specific to the type of the parent entity. To give a clearer vision, Fig. 4 depicts several schematic graphs, how they could look like for real applications. All graphs have been normalised, so the nominal power output equals 1. The first graph belongs to a photovoltaic power plant. Obviously, the output is limited to 0 and a maximum power that depends on the current solar radiation and thus is usually smaller than the nominal power. In between those boundaries, no significant price variation can be observed.

Secondly, a battery storage has a nearly rotatory symmetric shape, but is slightly shifted on both axes e.g. due to losses in power electronics and wear of the cells. Of course, a battery storage even wears out when it does not produce energy – compare calendric degradation. Again, this is a schematic overview. In a real application, a battery's function would be very complex and will depend on the state of charge or health, respectively.

The third graph in Fig. 4 belongs to a fuel-based generator, like a combined heat and power plant or a diesel generator. Considering only the usage of fuel would lead to a linear characteristic. Anyway, bad efficiency in low and greater wear on components in high power domains lead to higher gradients.

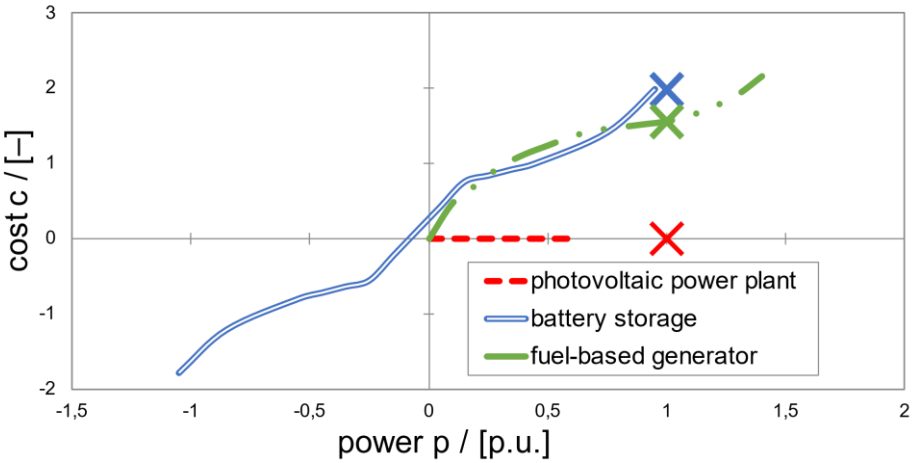

Fig. 4: Example schemes of cost functions of several possible entities

One entity can be a single specific flexible producer or consumer, but also a composition

of several of them. So, in the virtual setup all parameters are randomly chosen at the beginning and then kept constant, to obtain a static system.

3 Implementation

In this paper two python-based setups with different approaches are compared. One uses a basic client and server (see section 3.1), whereas the other uses the library FreeOPCUA, i.e. OPC UA Server and Clients (section 3.2). Here is a list of the common specification

- Each entity is simulated on its own virtual machine inside one testing computer. Hence, the number of entities is limited to 24 according to the RAM of the computer
- Each entity/virtual machine has a specific IP address and a randomly and uniquely prepared cost function; the setup is given in Tab. 1
- Each entity is aware of the present entities from a locally stored setup file. In a real-life implementation, this could be facilitated via OPC UA Discovery Service or as a side effect of the gossiping protocol, i.e. during consensus finding, every entity hands over the location of other known entities
- The delay for one entity to actively request another entity for consensus finding (compare Fig. 3) is chosen randomly within a range d_{min} and d_{max}
- The topology of the gossiping network is not taken into account; in a real-life scenario, this might be an issue, especially when the power grid needs to be considered like in [KH16]

	Entity with IPv4 Addresses: 192.168.56.1<ID>																							
<ID>	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
p_{init}	0.4	0.5	0.6	0.6	0.4	0.6	0.7	0.8	0.9	0.4	0.0	0.3	2.6	1.3	-1.0	0.8	4.3	0.3	0.2	3.3	4.3	-1.0	1.5	-1.0
c_{init}	2.58	1.88	4.12	4.12	3.38	1.82	4.04	4.28	2.03	0.31	3.0	0.2	4.52	-2.45	1.9	-2.21	-1.35	0.69	3.51	1.82	3.3	1.8	3.04	-1.1
p_{min}	-1.1	0.0	-2.0	-2.0	-1.5	0.6	-2.0	-2.8	0.1	-2.2	-1.0	-1.4	-1.4	-2.4	-2.2	-1.3	0.0	-2.2	-1.4	0.0	1.0	-2.0	-1.4	-1.4
p_{max}	1.9	3.0	3.0	3.0	1.5	6.0	1.7	1.8	2.2	1.3	1.0	1.2	8.4	1.4	1.1	0.0	5.0	1.1	1.4	5.0	7.0	3.1	1.6	0.3
a_0	1.0	2.0	4.0	4.0	1.0	2.0	4.0	4.0	2.0	0.5	3.0	0.2	4.0	-5.3	0.9	0.2	-3.1	0.6	3.2	0.3	-1.0	0.2	1.1	0.6
a_1	3.0	-0.5	-0.3	-0.3	5.0	-0.6	-0.7	-0.8	-1.2	-0.4	2.5	0.2	0.2	1.6	-0.8	-5.3	2.3	0.1	1.6	-0.2	1.0	-1.2	0.6	0.3
a_2	2.0	0.5	0.6	0.6	2.0	0.5	0.6	0.8	1.1	-0.3	-5.2	-0.8		0.38	1.0	1.1	-1.3	0.5	-0.3	0.2	0.0	0.6	0.8	-1.1
a_3	1.0		0.4	0.4	1.0		0.7	0.8	0.3	0.2	3.1	0.4		0.06	0.8	2.2	0.2	0.7	0.2			0.2	-1.2	0.5
a_4										0.1		0.2						0.1					0.2	0.2
a_5												-0.1											0.3	
a_6												0.1												

Tab. 1: setup overview; normalized parameters of all entities

3.1 Basic implementation

The basic implementation uses `socket` from the standard python library. A server is created inside the entities code. While listening, it will block the execution of any other task and is therefore interrupted by `settimeout`. Then a client tries to connect to a

randomly chosen server. Every connection is accepted, so no authentication (compare step 4 up to 6 in Fig. 3) is used. The sending buffer needs encoded data. So, a list containing the current setpoint, boundaries and cost function coefficients is packed and sent from client to server. Vice versa, the server returns a binary encoded new setpoint.

3.2 OPC UA implementation

The second implementation has a permanently running OPC UA Server at each entity. The python library `sched` is used for timing, i.e. to set the OPC UA Client up, whenever it is time to request a P2P consensus from a randomly chosen other entity.

The proposed authentication was not implemented so far but can be updated in future versions. OPC UA's specification enables such features, independent on the library used for the implementation. Another example on this, is the method call used for steps 8-10 in Fig. 3. UA Namespaces in OPC UA Servers provide all necessary information to OPC UA Clients, so they can browse through it and call certain methods, independent on the programming language or stack provider.

4 Simulation results

Both setups described in section 3 have been simulated several (in overall 72) times, varying the number of entities $N = \{8, 16, 24\}$ and the delay for the next outgoing request d_{min} and d_{max}. Each run was finished after 90 seconds. An exemplary cost development of each entity and their sum is depicted in Fig. 5.

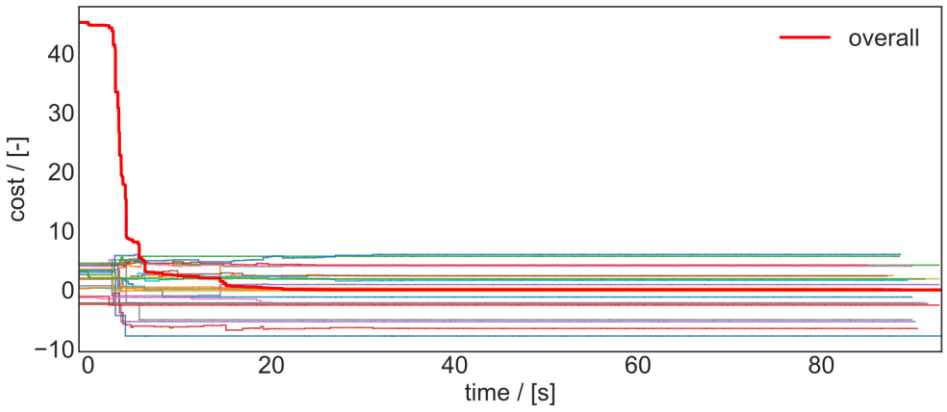

Fig. 5: cost development during one gossip simulation, with: $N = 24$, $d_{min} = 0.01$sec, $d_{max} = 0.3$sec, Run = 3

As a measurable result, the time span until the final overall cost are fitted up to 99% t_{99}

and the final overall cost c_{end} itself are displayed for each run in Tab. 2. Each testing scenario is processed three times to cover for statistical deviations, the mean values are given in overview Tab. 2.

A pretty good assumption for an optimum can be calculated very efficiently using gossip protocols, but there is no clear statement, whether it is a local or the global optimum. In this simulation set, the three optima at $c_{end} = \{13.953, 15.546, 16.343\}$ for $N = 8$ entities prove this behaviour. Neither the basic nor the OPC UA implementation achieved the better optimum more preferably.

d_{min}; d_{max}			0.01 sec; 0.3 sec				0.5 sec; 1.0 sec				1.0 sec; 1.5 sec				2.0 sec; 3.0 sec			
Type			basic		OPC UA		basic		OPC UA		basic		OPC UA		basic		OPC UA	
N	c_{init}	Run	t_{99}	c_{end}	t_{99}	c_{end}	t_{99}	c_{end}	t_{99}	c_{end}	t_{99}	c_{end}	t_{99}	c_{end}	t_{99}	c_{end}	t_{99}	c_{end}
8	26.23	1	3.881	16.340	1.121	16.343	4.275	16.343	4.406	13.953	3.496	16.343	6.163	16.343	8.128	16.343	6.032	16.343
		2	2.453	16.343	0.746	16.343	3.649	13.953	2.189	16.343	10.173	13.953	7.175	13.953	7.133	16.343	9.772	13.953
		3	2.342	13.953	1.756	16.343	2.341	16.343	3.312	16.343	7.985	13.953	11.925	16.343	7.782	16.343	19.370	13.953
		Ø	2.892	15.545	1.208	16.343	3.422	15.546	3.302	15.546	7.218	14.750	8.421	15.546	7.681	16.343	11.725	14.750
16	33.53	1	6.754	-2.919	7.225	-2.918	21.605	-2.918	10.940	-2.918	15.686	-2.918	22.971	-2.918	46.042	-2.918	32.889	-2.918
		2	7.081	-2.946	4.487	-2.918	16.779	-2.918	9.042	-2.918	18.532	-2.918	10.754	-2.918	29.510	-2.918	49.206	-2.918
		3	3.711	-2.918	6.047	-2.918	15.615	-2.922	14.719	-2.918	15.687	-2.918	18.104	-2.918	25.971	-2.918	23.767	-2.918
		Ø	5.849	-2.928	5.920	-2.918	17.999	-2.920	11.567	-2.918	16.635	-2.918	17.277	-2.918	33.841	-2.918	35.287	-2.918
24	45.25	1	35.316	-2.209	14.789	-0.061	51.898	-1.084	22.816	0.001	33.859	1.123	7.706	0.469	31.957	-0.042	13.756	0.001
		2	56.017	-1.003	8.152	0.001	5.125	-0.025	8.068	0.469	74.912	-2.479	17.406	0.469	23.103	0.020	28.636	0.016
		3	55.417	-2.503	18.829	0.001	63.858	-0.869	14.640	0.001	80.705	-16.86	16.864	0.001	50.545	0.602	54.278	0.008
		Ø	48.917	-1.905	13.923	-0.020	40.294	-0.659	15.175	0.157	63.159	-6.074	13.992	0.313	35.202	0.193	32.223	0.008

Tab. 2: Simulation results from gossiping with varying number of entities, delay, and concept

Still, at the basic implementation some errors or package losses might have occurred. The first run with $N = 8$ entities and the fastest delay beats both probable optima slightly. Also, when looking at the basic implementation's results for $N = 24$ entities, no clear optimum is visible. There are two possible reasons. First, the overall optimisation function is too complex, so it has that many optima. Second and more likely: the unreliable network connection in the basic implementation leads to unfinished consensus; Consequently, the overall initial power demand p_{init} is violated because requested Server B sets the new consensus setpoint, but entity A does not – no reliable check-up has been specified after step 10 (see Fig. 3). Tab. 3 proves that effect by plotting the summarized power output of all entities at the initial p_{init} and final state p_{end}, respectively.

In comparison and still considering $N = 24$ entities, OPC UA shows local optima at $c_{end} = \{0.001, 0.469\}$. They occur at least three times during the simulations. Also, Tab. 3 shows the reliable behaviour in that context. Except one outlining run, no power deviations can be observed between initial and final state.

Despite OPC UA has a fully developed stack on the backend, which could slow down the application, it shows to be equally fast for $N = \{8, 16\}$ entities, and even faster for $N = 24$ entities compared to the basic implementation. Another expected phenomenon only partly occurred: even for the largest set of entities the single computer did not go very slow for the shortest delay at the OPC UA implementation, due to overload of the machine.

In contrast, for the basic implementation it did. The time to get close to the consensus decreases when looking at "slower" parameters, i.e. higher delays.

d_{min}; d_{max}			0.01 sec; 0.3 sec		0.5 sec; 1.0 sec		1.0 sec; 1.5 sec		2.0 sec; 3.0 sec	
Type			basic	OPC UA	basic	OPC UA	Type	basic	OPC UA	basic
N	p_{init}	Run	p_{end}	p_{end}	p_{end}	p_{end}	p_{end}	p_{end}	p_{end}	p_{end}
8	4.6	1	4.595	4.6	4.603	4.6	4.6	4.6	4.6	4.6
		2	4.601	4.6	4.6	4.6	4.6	4.6	4.6	4.6
		3	4.599	4.6	4.6	4.6	4.6	4.6	4.6	4.6
		SD	0.003	0.0	0.002	0.0	0.0	0.0	0.0	0.0
16	9.9	1	9.902	9.9	9.9	9.9	9.9	9.9	9.9	9.9
		2	10.055	9.9	9.9	9.9	9.9	9.9	9.9	9.9
		3	9.9	9.9	9.901	9.9	9.9	9.9	9.9	9.9
		SD	0.09	0.0	0.001	0.0	0.0	0.0	0.0	0.0
24	21.8	1	21.769	21.892	22.415	21.8	19.324	21.8	22.243	21.8
		2	23.004	21.8	22.198	21.8	26.662	21.8	21.786	21.8
		3	20.302	21.8	18.371	21.8	21.735	21.8	21.223	21.8
		SD	1.11	0.053	2.025	0.0	3.15	0.0	0.42	0.0

Tab. 3: development of the power output; significant Standard Deviation (SD) is marked red

5 Conclusion

The integration of OPC UA into an application can be an easy and a straightforward task. Compared to classic OPC, OPC UA is built on a simplified architecture. In case other features are needed, the same building blocks can be used to design new models without the need to start from the beginning again. The OPC UA protocol is an industrial protocol that is used to communicate between devices within plants and factories.

In the simulation application, the OPC UA stack shows advantages, compared to the basic implementation, although the functionality has not yet been fully exploited. As a future work, we want to research features OPC UA provides, such as UA Discovery or authentication or the more sophisticated ABAC, within this paper's context. Besides that, we intend to implement and do more experiments to study the effect a real time communication protocol such as Profinet can have on the communication.

6 Acknowledgement

Some of the addressed topics are being elaborated as part of Framatome GmbH's participation in the DECENT R&D (2018-2020) with Technical University of Munich (Germany), FORTISS (Germany), IBDM (Germany), FENECON (Germany), VTT Technical Research Centre of Finland Ltd (Finland), Empower IM Oy (Finland), Wirepas (Finland) and Fourdeg (Finland), partially funded by German Ministry BMWi as well as

the ABAC R&D with University of Siegen.

Bibliography

[Al07] Alvisi, L. et.al.: How robust are gossip-based communication protocols?. ACM SIGOPS Operating Systems Review, vol. 41, no. 5, pp. 14-18, 2007.

[Cr17] Croce. D. et al.: Overgrid: A Fully Distributed Demand Response Architecture Based on Overlay Networks. IEEE Transactions on Automation Science and Engineering, vol. 14, no. 2, pp. 471-481, 2017.

[Di10] Dimakis, A. G. et.al.: Gossip Algorithms for Distributed Signal Processing. Proceedings of the IEEE, vol. 98, no. 11, pp. 1847-1864, 2010.

[DZ20] Dzone, CoAP Protocol: Step-by-Step Guide, https://dzone.com/articles/coap-protocol-step-by-step-guide, accessed: 23/05/2020.

[EAD16] Engels, J.; Almasalma, H.; Deconinck, G.: A Distributed Gossip-based Voltage Control Algorithm for Peer-to-Peer Microgrids. IEEE International Conference on Smart Grid Communications, Sydney, NSW, Australia, 2016.

[Fr13] Frejborg. A. et al.: OPC UA Connects your Systems – Top 10 reasons why to choose OPC UA over OPC. Finnish Society of Automation, Biannual Seminar, Finland, 2013.

[Je11] Jelasity, M.: Gossip. In (Di Marzo Serugendo G., Gleizes MP., Karageorgos A. ed.) Self-organising Software. Natural Computing Series. Springer, Berlin, Heidelberg, pp. 139-162, 2011.

[KH15] Koukoula, D. I.; Hatziargyriou, N. D.: Convergence Acceleration of Gossip Protocols Applied for Decentralized Distribution Grid Management. IEEE Eindhoven PowerTech, Einhoven, Netherlands, 2015.

[KH16] Koukoula, D. I.; Hatziargyriou, N. D.: Gossip Algorithms for Decentralized Congestion Management of Distribution Grids. IEEE Transactions on Sustainable Energy, vol. 7, no. 3, pp. 1071-1080, 2016.

[Kr11] Krkoleva, A. et.al.: Requirements for Implementing Gossip Based Schemes for Information Dissemination in Future Power Systems. IEEE PES International Conference and Exhibition on Innovative Smart Grid Technologies, Manchester, United Kindom, pp. 1-7, 2011.

[Le16] Lequay, V. et.al.: Flexible Load Shedding using Gossip Communication in a Multi-Agents System. IEEE International Conference on Self-Adaptive and Self-Organizing Systems, Augsburg, Germany, 2016.

[Mi18] Ming, Y. et al.: Distributed Energy Sharing in Energy Internet Through Distributed Averaging. Tsinghua Science and Technology, vol. 23, no. 3, pp. 233-242, 2018.

[Mu20a] Muehlbauer. N. et al.: Open-Source OPC UA Security and Scalability. 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA); Vienna, Austria, 2020.

[Mu20b] Muehlbauer, N., et al.: Open-Source OPC UA Features and Interoperability. GI

conference – INFORMATIK 2020, Karlsruhe, Germany, 2020.

[MU20] Muutech, Comparison MQTT vs OPC-UA, https://www.muutech.com/en/comparison-mqtt-vs-opc-ua/, accessed: 25/05/2020.

[OP10] OPC Datahub, What is OPC?, https://opcdatahub.com/WhatIsOPC.html, accessed: 14/04/2020.

[OP18] OPC Foundation, OPC Technology Well-positioned for Further Growth in Tomorrow's Connected World, https://opcfoundation.org/wp-content/uploads/2017/11/OPC-UA-Interoperability-For-Industrie4-and-IoT-EN.pdf, accessed: 06/04/2020.

[SP20] SpotLightMetal, IoT Basics: What is OPC UA? https://www.spotlightmetal.com/iot-basics-what-is-opc-ua-a-842878/?cmp=go-aw-art-trf-SLM_DSA-20180820&gclid=Cj0KCQjw2PP1BRCiARIsAEqv-pQ7tSJYVdlJPyXvxcfH_e3k8a_WvCeIldc5iPXFkUZnjk0nnbaEZcIaArWBEALw_wcB, accessed: 14/05/2020.

[TE20] electrek: Tesla has a new product – Autobidder, a step toward becoming an electric utility, https://electrek.co/2020/05/03/tesla-autobidder-new-product-electric-utility/, accessed: 27/06/2020.

[Wa19] Wang, R.; et al.: A gossip-based asynchronous distributed algorithm for economic dispatch problem with transmission losses. IEEE PES Innovative Smart Grid Technologies Asia, Chengdu, China, 2019.

[Wa20] Wang, R.; et al.: A Gossip-Based Distributed Algorithm for Economic Dispatch in Smart Grids with Random Communication Link Failures. IEEE Transactions on Industrial Electronics, vol. 67, no. 6, pp. 4635-4645, 2020.

[BUV12] di Bisceglie, M.; Ullo, S. L.; Vaccaro, A.: The Role of Cooperative Information Spreading Paradigms for Smart Grid Monitoring. IEEE Mediterranean Electrotechnical Conference, Yasmine Hammamet, Tunisia. 2012