European Commission

JRC TECHNICAL REPORT

Blockchain in the Energy Sector

WP4 – On field deployment and analysis of experimental use cases

WP5 – Use-Case lab testing on Energy-Mobility integration using DLT

Nai Fovino, I.

Geneiatakis, D.

Giuliani, R.

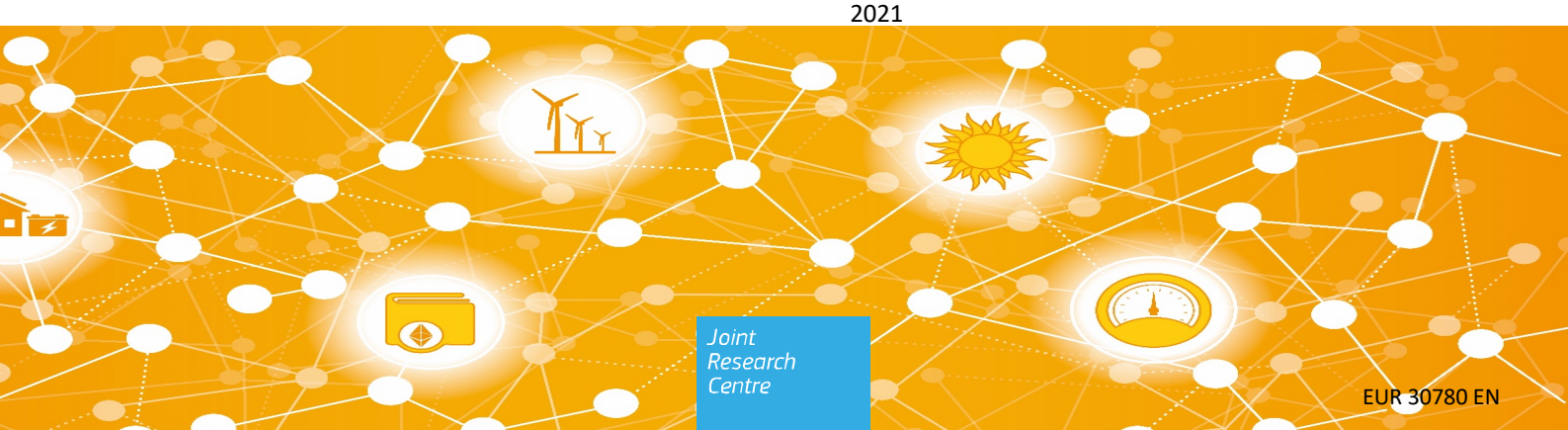
Kounelis, I.

Lucas, A.

Martin, T.

Steri, G.

2021



Joint Research Centre

EUR 30780 EN

Contact information

Name: Igor Nai Fovino

Address: Via E. Fermi 2749, TP 580, 21027 Ispra(VA), Italy

Email: igor.nai-fovino@ec.europa.eu

Tel: +(39) 0332 785809

EU Science Hub

https://ec.europa.eu/jrc

JRC125217

EUR 30780 EN

PDF ISBN 978-92-76-40550-4 ISSN 1831-9424 doi:10.2760/55525

Luxembourg: Publications Office of the European Union, 2021

How to cite this report: Nai Fovino, I., Geneiatakis, D., Giuliani, R., Kounelis, I., Lucas, A., Martin, T., Steri, G., Blockchain in the Energy Sector WP4 – On field deployment and analysis of experimental use cases, WP5 – Use-Case lab testing on Energy-Mobility integration using DLT, EUR 30780 EN, Publications Office of the European Union, Luxembourg, 2021, ISBN 978-92-76-40550-4, doi:10.2760/55525, JRC125217.

Contents

Abstract

This report describes the technical implementations of the Administrative Agreement between DG ENER and DG JRC, called Enerchain. In particular it presents the results of the experimental tests and of the results carried out to implement workpackage 4 and and workpackage 5 as defined in the technical annex of this administrative arrangement.

1 Introduction

In early 2019, DG ENER and DG JRC have signed an Administrative Agreement (AA), called ENERchain, in order to explore the area of "Blockchain Technologies in the Energy System". The purpose of this AA is to investigate the applicability of blockchain technologies to the power sector, assessing the actual potential of reaching greater accuracy, security, flexibility and money savings, which the technologies theoretically promise to bring.

This report comes as a follow up of WP3 ("Use cases identification and analysis") in which we analysed the most promising applications of Distributed Ledger Technologies (DLT) in the electricity energy sector. As a result, five different use cases were selected for further analysis and experimentation:

- Smart metering
- Energy communities
- Flexibility service provision
- Certification of origin
- Electric mobility grid integration

This large spectrum of use cases could be integrated into a future single system by the use of DLT technology as shown in Figure 1. In order to demonstrate the concept, we opted to use the same DLT technology, i.e. Hyperledger Fabric, for all of our experiments.

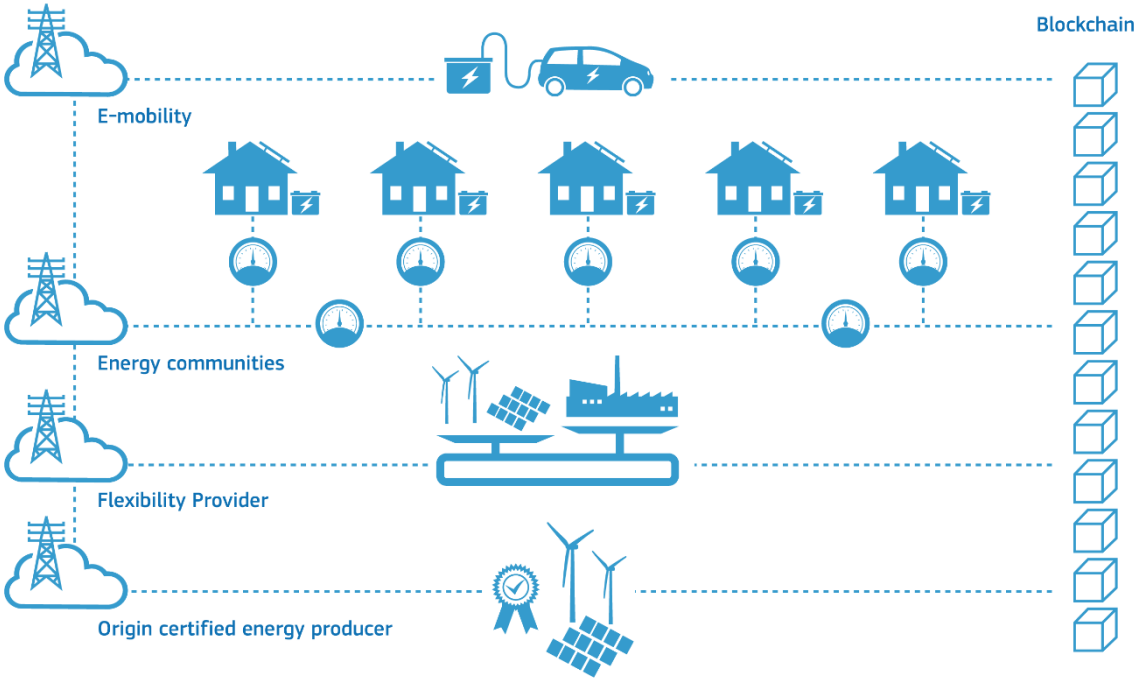


Figure 1. DLT integration of smart energy use cases

In the work presented in the following chapters, we describe the technical implementation details of the load flexibility services, the e-mobility grid integration and the energy communities use cases. We have implemented these use cases in our laboratories in order to prove their technical and logical feasibility. Based on the results of the various tests performed in all implementations, we evaluate how a DLT implementation affects the selected use cases. We also explain the difficulties we faced and potential issues to take into account for future developments. Moreover, as a vital part of such use cases is security, we have analysed in detail possible attacks and measures to defend against them.

The work is divided in the following chapters: chapter 2 describes the flexibility use case, chapter 3 the e-mobility use case, and chapter 4 the energy communities one. Chapter 5 lists the various problems that we faced in the

blockchain implementation of the use cases, while chapter 6 analyses all security related aspects. Finally chapter 7 concludes the report.

2 Flexibility and e-Mobility Use Cases

The flexibility and the e-mobility use cases were coupled in the same architectural development due to their similarities and potential application. The first refers to a demand response exercise, in which a battery storage unit provides or charges electricity to/from the grid upon a request, thus exploring its flexibility in terms of load. This is called the flexibility use case. The second focuses on the energy integration from electric mobility where likewise concepts of V2G were explored, also enabling the financial settlement of the transaction performed. Due to the sequence of information and the number of potential nodes involved, a common approach to the architecture of the two settings was developed and is described as follows.

2.1 Blockchain Architecture

The use cases are implemented over a Hyperledger Fabric infrastructure, which consist of the (a) ordering service, and (b) core blockchain network, as Figure 2 illustrates. In particular, any Hyperledger Fabric based blockchain system consist of the following fundamental components:

- **Ordering Service:** The main task of the ordering service is to sort the messages/requests exchanged between the participants. It consists of the following services: (a) Zookeeper (3 instances), Kafka (4 instances) and Ordering (3 instances). Each instance is executed on a different machine, and thus can be distributed to different 'locations'. According to Hyperledger Fabric documentation this is the minimum configuration in terms of capacity for supporting fail-over at the ordering service side. In particular, in this setup only one instance for each of the different services can be in fail status without affecting the ordering service availability. For instance, if one of the Kafka instances is not responding, the ordering service is still functional, however, if a second one becomes unavailable, then the ordering service will not be functional.
- **CouchDB:** Is the database that maintains all the valid transactions, known as world state, of the blockchain and allows the storage of JSON objects.
- **Peer**: Is a fundamental service of the Hyperledger Fabric as it stores the ledger and validates the transactions according to the defined policy.
- **Certificate Authority (CA):** Provides digital identities (certificates) to the participants (users) of the organisations (i.e., Aggregators) to the blockchain network, supporting a complete life cycle of the generated certificates.
- **Smart contract:** Implements the use case in the blockchain network providing access control, message conformity check, writing, reading and validation procedures.
- **Application interface:** Provides the interface for interacting with the blockchain, which is implemented as a REpresentational State Transfer (REST) service. This component accomplishes all the interactions on behalf of the user in order to commit a transaction in the blockchain network. In fact, this component generates the flexibility use case messages in JSON format.

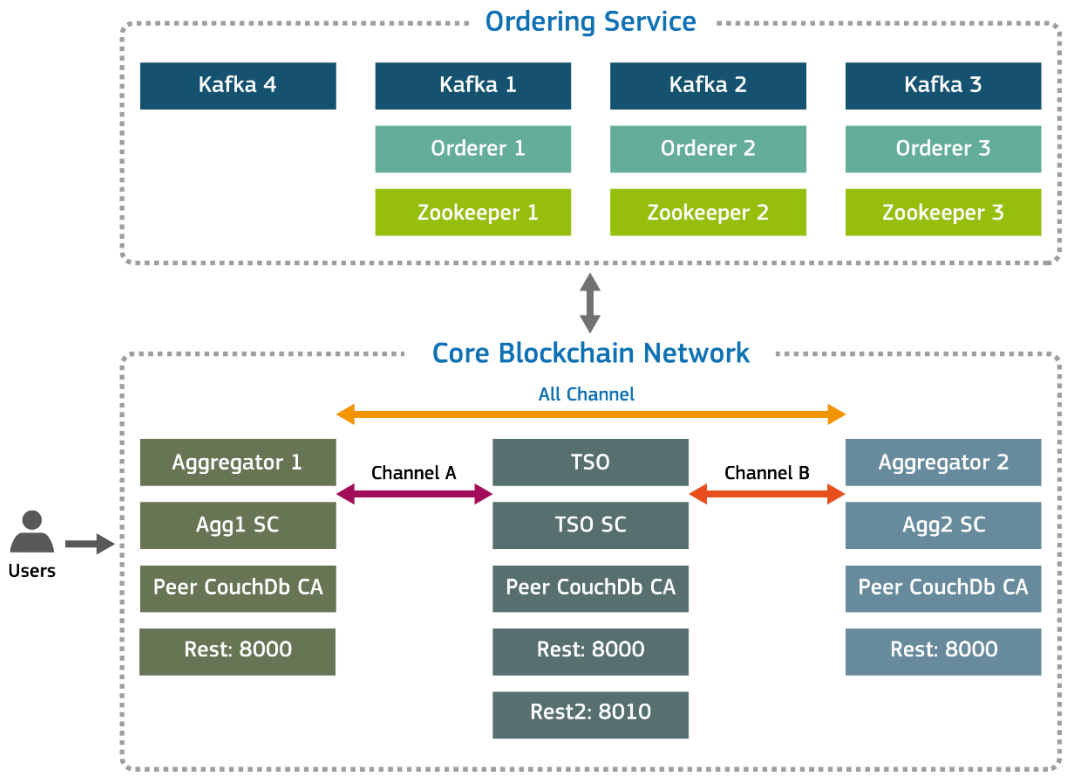


Figure 2. Flexibility example blockchain system implementation considering Hyperledger Fabric infrastructure. All aggregators and SO can share 'public' information through Allchannel and private data can be shared via bilateral channels.

The core blockchain network consists of the main participants i.e., aggregators and SO, or charging point operators in order to distribute the corresponding information and sharing common business logic and rules. Each participant in the network hosts a peer, CouchDB, a CA, the smart contract and the application interface.

At this point it should be noted that all the Hyperledger Fabric related services both for the ordering service and core blockchain network nodes rely on standard predefined deployment options. All the underlying network communications, between the participants (clients, peers, and the ordering service), are securely protected by TLS. All the required certificates and private keys both for TLS and for the blockchain services are generated during blockchain network initialisation procedure according to Hyperledger Fabric specifications.

2.2 Transaction Flow

To achieve the data sharing service over the Hyperledger Fabric infrastructure the standard transaction flow is the following (see Figure 3):

(a) **Transaction submission**: A client submits a signed transaction proposal to the corresponding node (i.e., Aggregator), in which it includes also his/her identity (X.509 certificate).

(b) **Transaction verification**: The aggregator node verifies the validity of the transaction, simulates the execution of the related smart contract functionality, and returns a signed endorsement result to the client. The identity of the aggregator node (X.509 certificate) is also included in this response.

(c) **Transaction ordering**: The client sends the transaction proposal and the related result to the ordering service. The latter orders all the transactions coming from different clients, builds the blocks and broadcasts them to all the blockchain nodes.

(d) **Transaction commitment**: All participant nodes receive the blocks of transactions from the ordering service and, upon verification of their validity in the context of the deployed policy e.g., at least one node should have signed the transaction, write them in their local copy of the ledger.

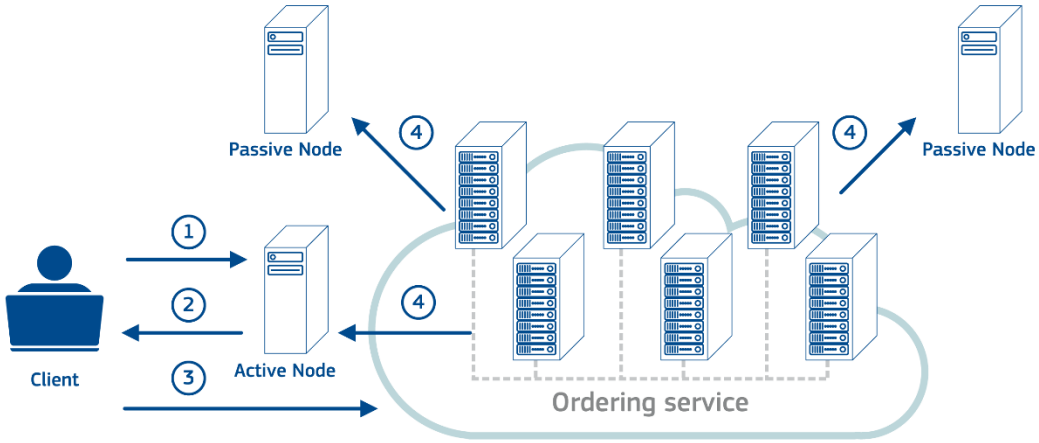


Figure 3. Overview of transaction flow on Hyperledger Fabric.

2.3 Methodology

The use-cases are deployed on the Experimental Platform for Internet Contingencies (EPIC), an infrastructure located in the JRC, which provides an emulation based environment enabling testing repeatability and consequently results reproducibility. In terms of capacity EPIC currently supports the deployment of network architectures up to 356 virtual nodes, while it can be interconnected with different network topologies, and special physical equipment such as Programmable Logical Controllers (PLCs).

In more detail, the EPIC allows the implementation and emulation of a realistic network topology by assigning physical equipment in order to create the designed emulated network. Over this network one can deploy applications and services specific for the experiment, e.g., Docker containers, can be initiated and configured either automatically or manually. The data acquired in the use case can be from a smart meters or a dedicated asset internal meter but typically retrieved with one of the following approaches considering possible limitations on connectivity between EPIC and the Smart grid lab (SGILAB), where the batteries are located network that are isolated:

- The SGILAB will supply the data in a public accessible service and through a pull mechanism, the data will be pushed to the emulated blockchain network.
- The SGILAB will use virtual private network to provide the data to the network, or
- There will be a direct connection between SGILAB and EPIC network.
- Data will be shared offline.

So briefly, considering the deployment of flexibility use case over the Hyperledger Fabric we retrofit it with the proper use cases in order to assess its effectiveness as Figure 4 depicts.

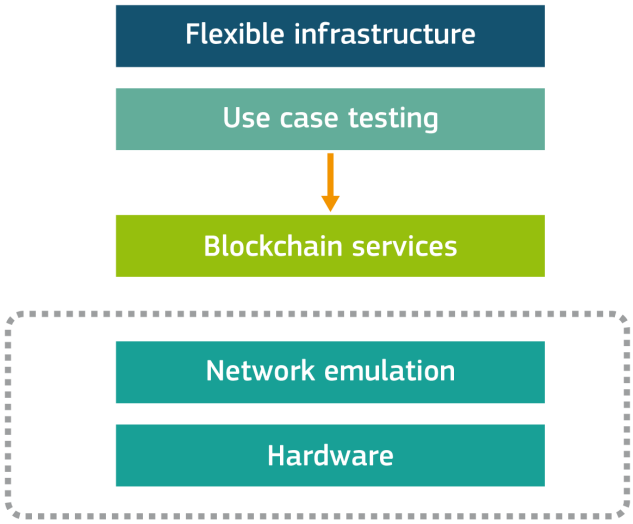


Figure 4. Flexibility use case evaluation approach

2.4 Smart Contract implementation

We deployed the smart contract on Go language as is supported by Hyperledger Fabric, a type-safe language that generates a compiled code enabling fast execution in comparison to other interpreted languages [1]. In fact, the smart contract enables the users or the corresponding entities of the system to:

1. Insert new transactions for recording demand response requests, and
2. Validate the correctness of the aggregators' claims recorded as transactions in the blockchain.

We model the data required for this use case in a data structure called EnergyFlex as depicted in the following Figure that consisted of following fields:

1. **Energy**: models a unique identifier for the transaction that will be submitted to the blockchain
2. **Time**: defines a timestamp linked to the time in which the energy measurements were taken
3. **Voltage1**: The root mean square (RMS) value of the line to line voltage (V) in the corresponding time step
4. **Voltage2**: The RMS value of the line to earth voltage (V) in the corresponding time step
5. **Power**: The running Power (+ if charging from the grid, or – if injecting into the grid) of the asset activated in kW in the corresponding time step.

The EnergyFlex data structure definition is shown in the pseudo-code below:

```
type EnergyFlex struct
{Energy   string `json:"Energy"`
Time     int    `json:"Time"`
Voltage1 int    `json:"Voltage1"`
Voltage2 int    `json:"Voltage2"`
Power    int    `json:"Power"`}
```

The transactions submitted to the blockchain system are sent by users in the form of a POST request with the following format `http://10.120.50.110:8000/Trasnsaction -H content-type: application/json' -d '{"Timestamp":"1", "Voltage1":2, "Voltage2":5, "Power":2}'`.

The smart contract parses JSON's message arguments and makes the corresponding controls to determine if the message complies with the rules of the system, and whether the values are in the expected ranges. If both controls are successful, the execution of the smart contract continues and writes the data to the blockchain, otherwise the transaction is recorded as a failed one in the blockchain.

This functionality is illustrated in the following pseudo-code example. Note that the blockchain allows transactions to be submitted only by entities that hold the corresponding certificate that have been issued by the corresponding certificate authority.

Create main function to add JSON info

```
func (cc *EnerChaincode) addEnergyJSON(stub shim.ChaincodeStubInterface,
args []byte) sc.Response {
      if len(args) == 0
        {fmt.Println("json string validatiton...")
        return shim.Error("Empty argument")}
```

Validate against json-schema

```
    jsonString := string(args[:])
    fmt.Println("jsonstring")
    fmt.Println(jsonString)
    if len(jsonString) == 0
        {return shim.Error("Empty argument")}
```

Extract info

```
    energy, time, voltage1, voltage2, power := extractInfo(stub, args)
    if time > current_time
 {return shim.Error("timestamp value error")}
    if voltage1 > threshold1 || voltage1 < threshold2
 {return shim.Error("voltage values errors")}
    if voltage2 > threshold1 || voltage2 < threshold2
 {return shim.Error("voltage values errors") }

    if power > threshold_p1 || power < threshold_p2
{ return shim.Error("power values errors") }
```

Write the data on the blockchain

```
    err1 := stub.PutState(energy, args)
    if err1 != nil
```

```
        {fmt.Println("error!!!! Failed to put to world state") }
    return shim.Success(nil) }
```

It should be noted that all the interactions with the blockchain system are accomplished through the restful service which acts on behalf of the user. As soon as a transaction is accomplished a corresponding entity (i.e., the SO) is able to check the validity of the transaction(s) and create a new block in the chain, illustrating the validation procedure according to the specifications of the system. The validating entity through a post request sends a message to the smart contract to validate the correctness of the aggregators 'submissions'. To do so, it is assumed that the end-users submit also their values in the blockchain so the TSO can retrieve them and confirm that the aggregator fulfils the corresponding demand response request.

2.5 Use Case Overview and Lab set up

Upon a request (1) by a System Operator (SO) to manage the grid, with a specific amplitude and duration (which is specified in a market position or bilateral contract), (2) the Aggregator rechecks the availability of demand response (DR) flexibility of its assets, and the service provision occurs by setting the loads/assets to a desired/allowed level (3). After the event takes place the TSO must confirm that the variation was actually performed in order to find the corresponding imbalance price/compensation (4). This procedure is illustrated in Figure 5.

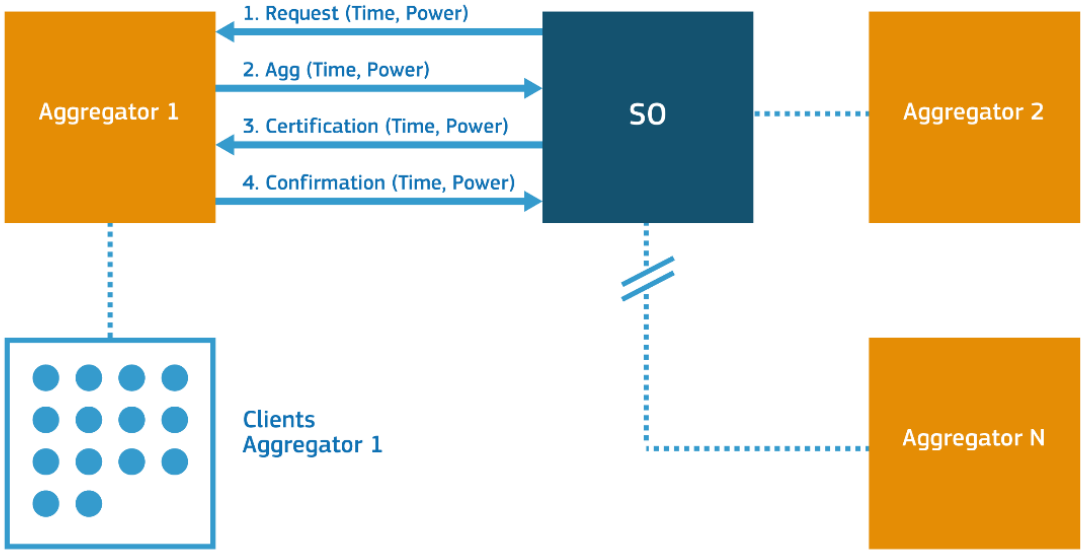


Figure 5. High level architecture of flexibility use-case

In terms of the experimental set up in the laboratory, Figure 6 presents the information flow between the assets involved. The active assets are either the battery storage unit or the electric vehicle, both under a meter. The computers act as gateways to collect and send data between the SGILAB and the EPIC Lab since they are in different buildings.

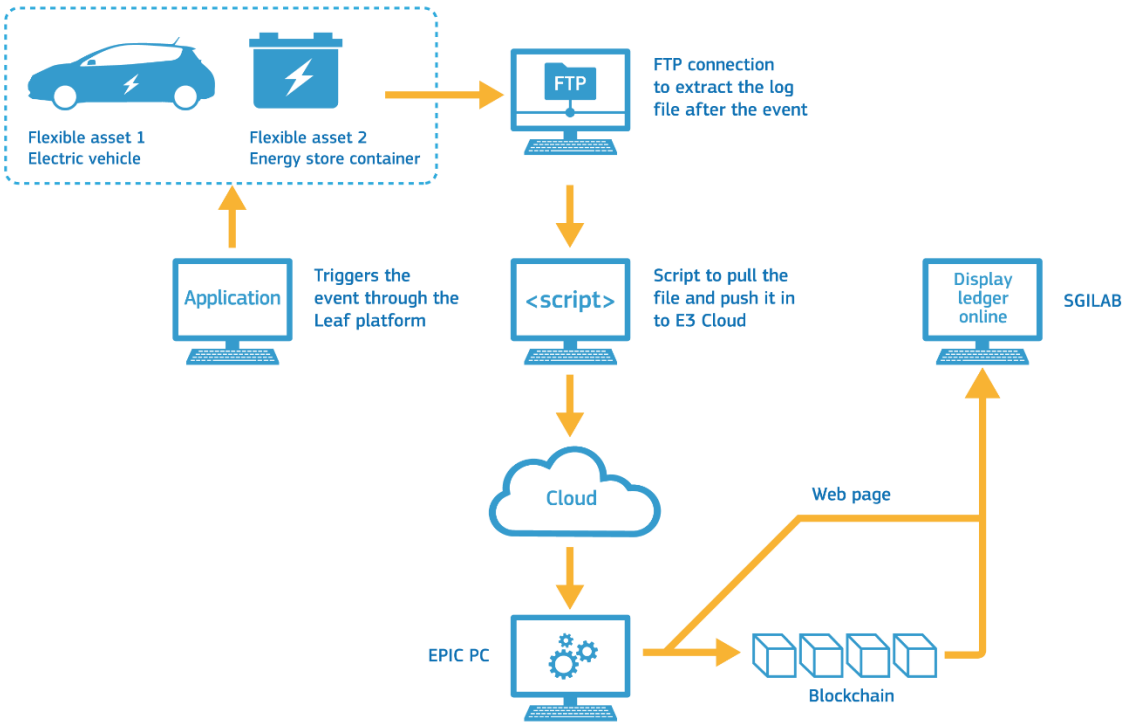


Figure 6. Laboratory equipment set up for Flexibility/Mobility Use cases

Figure 7. SGILAB's energy storage container

2.6 Data Generation

The current architecture consists of a Nissan electric vehicle (EV) with a 24 kWh battery connected to a type2 charger (<22 kW) and a storage system with 225 kW and 450 kWh capacity, connected to the same busbar in the SGILAB, which allows a bi-flow of power from and towards the grid. This means that a battery can charge or discharge from the grid, hence G2V and V2G can be performed, either using the vehicle or the storage system respectively which will be the one used for the sake of the demonstration.

The triggering of the experiment is done by an online platform, such as the one from an Aggregator, scheduling the battery to charge or discharge at a given time. The battery system has a SIM card allowing this type of communication. The communication with the asset is hence a bi-flow, exchanging data and signals. Such signals/data are part of the smart contract. Figure 8 shows the interface with the platform to schedule and initiate the DR event

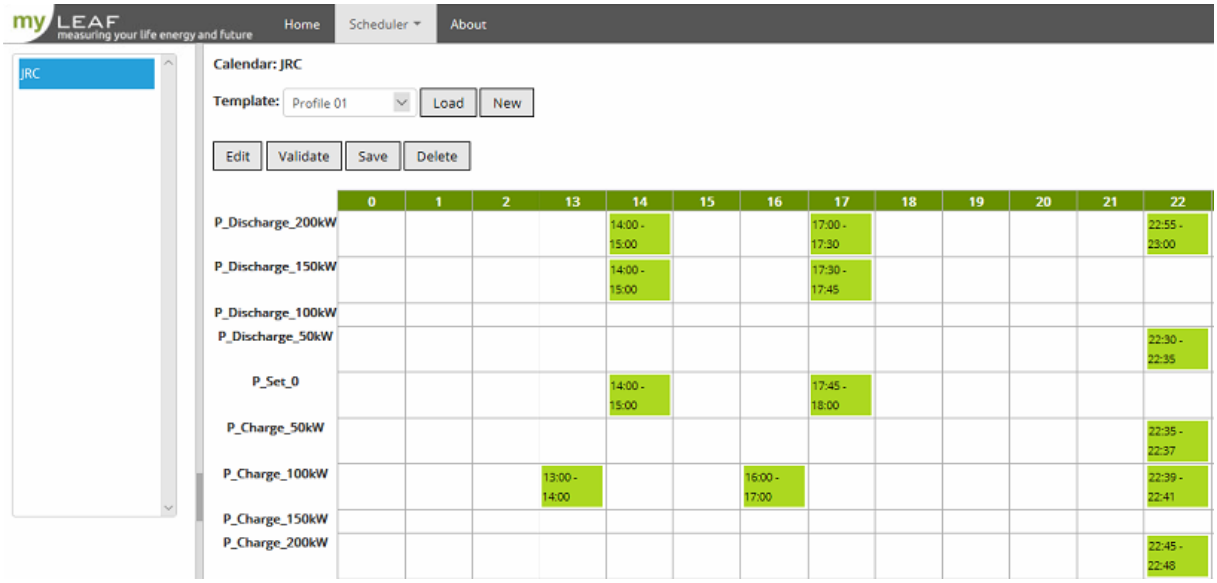


Figure 8. Interface for scheduling demand response event

Typically, if the data is acquired from a smart meter, a PLC or wireless communication is used. Since the use case is in a closed environment, the variables collected from the meter that reads such transaction are obtained through an FTP connection and transmitted in a log file (CSV format) after the DR event is over. An interval of readings before and after (+-15 min) the event is also required and will be provided in order to perform a baseline to detect that during the request time there was actually the requested change in power. The variables in stake are the very minimum for the sake of the experiment:

To be transmitted from the prosumer to the aggregator (minimum):

- Active Power,
- Prosumer ID,
- Time stamp.

The aggregator shares some of the data with the TSO and what will be recorded on the ledger is:

- Aggregated active Power (if more than 1 asset is involved),
- Time stamp,
- Aggregator ID.

Power is stated in kW and the time step of the time stamp is in minutes. The ID is a string or object variable, unique for each aggregator or prosumer. We assume a prosumer can only be a service provider through one aggregator.

2.7 Evaluation

Figure 9, shows the Active Power of the battery storage system during the demand response event +-15 minutes. This is the logfile actually recorded in the local PC retrieved by a simple FTP connection. The event lasts for 255 minutes (4.25h). One can observe there is a ramp up and down, which is the time the asset takes to reach the set power. The ramp up/down is foreseen and allowed by the market but it has limits. This is one of the characteristics that distinguish asset quality in the participation of DR events. The amplitude of the requested demand response is of -74 kW, which in this case corresponds to injecting into/supplying the grid active power.

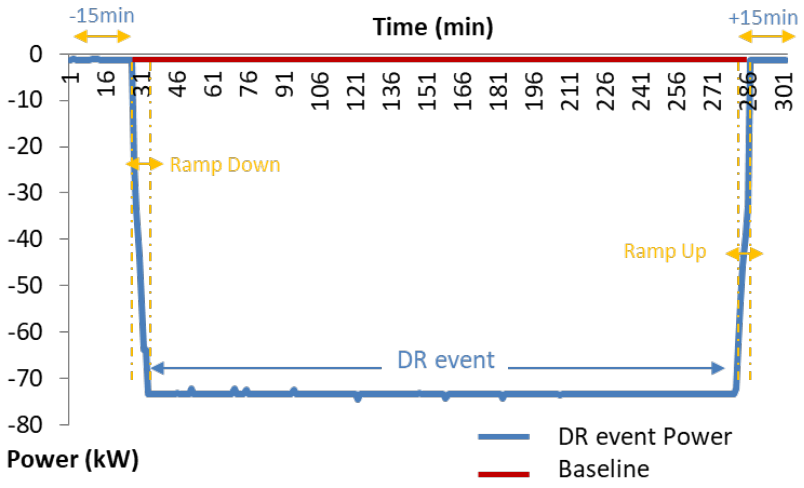


Figure 9. Load profile of the asset during the flexibility provision event

Shown in red, is the baseline, which continues stable as the asset switch from a steady state of -1 kW to -74 kW. The total energy supplied in the DR event is hence of (74-1)*4.25h=310.25 kWh. The event is then registered on the explorer from the Hyperledger Fabric, which can be seen with the following format with 54 characters:

```
Transaction ID:
"f5d0efd26c6361a74db3aa9ffa719ed2212df495795f58b0fc40578a137605a3"
```

Finally the data format stored in the ledger is the following:

```
{"Energy":"energyvalueAsset-1-18","Time":1581894000,
"Voltage1":398,"Voltage2":229,"Power":-1}
```

Each entry in the ledger corresponds to a given second or minute depending on the time step chosen. The format shown above contains the first second of the DR event of the addressed use case. The csv file from the retrieved asset (storage unit) meter contains more attributes, however only the time, two voltages and power were chosen for the simplicity of the use case. With such information the DR can be confirmed by the SO that was in fact implemented. The block details are provided by the explorer as shown in an example in Figure 10.

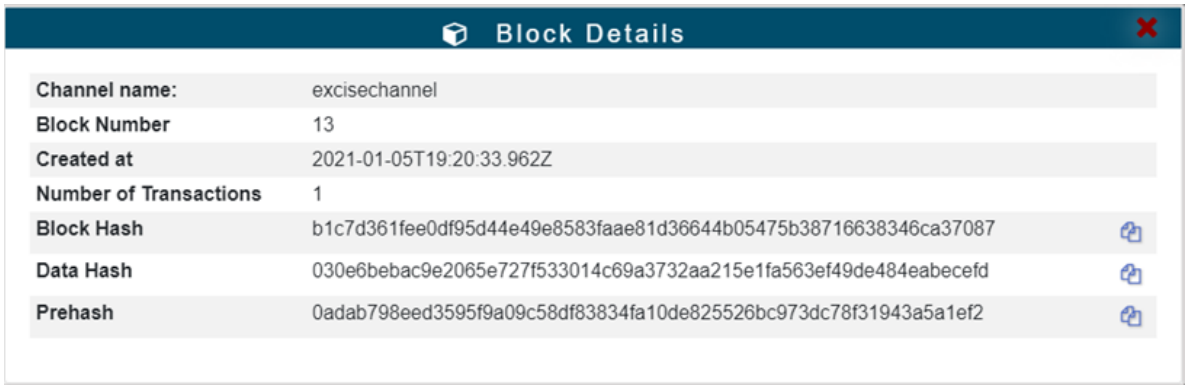


Figure 10. Block details in the Enerchain DR use case example

To evaluate the effectiveness of flexibility use, we deploy the use case it in a network that consists of different number of participants. We study how such a use case will scale considering different (a) number of participants, and (b) number of transactions submitted in the system per second (TPS). In the current setup we assume that all the participants are connected to the network over a network that supports 1 Gbps with a latency of 3 ms. Table 1 shows the executed scenarios for the evaluation of the flexibility use case.

Table 1. Overview of executed tests cases for the flexibility use case

No. of participants in blockchain network	Number of requests
3	1/2/4/8/16/32/36/40/48/50/52/54/64/128/256/512 transactions per sec. sent to the ordering service
10	
28	

To assess the system performance, we use the following indicators:

- Request round-trip time that is the time elapsed from the moment in which a user submits an operation request (i.e., write) and the moment in which he/she receives service response. The monitoring procedure is accomplished by integrating a recording service both in the user and the REST service sides.
- System resources utilisation in which we monitor the utilisation of CPU and memory for all the related services. To keep track of the utilisation of system resources we relied on docker's built-in monitoring services.

In fact, these indicators demonstrate how the system performs in terms of required resources (system resource utilization) and whether end user's experience is affected (request round-trip time) considering different number of nodes in the blockchain network as well as number of transactions per second.

In the following paragraphs, we overview the results considering different number of participants in the network and transactions per second submitted to the system.

2.7.1 Request round-trip time

Figure 11 overviews the end to end execution time considering different sizes of blockchain network, i.e. 3, 10 and 28 participants, while different number of requests per second are submitted to the network. Results indicate that for up to 32 requests per seconds, the end to end execution time is less than 1 second (independent of the network size), without having any impact on end user's experience. However, as the number of transactions increases the total execution time grows exponentially. This is because the system cannot handle properly the increased number of requests; indeed, the more the network participants the more the transaction

end to end execution time takes to complete in rates greater than 32 transactions per second. This trend is also confirmed by the number of errors that occur during the different scenarios (see Figure 12). Note that the number of failures for higher than 48, 52 and 54 TPS for 28, 10 and 3 nodes correspondingly is more than 20% of the total transactions, that indicates that the system cannot handle such traffic and consequently its behaviour is unpredicted.

In particular, for a network size with 3 participants and a throughput up to 51 TPS, the total execution transaction time does not exceed two seconds. This can be considered as an acceptable response time for end users, taking also into account the number of errors. Should the system response be intensified, in 54 TPS the end to end delay reaches up to 10 seconds without losing its stability as the number of errors remains low. Of course as we increase the number of nodes, the end to end delay increases faster. For instance, in a network that consists of 28 members under 32 TPS the end to end delay approximates 2 seconds, while for 3 nodes it is less than 1 second. Figure 13 indicates an example of end to end delay for a specific time window for 8 TPS for all the cases; it should be noted that it is not expected that all the configurations would have exact the same behaviour i.e., in a network with 28 nodes the end to end delay time has high variability that is an indication that the system is already under stress. Note that we do not illustrate outcomes for TPS greater than 128 as the system becomes completely unstable with high percentage of submitted transactions failing.

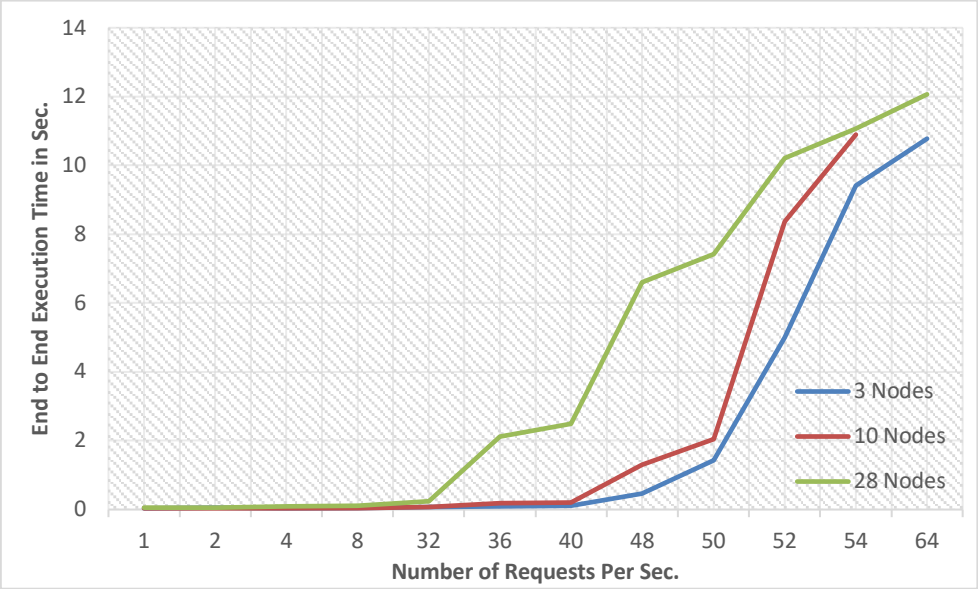


Figure 11. End to end execution time considering different blockchain participants and number of requests per second

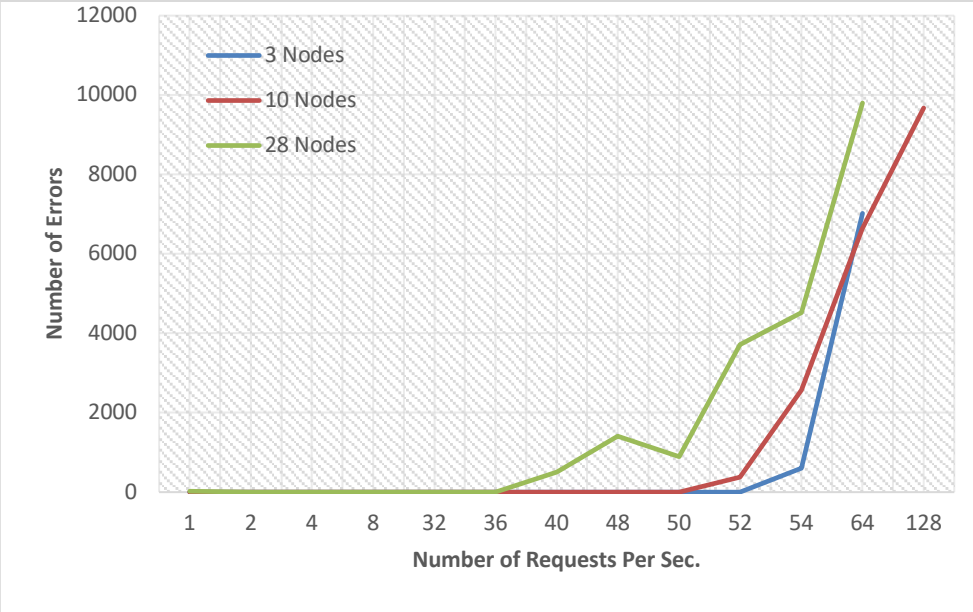


Figure 12. Errors considering different blockchain participants and number of requests per second

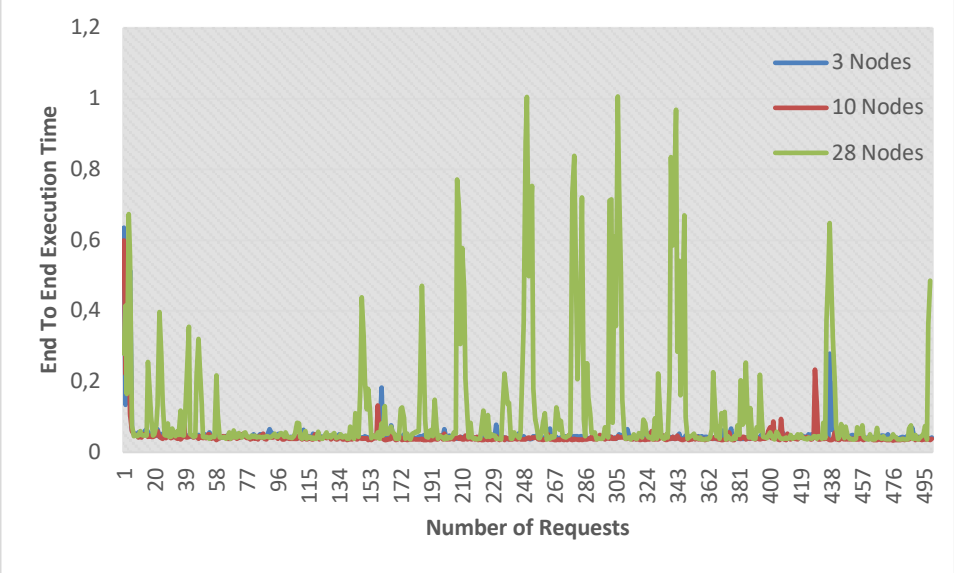


Figure 13. End to end transaction sample execution time for 3,10 and 28 peers per request in a scenario of 8 TPS

2.7.2 System resources utilisation

To identify the system resources utilization we monitor the following components:

a) an active node (node 1) that is a participant that submits requests continuously

b) a passive node (node 2) that receives all the successful submitted transactions, and

c) the main ordering node (ordering 1) that currently handles all the submitted transactions.

By monitoring these components we can identify the needs of a future setup for similar deployments with the current use case, i.e., would it be possible for such a use case to be deployed using the existing computational resources?

2.7.2.1 Bandwidth

Results indicate that as the number of nodes increase the bandwidth utilization of the ordering service increases as well, while the bandwidth utilization from the participants' perspective is not impacted at all. For instance, considering the case of 3 participants in the blockchain network at very low rates, the bandwidth utilization is the same for node 1 and for the ordering service; it approximates to 250 Kbits. However, as the number of transactions per second increases the bandwidth utilization sharps as well i.e., in 8 TPS the ordering service utilizes 2500 Kbits (see Figure 14). Note that as the end to end delay increases (e.g., in 1 TPS the average time is 0.04, while in 36 TPS the end to end delay doubles) the bandwidth utilization decrease slightly. Of course, this trend is followed as the number of TPS increases and as a consequence the number of errors increases causing smaller number of successful transactions and less bandwidth utilisation. Figure 15 and Figure 16 validate the same behaviour and that the number of nodes influence the bandwidth utilisation. This means that during the design phase of the system the proper network bandwidth should be in place for all the relative connections in order to eliminate failures due to the increased number of requests per second.

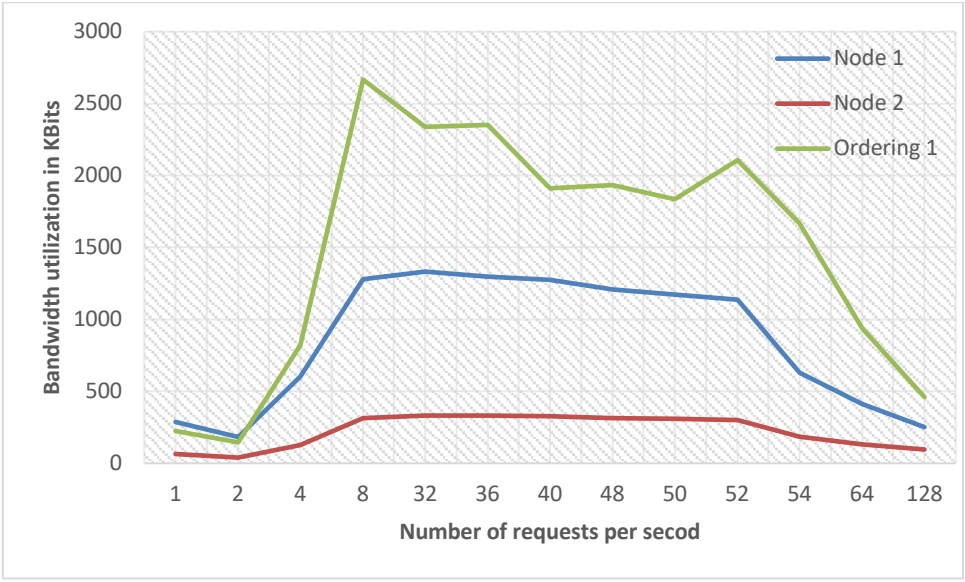


Figure 14. Bandwidth utilisation considering 3 blockchain participants and different number of requests per second

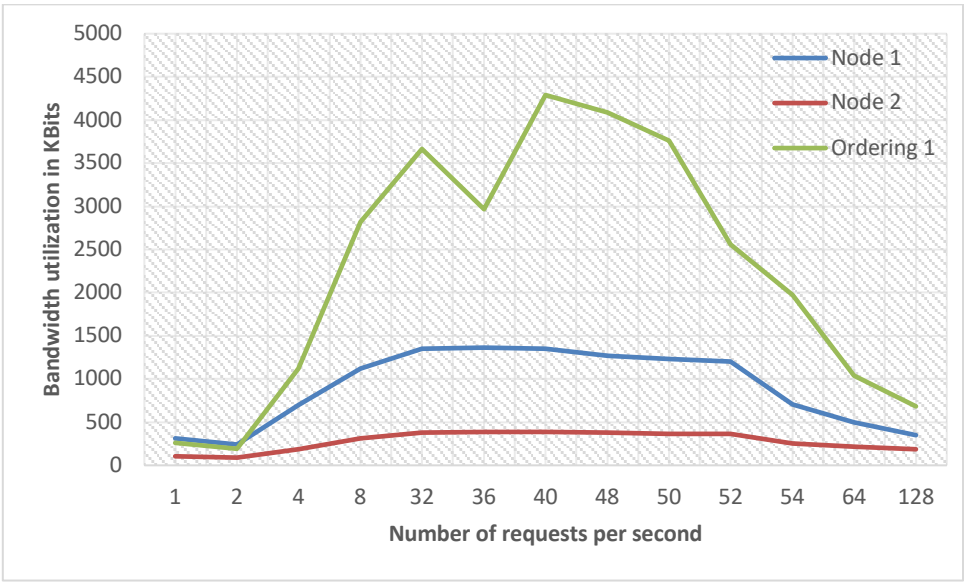


Figure 15. Bandwidth utilisation considering 10 blockchain participants and different number of requests per second

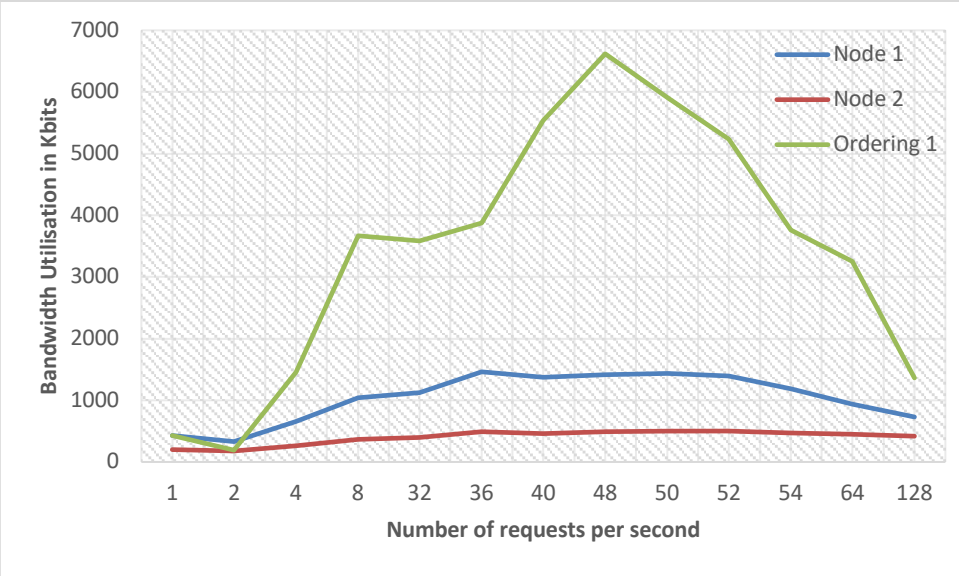


Figure 16. Bandwidth utilisation considering 28 blockchain participants and different number of requests per second

2.7.2.2 Memory

2.7.2.2.1 Ordering Service

For evaluating the ordering service in terms of memory utilization we monitor the main node of the service in which Zookeper, Kafka and Ordering service are running; however, the behaviour would also be the same for the other nodes of the service. In particular, Kafka service memory utilisation ranges between 0.5% and 4.5%. Though the relative variation might be significant, for the number of nodes it has it is not a noteworthy impact (see Figure 17). Overall, as kafka is in place for systems reliability, results show that the number of nodes and TPS have minor influence in the service memory utilization. Similar is the behaviour of Zookeper that has memory utilisation as little as 0.2% (see Figure 18).

On the contrary, as Figure 19 depicts it is clear that the Ordering service memory utilization increase as the number of TPS increase, without being affected by the number of participants in the network. As the number of TPS rise, the memory utilization increases gradually up to 5% and up to 50 requests per second. In rates greater than 50 TPS the number of errors increases and as a consequence the memory utilisation decreases.

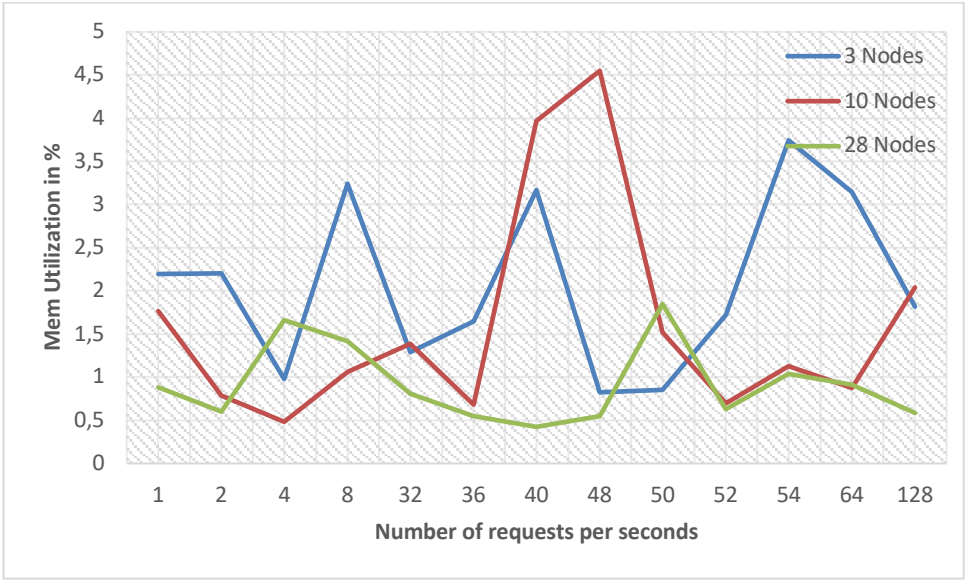


Figure 17. Kafka memory utilization of the main ordering service considering different number of participants and number of submitted requests

Figure 18. Zookeeper memory utilization of the main ordering service (node 1) considering different number of participants and number of submitted requests

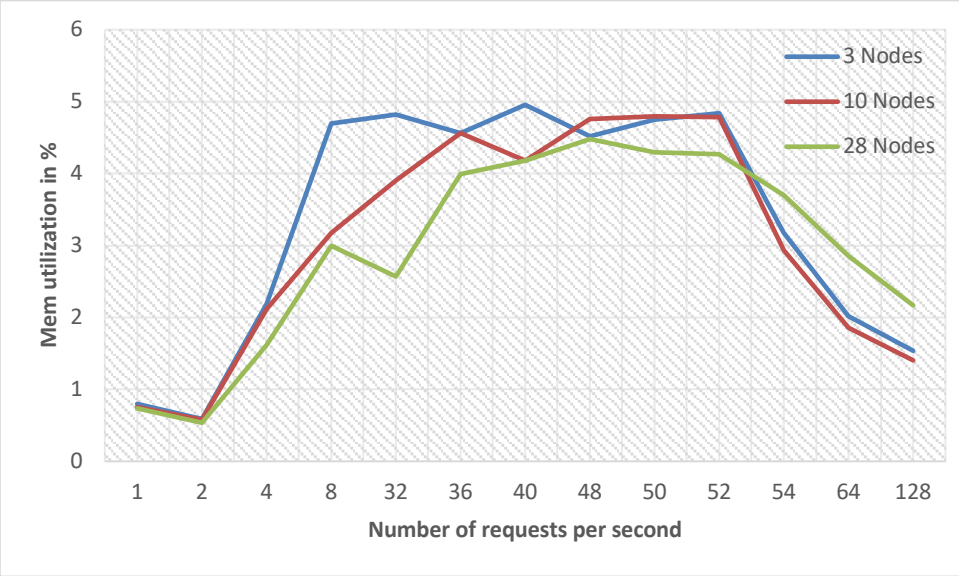


Figure 19. Ordering service memory utilization of the main ordering service considering different number of participants and number of submitted requests

2.7.2.2.2 Certificate Authority

The certificate authority service does not have any active role for the execution of transactions and consequently does not have any significant memory utilisation for both nodes that we monitor among the participants (see Figure 20, Figure 21).

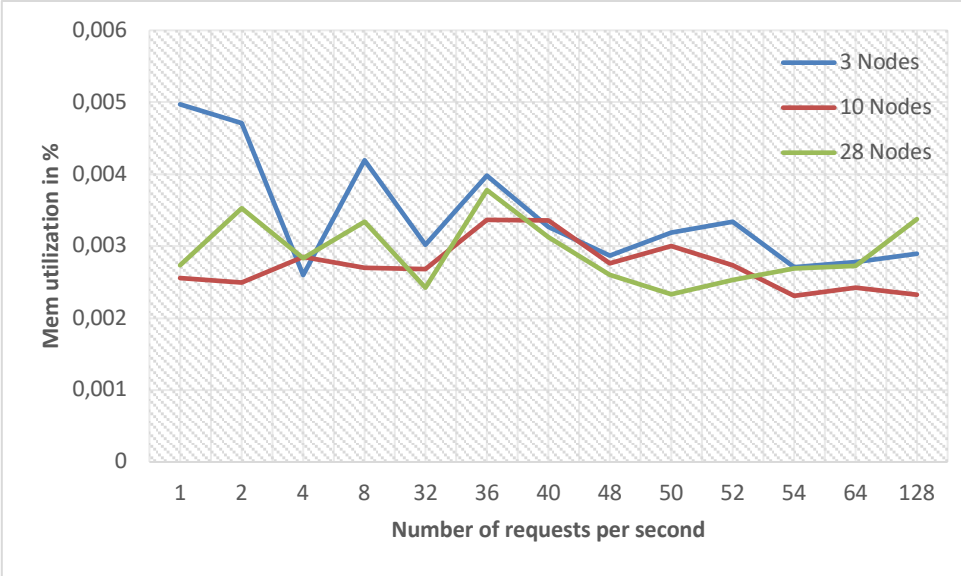


Figure 20. CA memory utilization for an active node

Figure 21. CA memory utilization for a passive node

2.7.2.2.3 CouchDB

Results show that nodes actively submitting transactions are highly impacted by the number of nodes that participate in the blockchain network in relation with memory utilization. Figure 22 illustrates the trend of memory utilization considering different number of nodes and TPS. In fact, TPS does not influence highly CouchDB utilization, however in case of increasing the number of nodes to 28, CouchDB memory utilization increases up to 8x. On the contrary, nodes that are not actively submitting transactions in the network, are not affected by the number of TPS but slightly by the number of participants (see Figure 23).

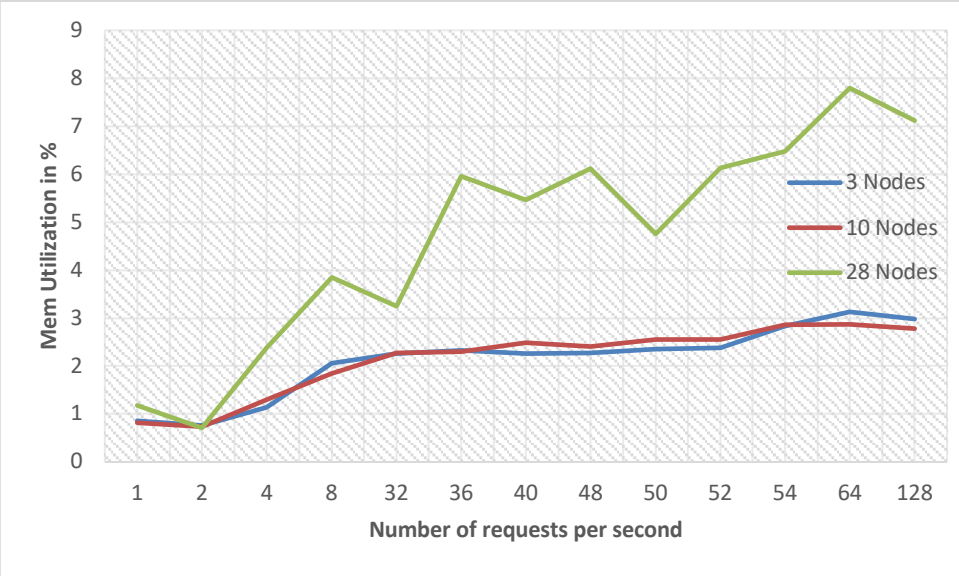


Figure 22. CouchDB memory utilization for an active node.

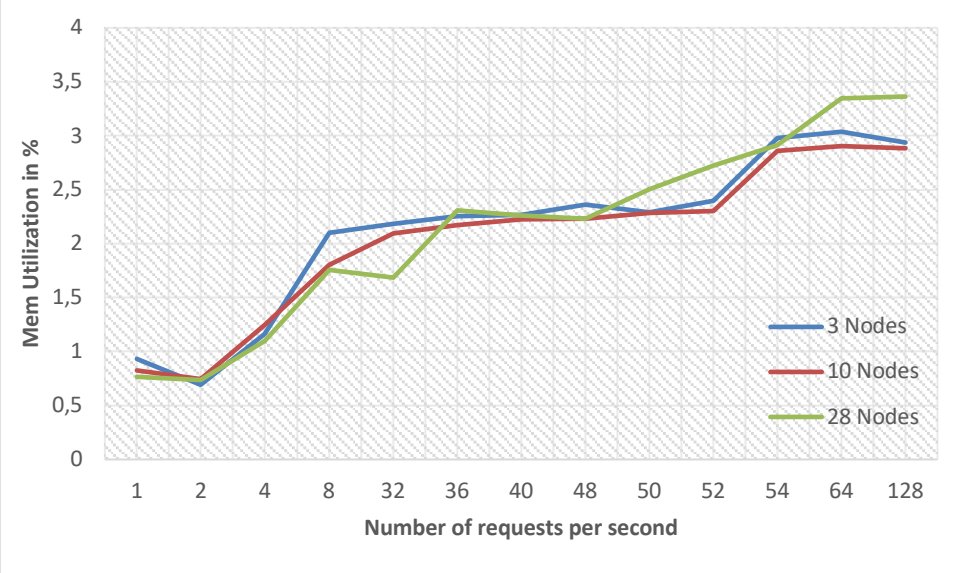


Figure 23. CouchDB memory utilization for a passive node

2.7.2.2.4 Peer

As far as the peer service memory utilisation is concerned the analysis shows that it is highly influenced by the number of participants in the blockchain network; the more the participants are, the more the memory utilization is. For instance, in case of 1 TPS the memory utilisation for a network with three participants is up to 5%, while when there are 28 participants the memory utilisation reaches up to 15% (see Figure 24).

A similar trend is followed when the number of transactions per second increases, for instance in the network size of 3 participants the memory utilization reaches a maximum of 15% (see Figure 24). The same is observed in a node that only receives the validated transactions (see Figure 25).

Figure 24. Peer memory utilization for an active node

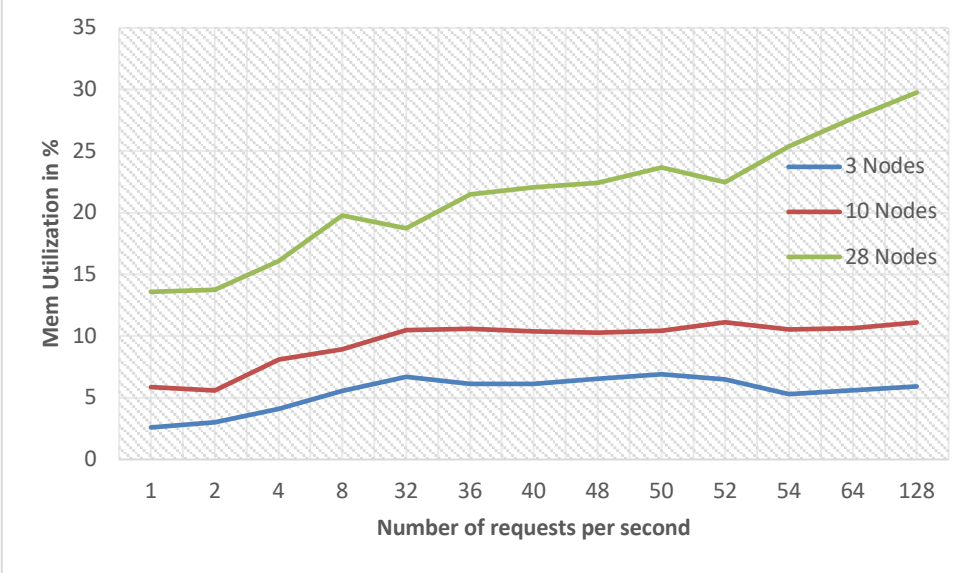


Figure 25. Peer memory utilization for a passive node

2.7.2.2.5 Client (Application Interface)

Clients, as expected, are not influenced by the number of participants in the network as they do not have any interaction with each other, but only with the peer and the ordering services. In fact, results indicate that the number of submitted transactions affect the memory utilisation of the client service where it reaches its maximum ~25% between 8 and 52 requests per second for an active node (see Figure 26). On the contrary a passive node (see Figure 27), as expected, has negligible memory footprint since it does not interact with the network.

Figure 26. Client memory utilization for an active node

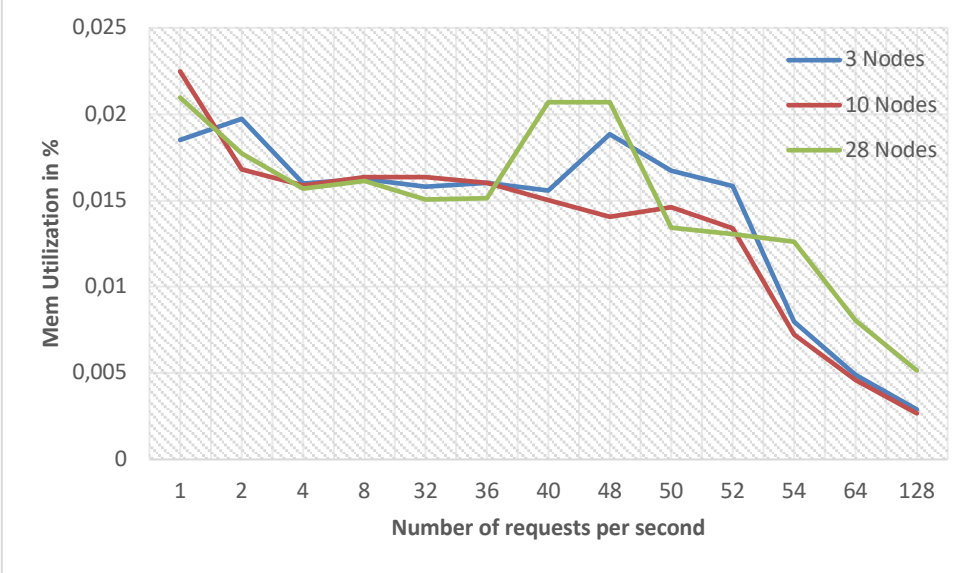


Figure 27. Client memory utilization for a passive node

2.7.2.2.6 Smart Contract

Smart contract for both active and passive nodes has a similar trend with the corresponding clients, as Figure 28 and Figure 29 illustrate. In fact, the smart contract memory utilisation increases as the number of requests grows, however, in no case does it exceed the threshold of 1%.

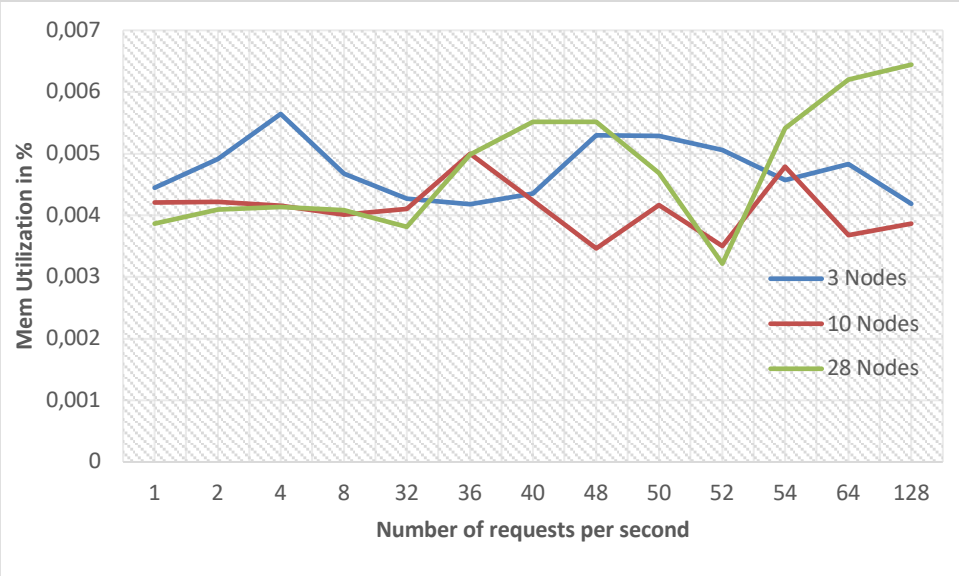


Figure 28. Smart contract memory utilization for an active node

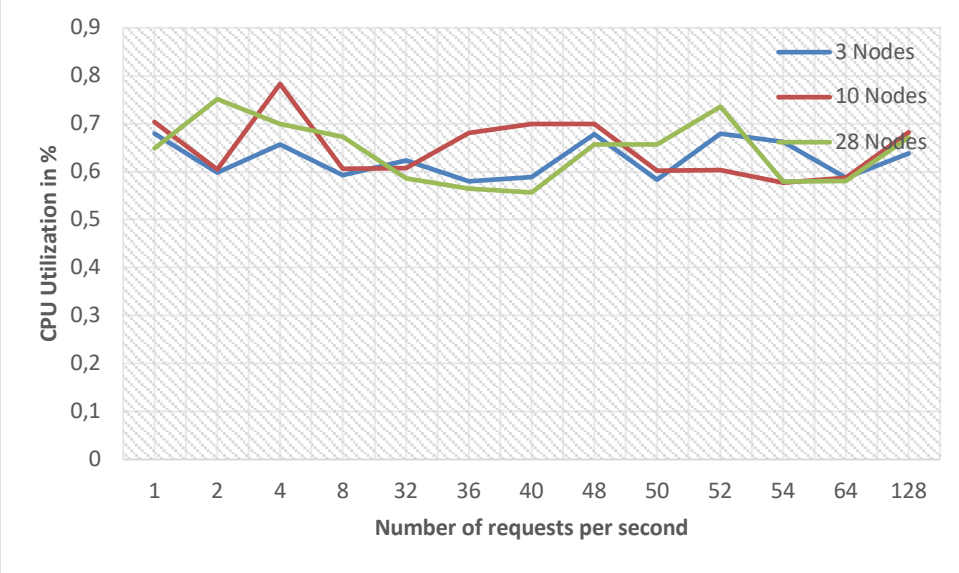


Figure 29. Smart contract memory utilization for a passive node

2.7.2.3 CPU

2.7.2.3.1 Ordering Service

For evaluating the ordering service in terms of CPU utilization we monitor the main node of the service in which Zookeper, Kafka and Ordering service are running; however, the same behaviour would also be the same for other nodes of the service. In particular, Kafka CPU utilization ranges between 6 to 14% without being influenced by the number of requests per second or the number of participants in the network. Under normal cases in which the system is not stressed, CPU utilization is ~6%. However, results indicate that when the system is under stress then Kafka CPU utilization increases; recall that the optimal case for 3 participants can be up to 52 requests per second (see Figure 30).

With reference to Ordering and Zookeeper services CPU utilization ranges between 1% to 3%, without being impacted by number of the blockchain participants and requests per second (see Figure 31 and Figure 32).

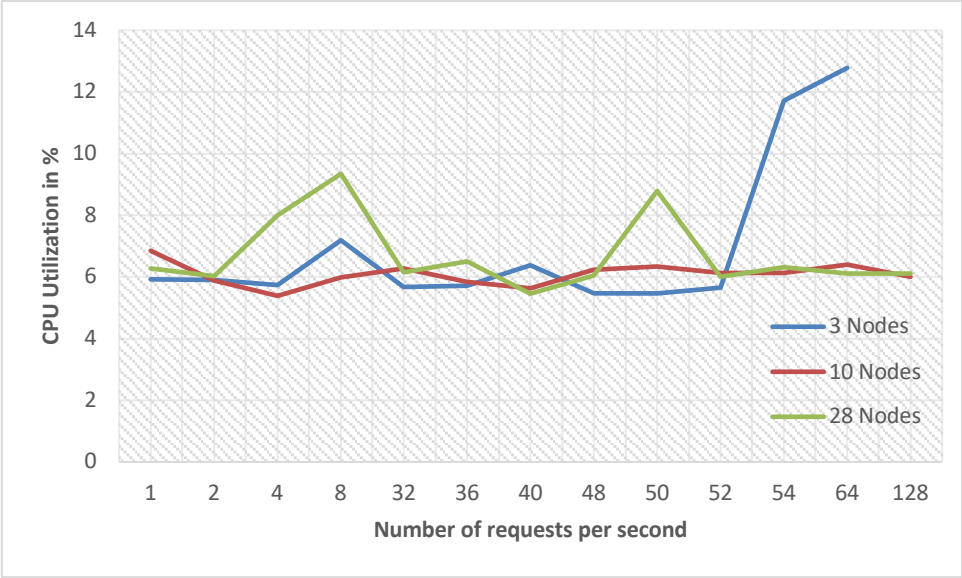


Figure 30. Kafka CPU utilization of the main ordering service considering different number of participants and number of submitted requests

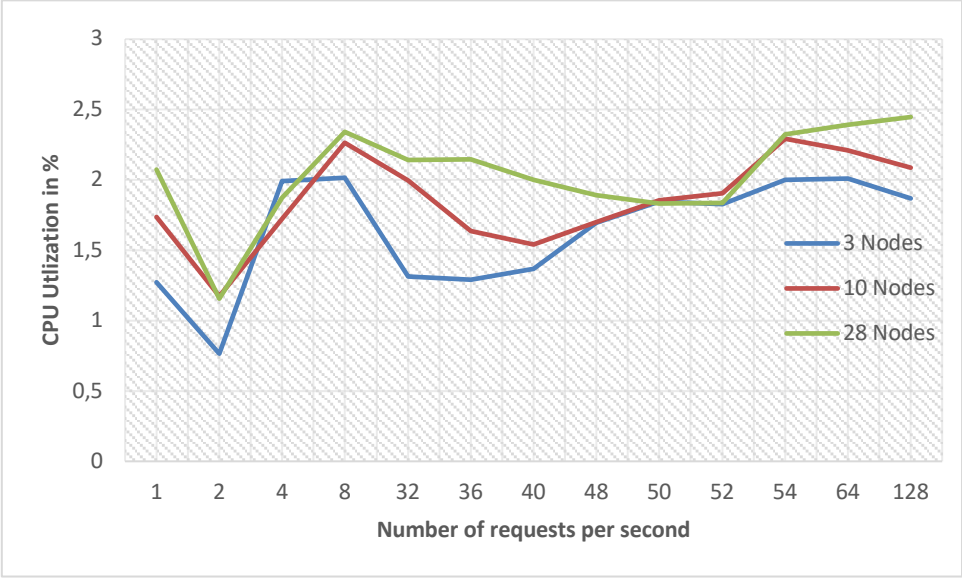


Figure 31. Ordering CPU Utilization of the main ordering service considering different number of participants and number of submitted requests

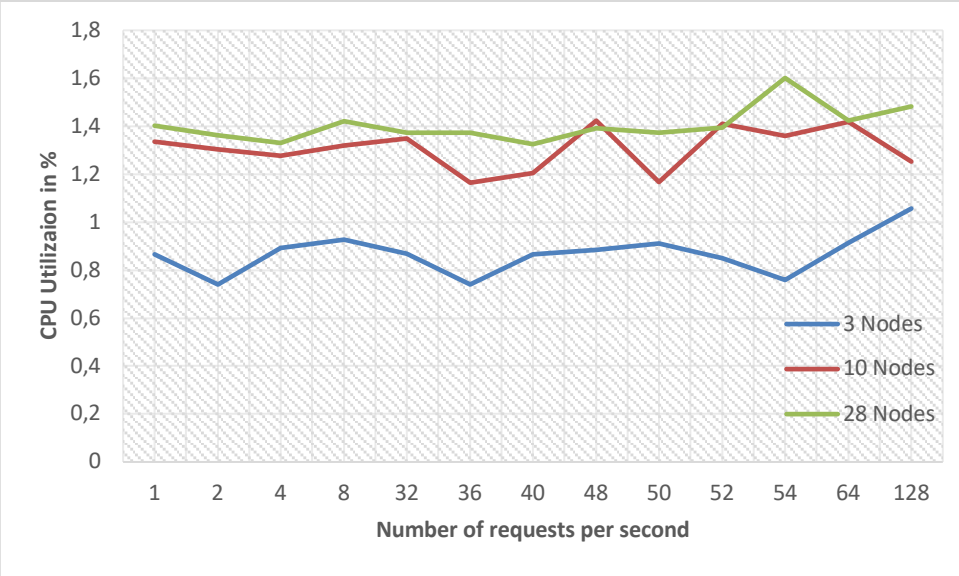


Figure 32. Zookeper CPU Utilization of the main ordering service considering different number of participants and number of submitted requests

2.7.2.3.2 Certificate Authority

Certificate authorities' CPU utilization, both for active and passive nodes, is similar to memory utilization; it is negligible since they do not have any active role for the submitted transactions (see Figure 33 and Figure 34).

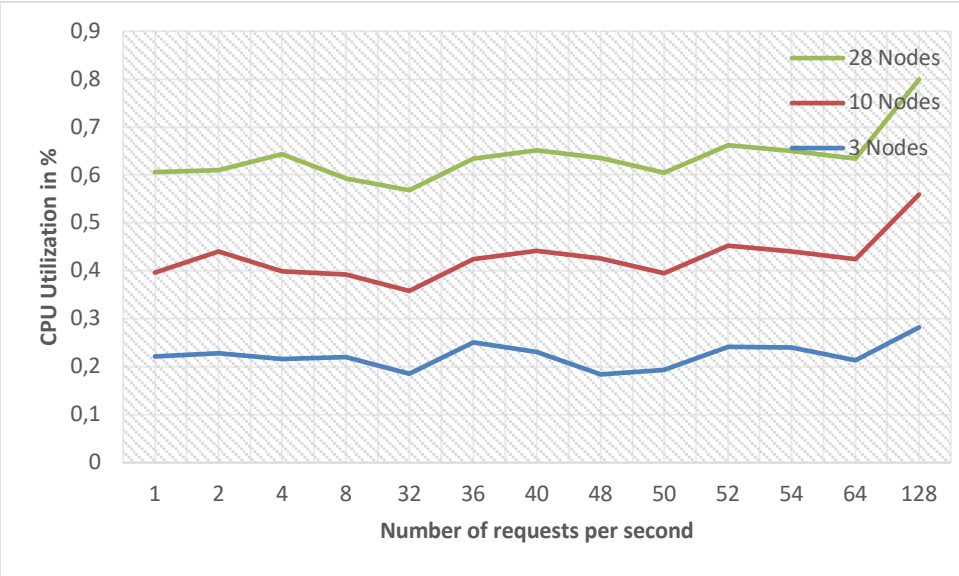


Figure 33. CA CPU utilization for an active node

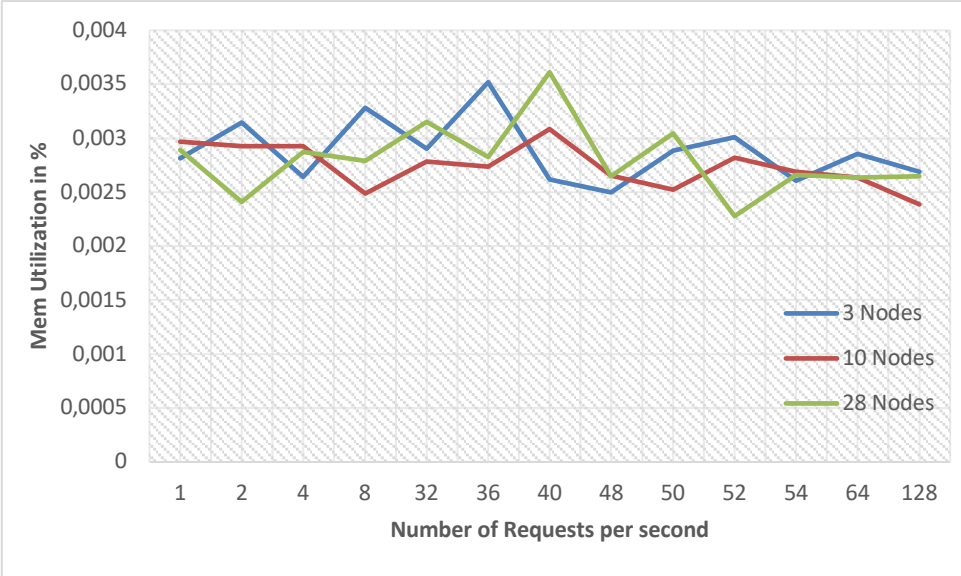


Figure 34. CA CPU utilization for a passive node

2.7.2.3.3 CouchDB

CouchDB CPU utilization, in contrast to memory utilization, is as low as 1.5% in the case of an active node, considering 28 blockchain network participants. For less participants CPU utilization ranges between 1,2% to 1,4% (see Figure 35), whereas for a passive node CPU utilization it is almost the same under all the cases (see Figure 36). Overall, CouchDB in terms of CPU utilization is not impacted by the number of nodes and number of requests per seconds.

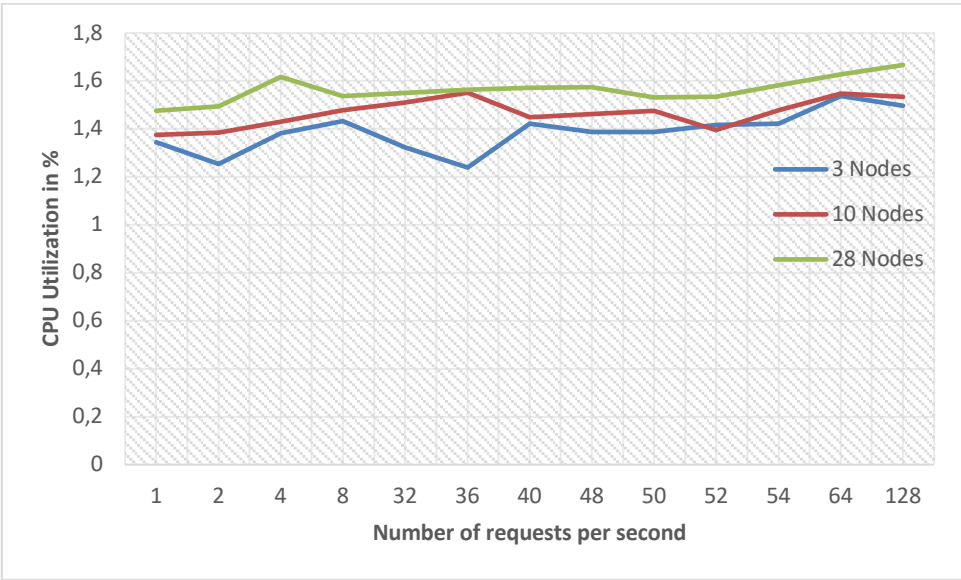


Figure 35. CouchDB CPU utilization for an active node

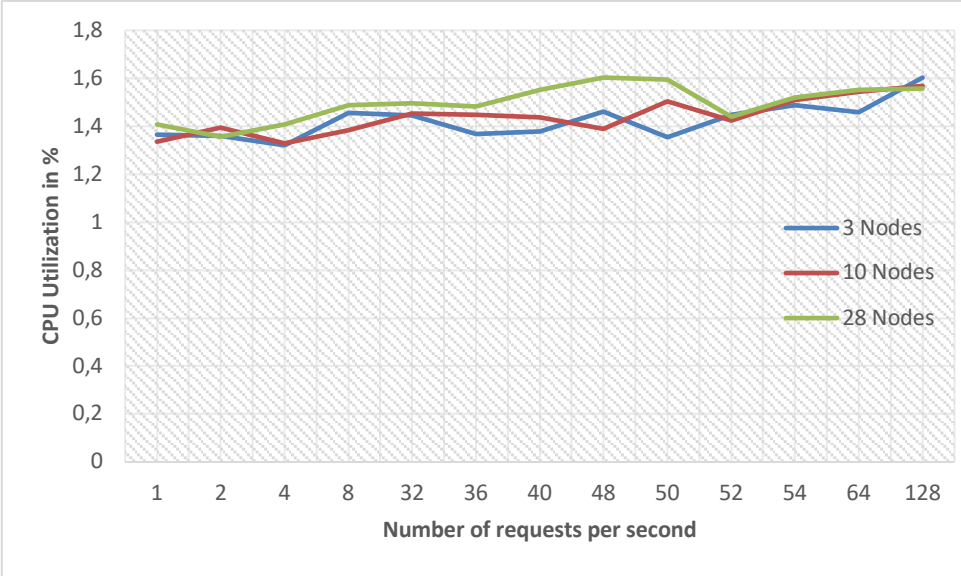


Figure 36. CouchDB CPU utilization for a passive node

2.7.2.3.4 Peer

Peer service CPU utilization ranges between 3% to 6% for an active node (see Figure 37), while for a passive one it varies between 2% to 4,5% (see Figure 38). For both types of nodes the CPU utilization does not depend on the number of the participants, however, the CPU utilization grows as the number of requests increases. For instance, CPU utilization for 2 requests per second approximates to 2.5%, whereas for 8 TPS it reaches up to 5% in the case of an active node (see Figure 37). The same trend is also followed in the case of a passive node (see Figure 38).

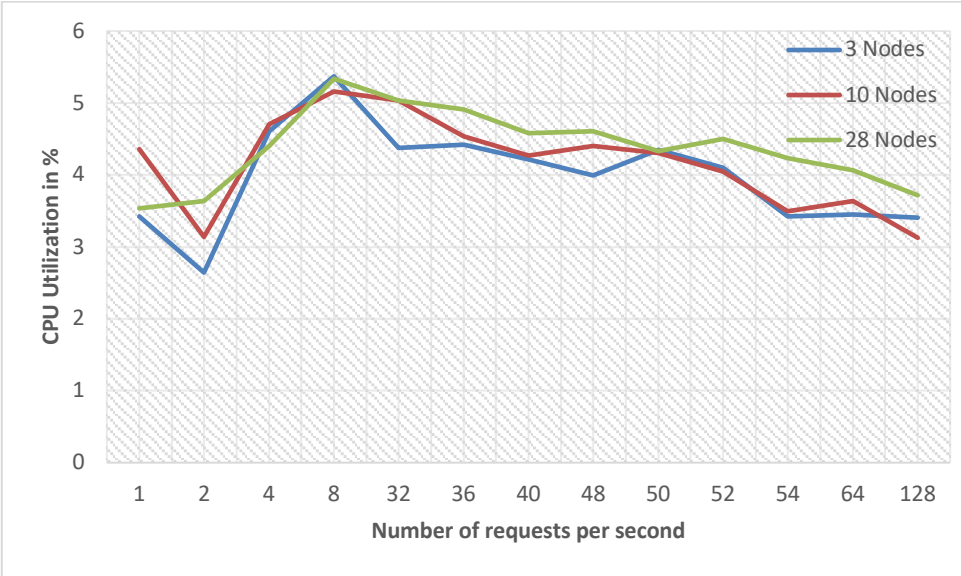


Figure 37. Peer CPU utilization for an active node

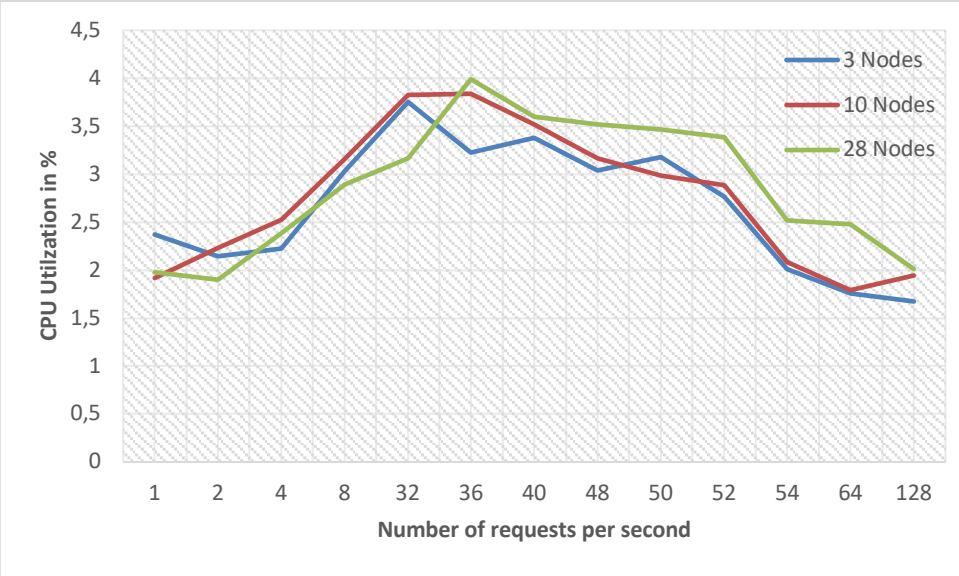


Figure 38. Peer CPU utilization for a passive node

2.7.2.3.5 Client (Application Interface)

Similar to the other participant's blockchain components (i.e., peer, CouchDB, etc.) CPU utilization is influenced only by number of requests per second. For an active node the CPU utilisation ranges between 3% to 5% (see Figure 39), whereas of a passive one it is negligible (see Figure 40).

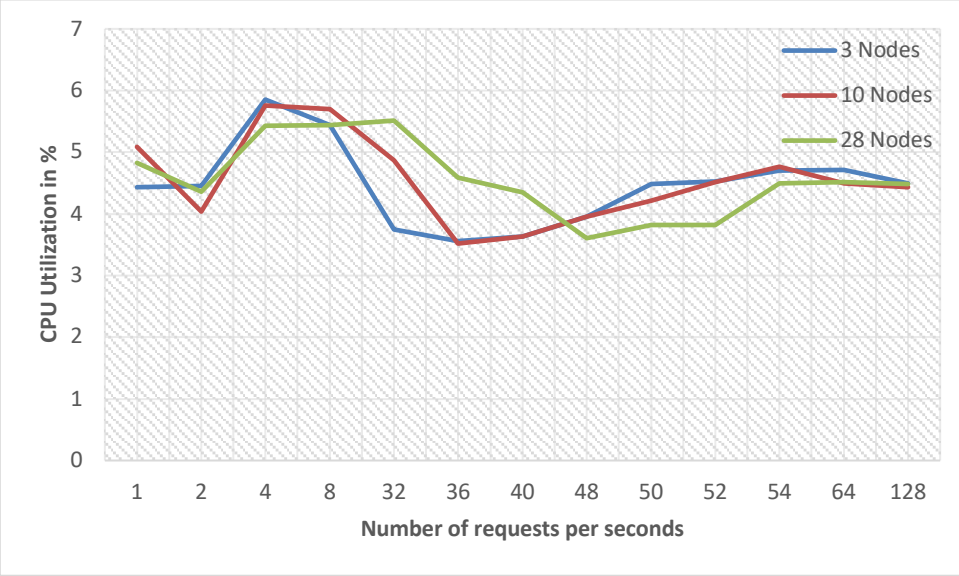


Figure 39. Client CPU utilization for an active node

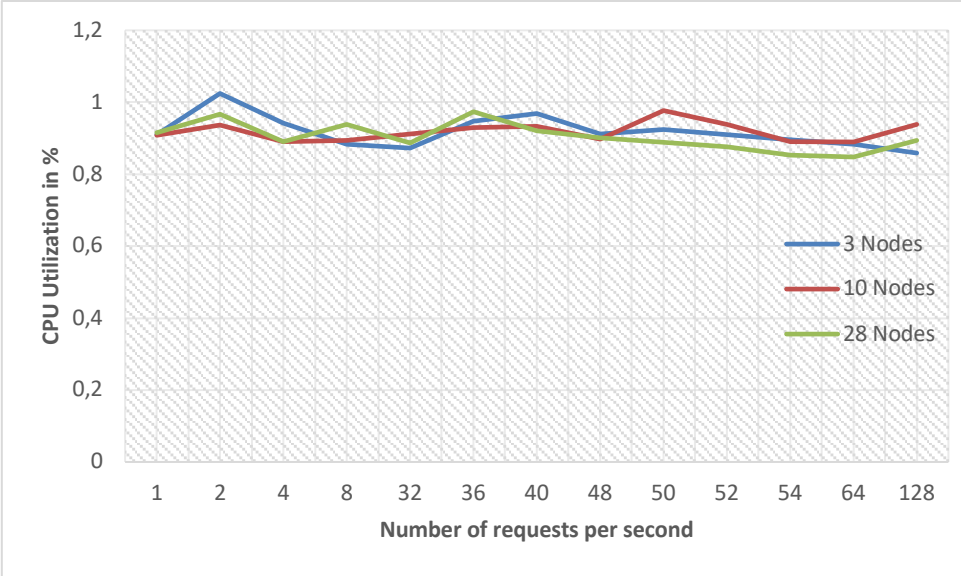


Figure 40. Client CPU utilization for a passive node

2.7.2.3.6 Smart Contract

Last but by no means least, smart contract CPU utilisation as Figure 41 and Figure 42 illustrate is negligible without being affected neither by the number of nodes nor by the number of transactions.

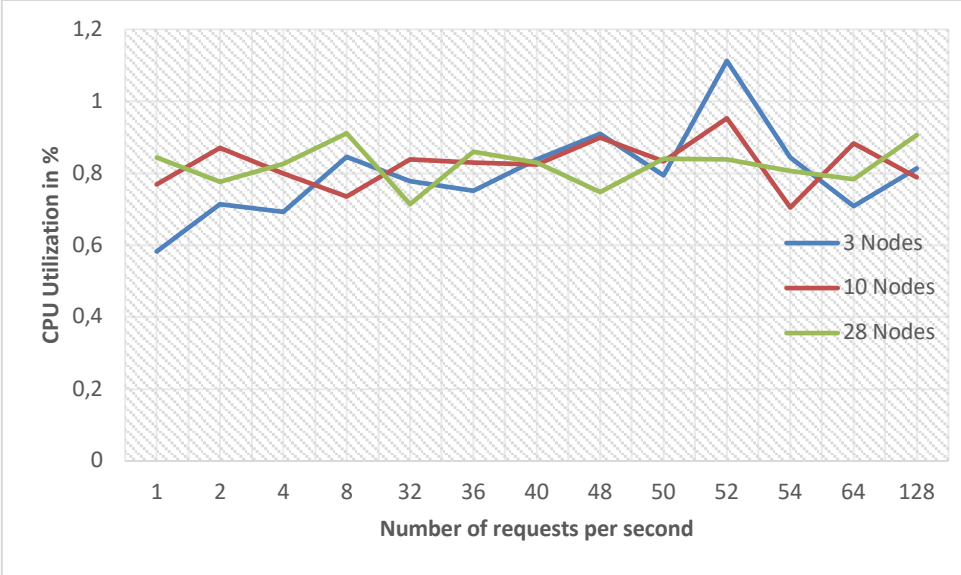


Figure 41. Smart contract CPU Utilization for an active node

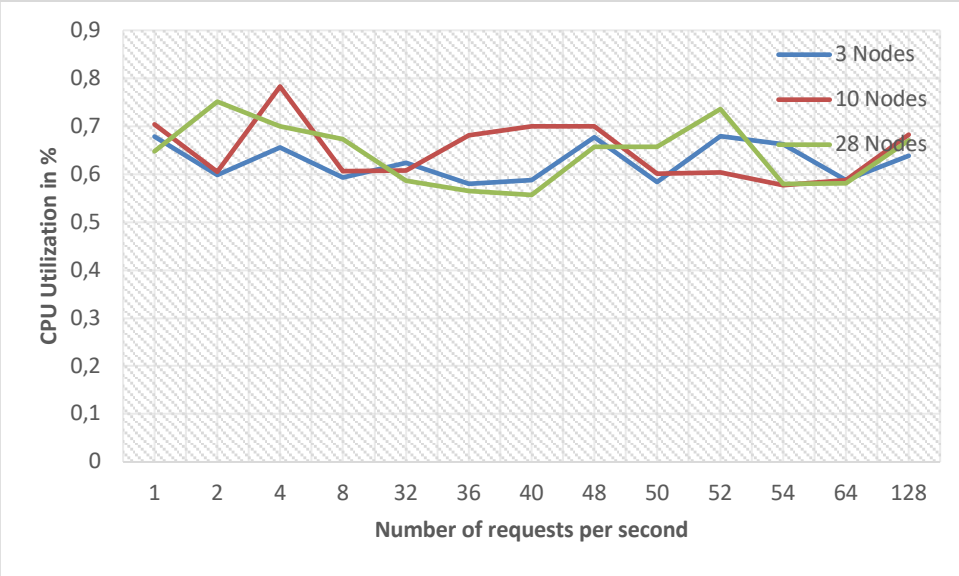


Figure 42. Smart contract CPU Utilization for a passive node

This can be seen by the low utilization in both active and passive nodes, not surpassing in any case around 1%.

2.8 Discussion and implications

The analysis shows a robust and straightforward implementation of the blockchain solution for DR event verification and tracking. The simulations prove that the solution can be scalable to thousands of assets. This can be assumed as the simulations were done on a per second basis, whereas demand response events would typically take several minutes with a minimum of a settlement period (for example 15 minutes). This procedure will facilitate the financial settlement and shorten the time of the settlement process in comparison to what is done presently. It will also allow for a communication of data between the TSO and DSO which is incentivized in the Clean Energy Package (CEP). The adoption of smart contracts and facilitation of DR event tracing will enable the large scale service provision, including medium and large assets, paving the way to citizen engagement and involvement in the energy market.

On one side, results indicate that in general CPU utilization is not significantly influenced by the number of nodes participating on the network, however, the number of submitted transaction per second can mainly affect the CPU utilization of the peer service that under stress it can reach up to 4%. On the other side, all services memory utilization is affected by both the number of nodes and the number of transactions submitted per second in the system and it can be as high as 25% of the available memory.

Moreover the end to end execution time considering different sizes of blockchain network i.e. 3, 10 and 28 participants, and different number of requests per second shows no problem up to 32 requests per seconds. In particular, for a network size with 3 participants up to 51 TPS the total execution transaction time does not exceed two seconds that can be considered as an acceptable response time for end users taking also into account the number of errors. If the system is more reluctant in the end to end delay, for 54 TPS it reaches up to 10 seconds without losing its stability as the number of errors remain low. Of course as we increase the number of nodes the end to end delay increases faster. For instance, in a network that consist of 28 members under 32 TPS the end to end delay approximates 2 seconds, while for 3 nodes is less than 1 second. Lastly, in terms of resources it is worth to mention that the bandwidth is also influenced by both the number of (a) participants and (b) transactions per second submitted to the system, especially the ordering service as it is responsible to share the data with the corresponding entities. Thus, network resources should be defined properly in order to eliminate fundamental operational flaws.

Since a typical DR event lasts no less than 900 seconds (15 minutes) and the seldom expected frequency of requests of such events, the systems would allow thousands of assets to be considered. Furthermore, it should be mentioned that asset power from the same aggregator will be added (aggregated) which decreases the effort requested to the blockchain. These conditions suggest that the use of DLT is more than capable of being implemented in a real world scenario for DR use in terms of event recording.

3 Energy integration from E-mobility Use Case

This use case describes how an e-vehicle performing a charging event with an integrated digital wallet or wallet app, is enabled to make payments on its own. With DLT, payments concerning every aspect of the car's mobility can be executed quickly, securely and automatically. Similarly to the flexibility use case, the energy flow can be from (V2G) and to the car. Typical public normal charging profile is show in Figure 43.

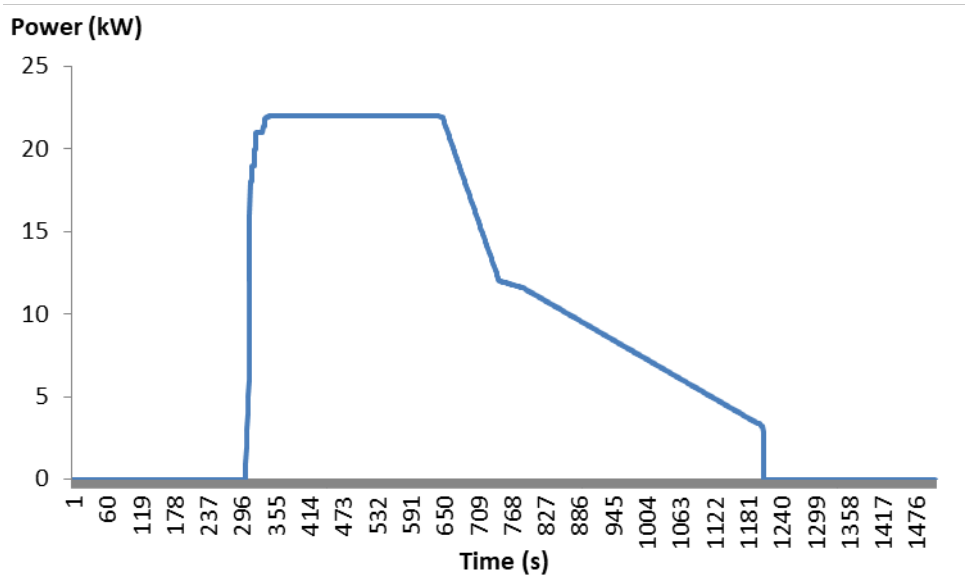


Figure 43. Charging profile of an e-vehicle using a normal charger of 22kW

The charging event is done towards the vehicle, hence charging the battery. It is using a 22 kW charger three phase, and it has an integrated charger meter. In this case the time before/after the event captured for baselining is 5 minutes, due to the nature of the event. The implementation in terms of smart contract and infrastructure is the same as the flexibility use case. The only differences to mention from the flexible use case are the variables to be recorded, which are the following:

- **Power profile**: Active Power for integration over time kW/Energy in kWh
- **User ID**: Car/wallet ID
- **Charger ID**: To allow for party exchange identification
- **Time stamp**: Time of charging in seconds (s)

The vehicles connecting to a charging infrastructure, request or supply energy from/to the grid. The Charging Point Operator acts as an aggregator and sends the transaction at the end of the exchange for publication. Variables to be published on the ledger are the power profile in order to calculate the energy to be built by integrating it over time. The time window of occurrence and the ID of the vehicle allows for various ways of financial settlement. It can gather all information and bill the user at the end of the month or perform it immediately, even if micropayments are to be made. The information is stored on a ledger to enable user profile, behavior, flexibility forecasts and locations (if allowed by the user) and intra-infrastructure owner/technology recognitions. The fact that it contains the ID of the charger also enables the location of a given flexible asset to be known as well as the asset definition. For flexibility provision (V4G) to markets a flexibility resources register will be developed which can be flagged by the User ID. The User ID/charger ID provides access to all the attributes of the flexibility. A list of attributes that can be thought of, although not exhaustive, is the following [2]:

- Direction of deviation (up/down)
- Minimum/maximum duration (e. g. 15 min/60 min)
- Minimum/maximum bid size

- Definition of congestion point (identification of the congested area/locational information provided by the charger)
- Bidding period: time granted to the market parties to
- Offer bids
- Maximum ramping period (15 min, 5 min, …)
- Activation period: time before activation signal and ramp up period (1 h, 15 min, 0 s)
- Mode of activation (automatic, manual)
- Availability window (per day, per week, per year)
- Frequency: maximum number of activations (per day, per week, per year)
- Recovery conditions
- Baseline methodology
- Measurement requirements

An implementation of DLT in this use case, not only allows the verification of energy provided or taken from the grid, but implicitly allows a financial settlement of the simple charging activity. Given a digital identity of the user, an integrated record of all charging activities can be performed regardless of the charging point operator, region or charging type. This would not only facilitate the payment for the user, but also allow an access to all the data generated by the charging activity of EVs both for the users, charging operators and other service providers.

Given a trusted environment of the charging activity, the financial settlement does not need to be subjected to a consensus mechanism constantly. What happens is that after a determined number of exchanges, the network would communicate the result of the given number of transactions as one, similar to lightning networks or side chains. Another aspect to consider is that many of the transactions (financial and non-financial such as data) are expected to be micro-transactions. For this reason micro-payments is one of the characteristics enabled by DLTs provided already by some public projects such as IOTA or NANO DLTs, promoting a feeless machine to machine economy. Vehicles performing charging is just one use, but it can be applied to parking, tolls, shared vehicles, etc.

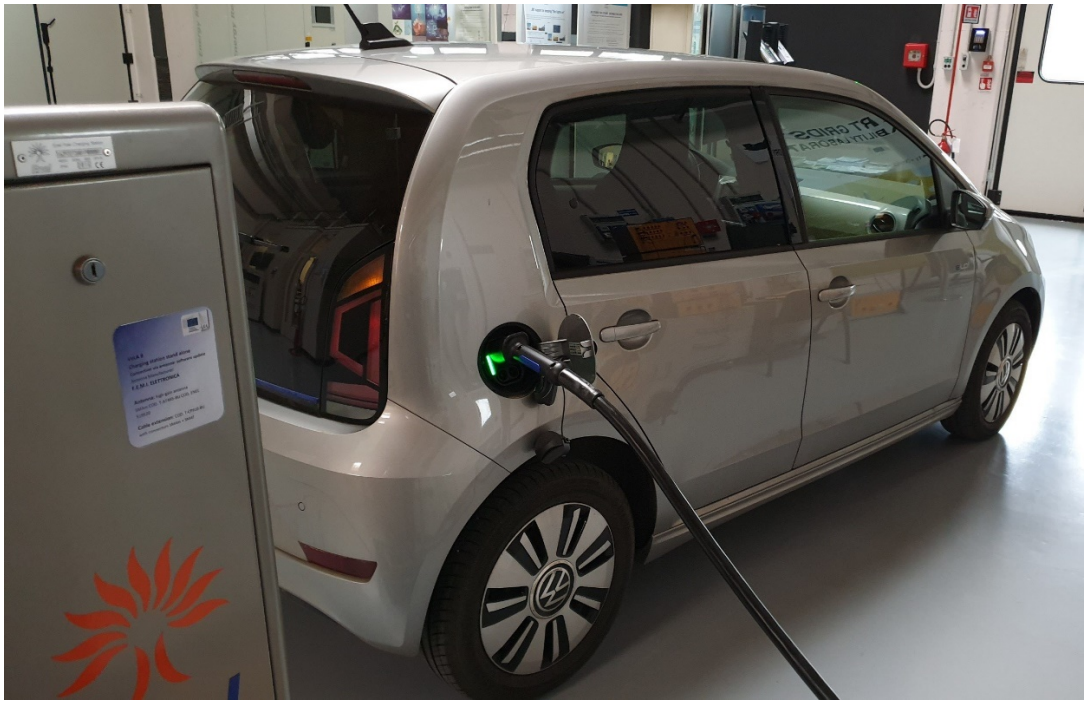


Figure 44. An electric car charging in the SGILAB

4 Energy Communities Use Case

The peer-to-peer electricity community is a use case where a group of prosumers exchange electricity among themselves in an autonomous fashion. The community is autonomous in terms of electricity production, management of the internal operations and exchanges with external power grids. Although the community can be completely independent and disconnected from a network operator (*island* model), in the most general case it is connected to a number of grids in order to interact with the outside world.

Figure 45 displays the general structure of a community. Electrical nodes connect with each other through a common electricity bus. Exchanges to and from the bus are recorded by a smart meter that are placed at the interfaces of each node's infrastructure (household, farm, factory etc.) to the rest of the network. Common meters owned by the community might be installed on the main line for verification purposes. Special-purpose nodes are introduced as gateways to the rest of the world and balancers for insuring stability and reliability of the grid. These balancers are basically flexibility providers, as defined in chapter 2.

The purpose of the energy community is to offer to the end user the maximum flexibility in sourcing its energy needs. Thus a node in the community can decide to buy/sell energy locally within the community or to have access via a gateway to external spot energy markets. The community itself can decide which competitive gateway and balancing services provider to use.

Energy and economic efficiency are the key parameters that can be optimized in this case study. In order to allow and facilitate the reaping of such benefits a decentralised method of account-keeping, information sharing and trusted smart metering must be established in the distributed community. The use of DLT technology is very promising in this context as it can accommodate all these functionalities in a distributed and decentralized manner.

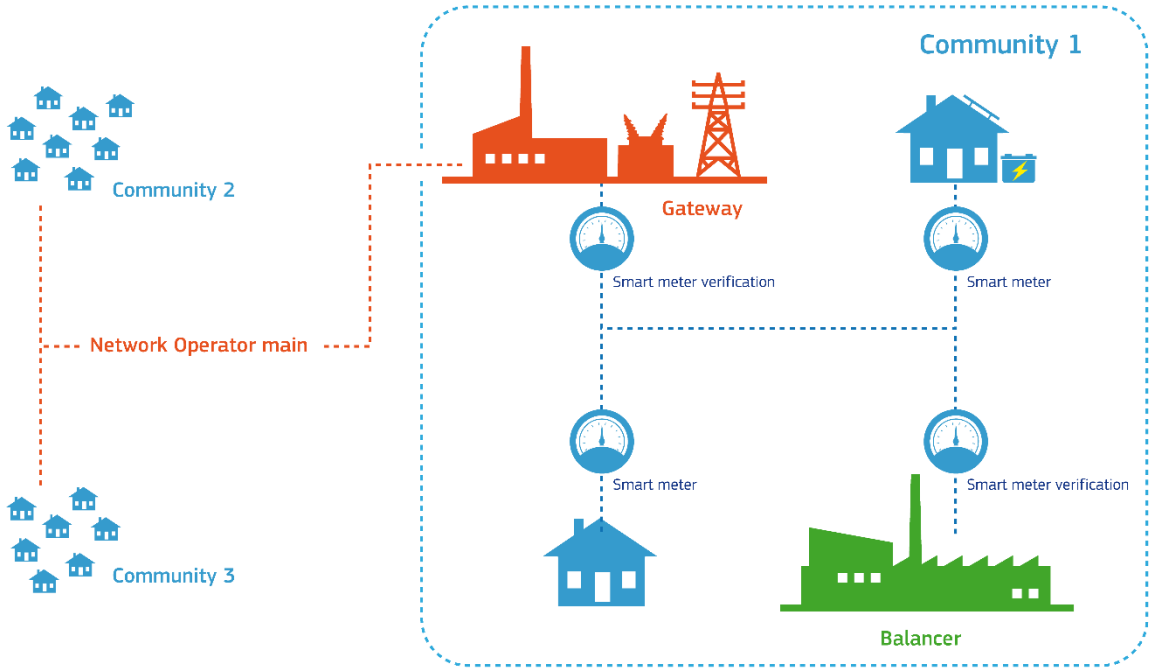


Figure 45. Generic structure of an energy community

4.1 Test setup

We implemented an energy community in our laboratory, composed of five scaled-down houses that have the capability of generating, consuming and storing electrical energy. The five houses or nodes are arranged in a common electric distribution bus. Exactly the same topology is currently in use for energy distribution. The difference, as shown in Figure 46, is the decentralized energy metering and accounting and the use of DLT technology. This technology provides also for metering verification, sharing of telemetry/control information and seamless integration with the outside world. In the figure are visible, in orange the flow of control information

that is necessary to keep the network alive and balanced, and in green the transaction data: these are produced by the nodes and contain untrusted metering information. Part of the importance of the DLT implementation is to establish a set of rules based on distributed consensus to allow fair retribution and billing and also to mitigate the effects of fraud and metering errors.

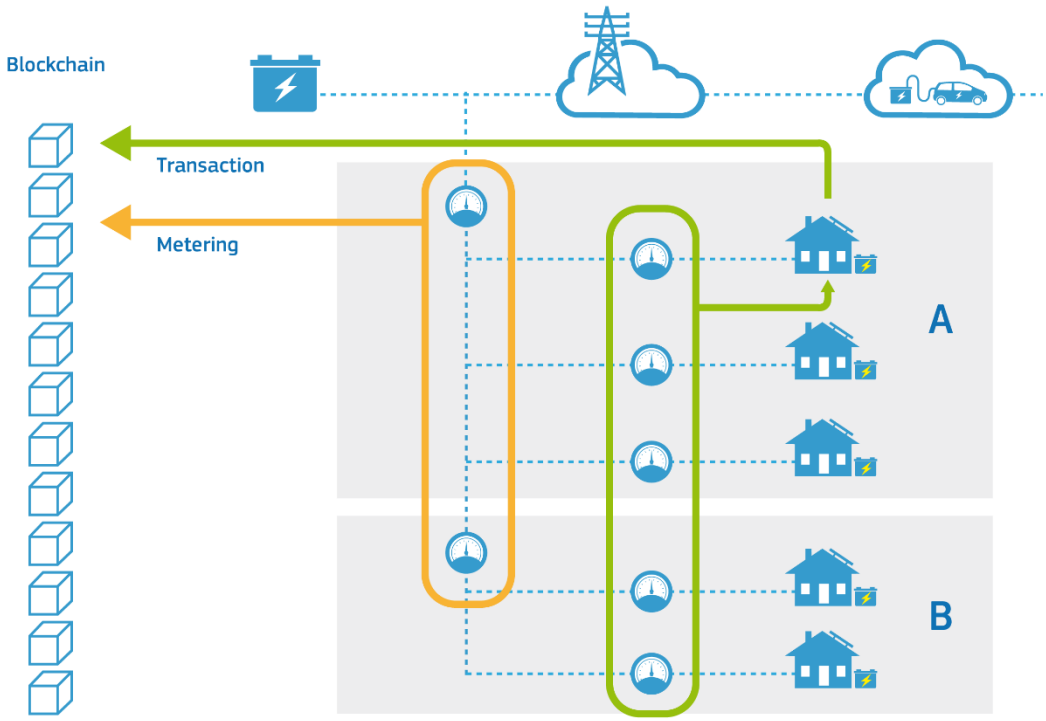


Figure 46. Energy community test bed for the Enerchain project

The ICT network and control sub-system is composed of the following elements, the network diagram of which, showing the interactions of these elements, is presented in Figure 47:

- Smart meters
- Node controllers
- Distributed ledger (Blockchain)

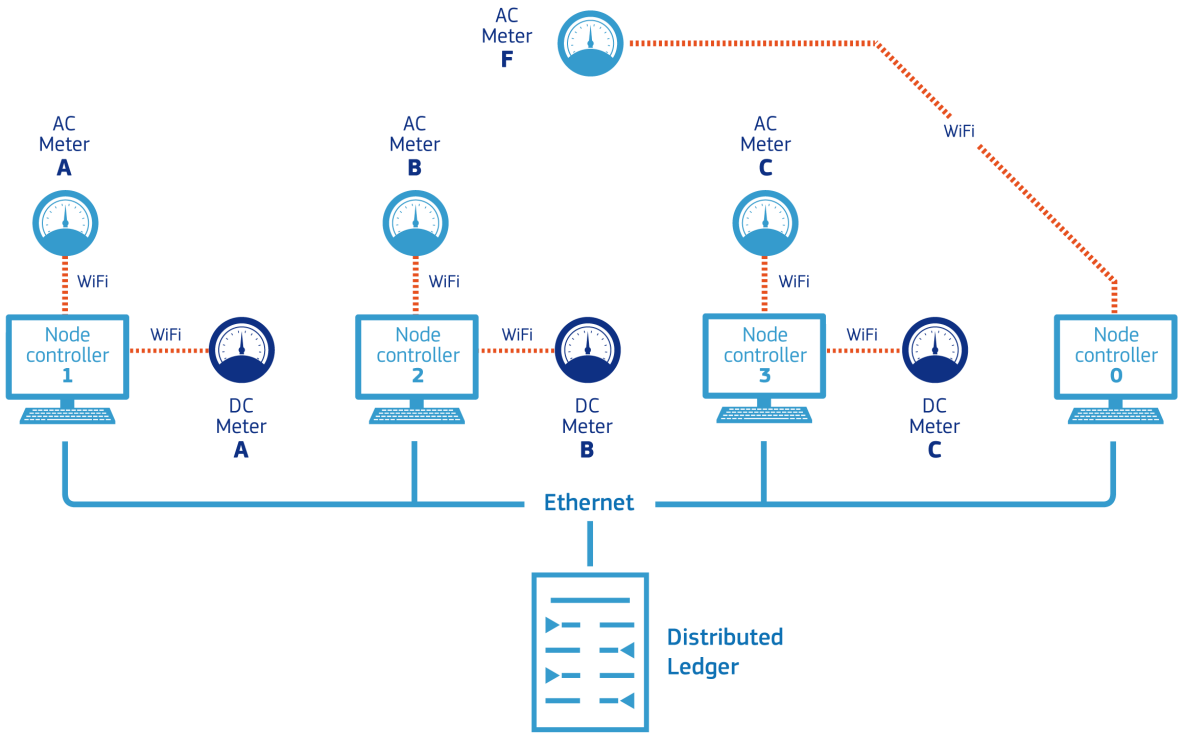


Figure 47. Test bed data network

The node controllers are the key elements of the network, they poll the smart meters for power data, they update the distributed ledger and they control the nodes' behaviour.

4.1.1 Assumptions

A simple set of rules defines the behaviour of the energy community in the laboratory implementation. These rules represent the use case of a peer-to-peer energy community as defined in WP3.

- The energy community has a limited number of nodes sharing a common main frequency and a common power bus.
- Each node can, in general: generate, consume, and store energy.
- Each node declares a maximum power for both injection and withdrawals.
- Each node provides metering on all interfaces to the common bus.
- The community is sub divided in groups. Each group runs trusted power measurements.

4.1.2 Hardware

This paragraph describes the hardware used for the test bed in its evolution. We opted for low-cost hardware for implementing smart meters and node controllers.

4.1.2.1 Smart meters

The smart meters are implemented using Arduino IoT technology. At the core of the smart meters power sensors measure both AC and DC power. In order to obtain power measurement, both voltage and current must be measured. They can be both measured indirectly by a low frequency digitizer using voltage as a proxy. DC power sensors measure current as the voltage drop over a highly precise, low ohm resistor.

AC power sensors use a CT sensor (basically a static voltage transformer) to measure current bringing down voltage to the 0-5 Volts range that is accepted by the digitizer.

Three types of power sensors were tested in the test bed:

6. **Un-calibrated scalar AC current sensors (Arduino Energy Monitoring Shield R2.1)**: they measure RMS current only, in modulus, can only be positive value.
7. **Un-calibrated vector AC power sensors (EmonTx Arduino Shield)**: they measure true RMS power, including vector voltage, vector current, phase angle and various power quality parameters. They are capable of measuring the direction of power flow and can be of positive and negative value.
8. **Calibrated professional AC power sensors (Chauvin Arnoux PEL 103)**: they measure true RMS power, including vector voltage, vector current and phase angle. They are capable of measuring the direction of power flow and they are calibrated and very precise even for small currents.

The final version of the test bed employed only type 2 sensors as they are a good trade-off between the very inaccurate type 1 sensors and the very expensive type 3. These meters, shown in Figure 48 measure true RMS vector power and are directional, meaning that the measured power has a sign corresponding to the direction flow. These meters are capable of handling four CT sensors, thus independently measuring four power channels.

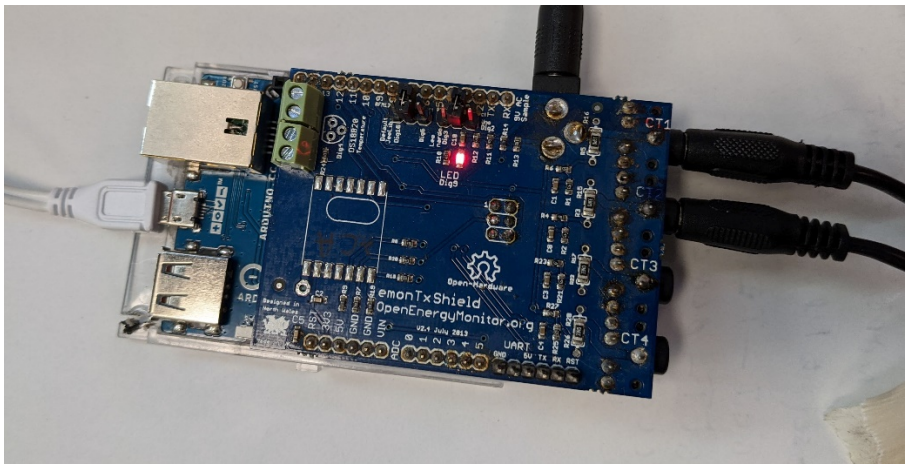


Figure 48. The AC smart meter: Arduino YUN + EmonTX shield from open energy monitor: on the left power supply, up votmetric derivation, right amperometric derivations with CT sensors plugged in

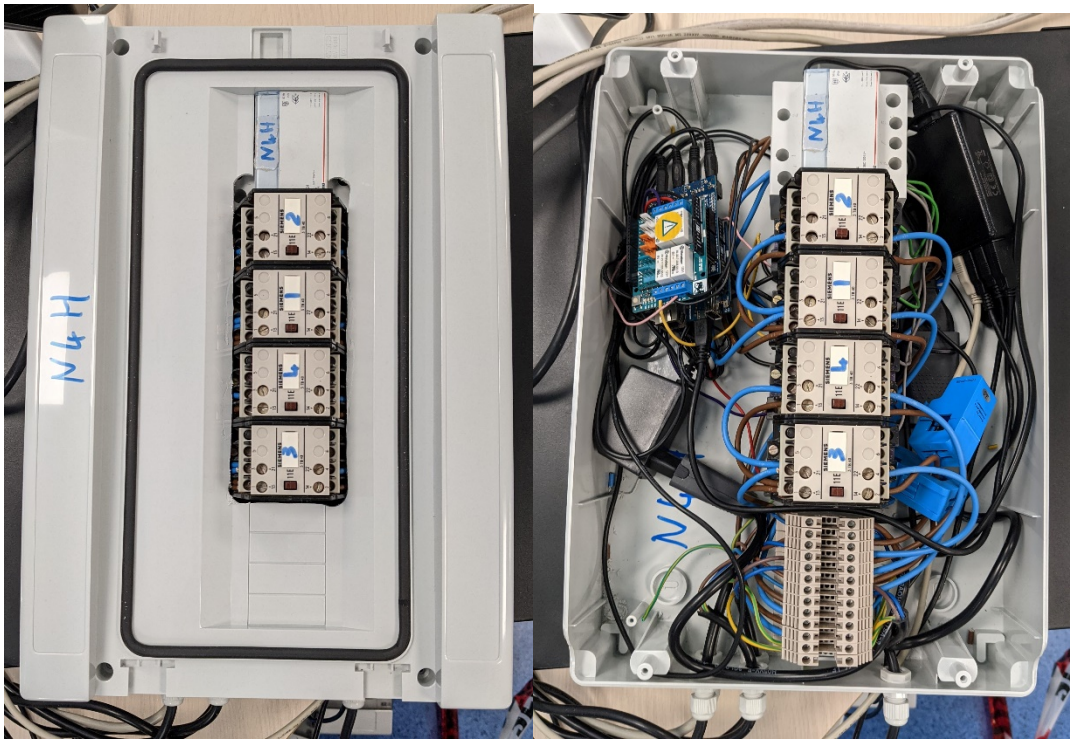


Figure 49. The node guts: meters, relays and auxiliary power supplies

4.1.2.2 Node controllers

Node controllers are implemented using Raspberry PI V3B with Wi-Fi connectivity. The operating system is the standard Debian Linux-based distribution. The hardware is capable of sustaining the control software, the network traffic generated and the interaction with Hyperledger Fabric. The node controller computer is shown in Figure 50, the metering and control subsystems are shown in figure Figure 49, a 130W node battery with DC smart meter is shown in Figure 52 and the node power inverter bank is in Figure 53

Figure 50. The node controller: the heart of the smart house energy system

4.1.2.2.1 Software

The node control software is written in Python and runs on a single board computer powered and maintained by the node.The text interface of the node controller script is presented in figure Figure 51. The node controller

regularly connects to several smart meters and receives the corresponding data complete with a UTP time-stamp. This data generates the energy transactions. The node controller reads directly a number of meters, the relationship between nodes and meters is shown in Table 2. Some meters (green ones in Figure 46) belong to the node. The values read by these meters are subject to validation. Other meters (orange ones) belong to the community and form the basis for transaction verification, they are trusted by all parties by default. Meter ACF is a common (orange) one. In order to further facilitate verification, the community is split in subgroups. Each 'orange' meter caters for one subgroup. In this case with two subgroups there will be two meters: 0-ACF-2 for group A and 0-ACF-1 for group B.

The data produced by the smart meters is stored locally for testing purposes and also in the DLT by the node controller. The interaction is provided by a specific API using JavaScript. The community functionality is guaranteed solely by the data in the distributed ledger. No external oracles or data sources are needed.

Table 2. Nodes grouping and associated smart meters

Node name/ meter name	**NODE 1** GROUP A	**NODE 2** GROUP A	**NODE 3** GROUP A	**NODE 4** GROUP B	**NODE 5** GROUP B
ACA	X				
ACB		X			
ACC			X		
ACD				X	
ACE					X
ACF	**X**	**X**	**X**	**X**	**X**

The control software for the smart meters is written using Arduino IDE using the open source energy metering system Open Energy Monitor[1], with the emonTx hardware. The node control software CLI interface, can be seen in the code below, while a screenshot of the "control panel" that monitors all nodes in parallel can be seen in Figure 51.

```
RELAY 4 OFF
ACB
--CURRRENT--
0.03<-1# 0.05<-2# 1.36<-3# 0.04<-4#
--AVERAGE POWER--
0.52<-MOV AVG 1# 0.17<-MOV AVG 2# 90.90<-MOV AVG 3# -65.70<-4#
--INSTANT POWER--
0.06<-Pcur 1# 0.76<-Pcur 2# 183.27<-Pcur 3# 0.16<-Pcur 4#
--APPEARENT POWER--
7.51<-Pmax 1# 11.90<-Pmax 2# 310.59<-Pmax 3# 9.21<-Pmax 4#
--VOLTAGE-
227.95
227.86
```

[1] https://github.com/openenergymonitor

```
228.21
228.01
Timestamp: 16135798100
Hits so far: 16

RELAY 3 ON
ACD
--CURRRENT--
0.03<-1# 0.15<-2# 1.00<-3# 0.15<-4#
--AVERAGE POWER--
-0.12<-MOV AVG 1# -6.90<-MOV AVG 2# 217.74<-MOV AVG 3# -10.13<-4#
--INSTANT POWER--
-0.65<-Pcur 1# -7.08<-Pcur 2# 218.93<-Pcur 3# -11.23<-Pcur 4#
--APPEARENT POWER--
6.11<-Pmax 1# 32.75<-Pmax 2# 224.52<-Pmax 3# 33.76<-Pmax 4#
--VOLTAGE-
224.81
224.63
224.93
224.72
Timestamp: 16135777242

Hits so far: 297171
```

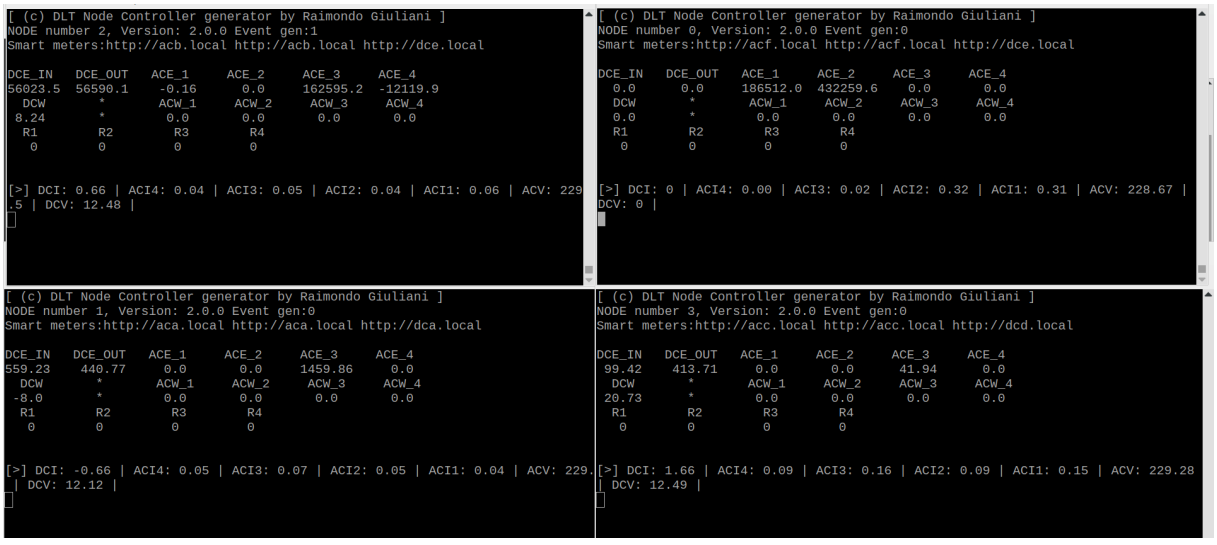


Figure 51. The node control software CLI interface: nodes 1, 2 and 3 plus the common meters

4.2 Experimental tests

The initial test bed network included three 150 W autonomous nodes plus a power generator and a set of fixed loads were observed for a period of time of one year. The observations include readings from power meters as well as AC voltage readings and DC voltage/current measurements of the nodes' batteries.

Once basic automated operation for the energy community was established, the experimental setup was extended in 2020 to accommodate additional smart meters for group-metering and two extra nodes. These have a nominal power of 300W and can provide autonomous main synchronization and balancing.

During the observation period with the new setup 4 different test scenarios were performed in order to evaluate the performance and logic implementation using the Hyperledger Fabric with real transactions and metering data. More details on the logic are given in chapter 4.4, where the implementation of energy transactions and smart-contract based verification is explained.

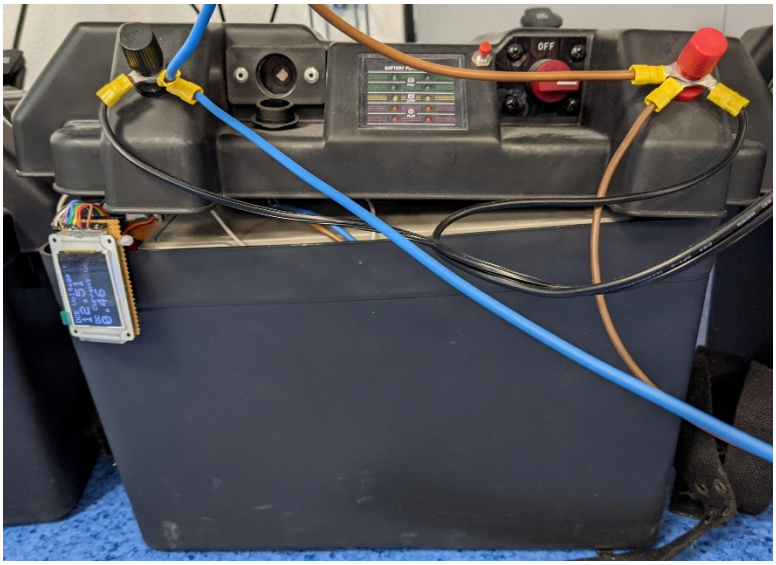


Figure 52. 130 W node battery with DC smart meter

Figure 53. 250 W node inverter and rectifier bank: from left grid-tie inverter, line battery charger, local inverter, solar battery charger

4.2.1 Methodology

The development of the platform focused both on hardware and software tuning and testing. We first set up a node activity controller model to test electrical network stability and self-sustainability for a long period in a semi-autonomous fashion. This model basically decides whether each node in the network is active, i.e. if it is injecting or withdrawing electrical power at any given moment. In the second testing phase we verified the correct operation of the DLT implementation by running fixed simple series of transactions and process them.

4.3 Basic stability and metering accuracy

Using the initial three nodes configuration, five data sets were collected spanning a period of over one year. Data sets are composed of smart meters measurements plus self-reported node activities. The first dataset does not display any energy transaction, it is useful however for establishing a baseline of solar production. These patterns of energy production are useful to identify the nodes' intrinsic energy patterns that in turn can be used for node authentication and node activity verification. An excerpt from the DC activity of each node in this period is presented in Figure 54.

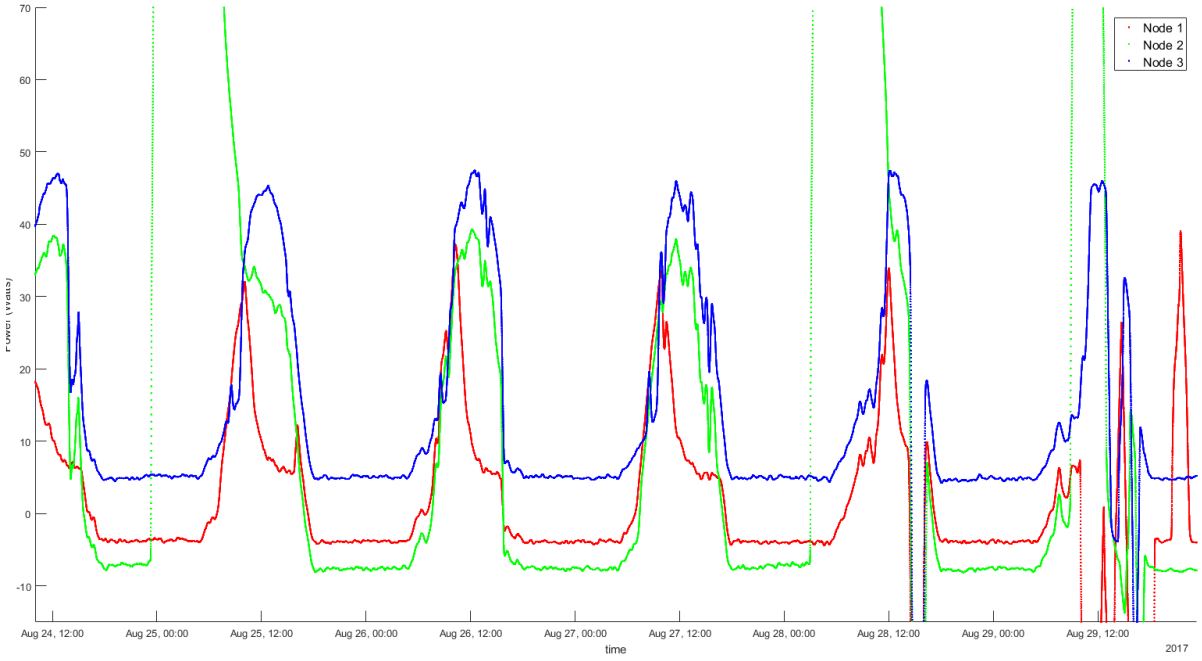


Figure 54. Typical day/night pattern of PV generators for a node

4.3.1 Node activity controller model

During the observation period several parameters that govern the community functions were evaluated: in particular activity rate and poll rate.

Activity rate refers to the AI that operates the test node. Activity rate is the interval in seconds between the finite state machine state changes. In practice the node controller waits a fixed amount of time before making a new decision about which load to activate. There are two programmable 'local' loads for each node as well as a power out (grid tie inverter) and power in (battery charger) loads.

The poll rate is the most important parameter. It is discussed extensively in literature and has important implications for security, privacy, energy efficiency, economic efficiency and cost optimization. The basic trade-off is between accuracy, system stability and users' privacy. The smaller the polling rate the better the dynamic estimation of the system state will be. Less errors mean greater measurement accuracy, higher system neutrality and fairness, more accurate error checking and better protection against malicious attacks. However a small polling rate allows an external operator to perform a very fine profiling of the users' behaviour. Small polling interval also requires more power from the control hardware, more intensive computing and more network activity. Moreover, the stability of the grid control systems can be affected by too many short-term variations.

In our experiment the rate was initially set to 0.5 seconds for the first data set. The smallest feasible value in order not to overload the Wi-Fi network supporting the test bed. In current production environments the poll

rate is typically in the range of 900 seconds. However the system proved to be unstable with this setting with several instances of nodes going dead due to battery depletion. Thus the hardware polling rate was increased to 10 seconds for the following data sets also to reduce loads on the computing/storage side (all node controllers are implemented in a single SBC Raspberry 3). Unlike the poll rate which is common to all power metering systems, the activity rate is strictly related to our node controller architecture and implementation. The finite state machine that controls the node activity is activated after a number of loops defined by the parameter activity rate. The interval in seconds between the machine activation is defined as:

$$T_a = A_r * P_r \text{ [Equation 1]}$$

where

$$T_a = interval;\ A_r = activity\ rate;\ P_r = poll\ rate$$

4.3.1.1 Key parameters

As high frequency access to the power bus and low durations lead to highly inaccurate measurements, after initially setting the sampling rate to 1 second and performing access to the bus manually, we decided first to automate the nodes' access to the energy bus with a decision taken every 10 seconds. Afterwards, the sampling rate was increased to 10 seconds and the bus activity was increased to 300 seconds.

These parameters control the behaviour of each node, based on battery voltages, time of the day and a random seed. The listing of parameters and the figures used for the test bed are presented in Listing 1.

```
#basic parameters
ac_current_th=0.06 #minimum current that ac meters can measure
poll_rate=10.0;# (in seconds)
#
#activity and safety parameters
#
do_something_day_vth=12.6 #min voltage to allow power injection into the grid
do_something_day_vld=12.1 #min voltage to allow local load to be activated <-> battery charger
    activated
do_something_night_vth=12.8 # threshold maximum voltage for activating charger at night
do_something_night_vld=12.7 # threshold minimum voltage to activate local load at night
day_probability=0.5;# probability of electrical bus activity in the day (injection only)
night_probability=0.3; # probability of electrical bus activity in the night (withdrawal only)
#
safety_vth_min=11.7; #min voltage allowed for  battery safety
safety_vth_med=12.0; #warning voltage for  battery safety
```

Listing 1. Key behaviour parameters

4.3.1.2 Control Software

The core functions that define the finite state machine behaviour are presented in Listing 2 and Listing 3. The safety functions are executed every 6 program cycles of the node controller. They override any other system behaviour in order to keep the power section of the node in a safe and stable state. The basic functionality relies on DC readings from the battery. If the voltage drops below a certain threshold, all loads are turned off. If the voltage drops further the battery charger is activated. If the voltage grows above a certain threshold and the current is low, the battery charger is stopped in order to avoid over-charging.

```
  if DCV > safety_vth_min and DCV <= safety_vth_med:
    log_file.write (str(ACtime)+' DC voltage low, shutting down loads'+'\r\n')
    send_command (myurl+'11')
    send_command (myurl+'21')
    send_command (myurl+'41')

  if DCV <= safety_vth_min:
     log_file.write(str(ACtime)+' DC voltage very low, shutting down loads and starting
    battery charger'+'\r\n')
    send_command (myurl+'11')
    send_command (myurl+'21')
    send_command (myurl+'41')
    send_command (myurl+'30')

  if DCV > 12.7 and DCI < 1.5 and charger_relay=='1':
    log_file.write(str(ACtime)+'battery full stopping battery charger'+'\r\n')
    send_command (myurl+'31')
```

Listing 2. Safety function (called every 6 poll cycles)

The activity function is called every 100 program cycles. It is used to control the nodes' access to the power bus as well as to control the local loads. In practice it decides when to inject, to withdraw or to utilize power for local needs. In an energy community the most important aspect is the node's behaviour. Its working can be influenced by many parameters: price of energy, state of the local grid, state of the outside grid, time of day, time of the year, energy demand etc. This in turn can lead to a very complex behaviour and to the need of network optimization. In this stage of the implementation the focus was not on node behaviour but on basic metering, accounting and system stability.

Therefore, in order to generate meaningful events without introducing further complexity a simple AI based on a finite state machine was implemented. It defines two types of behaviour, night-time and day-time. During the night, the grid tied inverter cannot be used and there is a random choice taken every time on whether to activate the local load or the battery charger. Similarly during the day, the battery charger is always off and a random choice is performed on whether to turn on the local load or the grid tie inverter.

```
  mytime=int(ACtime_)
  curr_h=int(datetime.datetime.fromtimestamp(mytime).hour)
  print (curr_h)
  #if current time is between 8 and 20 activate loads
  if curr_h >=8 and curr_h < 20:
    if random.random() > day_probability and DCV_ >do_something_day_vld:
      send_command(myurl+'10')
    else:
      send_command(myurl+'11')
    if random.random() >day_probability and DCV_ >do_something_day_vth:
      send_command(myurl+'40')
    else:
      send_command(myurl+'41')
#if R1_=='0' and R4_=='0' and DCV_<12.1:
    if DCV_<do_something_day_vld:
      send_command(myurl+'30')
    else:
      send_command(myurl+'31')

  else:
#it's night
#shut down grid-tie inverter
    send_command(myurl+'41')
#starting battery charger if voltage is below 12.8 volts with a probability given by
    night_probability variable
    if random.random() > night_probability and DCV_ <12.8:
      send_command(myurl+'30')
      R3_='1'
    else:
      send_command(myurl+'31')
      R3_='0'
#activate load
    if R3_=='0' and DCV_>do_something_night_vld:
      send_command(myurl+'10')
    else:
      send_command(myurl+'11')
```

Listing 3. Activity function (called every 100 poll cycles)

4.3.2 Smart Contract Validation

For a smart contract validation process it would be necessary to verify the exact amount of energy that has been transferred. This has a number of implications and requirements for the data analysis:

- The full state would need to be estimated, especially a reliable real time estimate for the injected or extracted power.
- The measurement noise of all employed volt and ampere meters would need to be under control and below certain limits to enable the former point. This might not be feasible if the local grid is only assembled from simple, low-cost devices that can be very inaccurate.
- In principle, the measurement from all smart meters and devices attached to the local grid should be shared in same way among all users but only locally.
- In order to fully enable the operation of the blockchain smart contracts, the validation process and, therefore, the state estimation needs to be fully autonomous.

4.3.3 Test bed metering verification

The energy/power figures in the blocks are self-reported by each node. The unverified blocks are sufficient for energy metering at the local level. However additional metering is performed at the edges of the network, specifically at the interfacing points with the outside grid. These cumulative readings must be reconciled with the local metering performed by the single node. The main reason for the discrepancy between the single node data and the cumulative reading at the network edges are:

9. Systematic measurement errors
10. Random measurement errors
11. Node/sensor malfunctions
12. Intentional data forgery by one or multiple nodes

The verification process must take into account all the above-mentioned points and try to recover from error or fault situations. In order to perform the reconciliation and to validate the blockchain entries, additional redundant readings from the meters are used.

The next paragraphs describe the data structure for blocks and for auxiliary data, the control equations used for redundant metering verification and qualitative and quantitative considerations about the performance of the data validation approach.

4.3.3.1 Control equations

The most basic form of validation relies on the principle of conservation of energy and grid balancing. At each given time the sum of the RMS power of all the nodes in the local grid must be equal to the total power measured at the edges of the local network. If we consider a finite time interval, the total energy exchange measured by the nodes should be equal to the energy measured at the interface to the grid.

Let's consider a discrete time series with index starting from 1 to Nt. Nt being the total samples in the data set. For a given energy transaction, let's define the start index and end index, corresponding to the initial and final timestamp of the transaction:

$$T_{start} = T_{i_{start}} \text{ [Equation 2]}$$

$$T_{stop} = T_{i_{stop}} \text{[Equation 3]}$$

By definition the measured energy in a transaction is:

$$E_{meas} = \sum_{i=i_{start}}^{i_{stop}} P_i(T_i - T_{i-1}) \text{ [Equation 4]}$$

We can also represent the measured energy as the actual value plus a term containing the measurement errors.

$$E_x = \varepsilon_x + N_x \text{ [Equation 5]}$$

if node x is active on bus during the transaction, otherwise:

$$E_x = 0 \text{ [Equation 6]}$$

The meters used are affected by a relevant measurement error. This can in turn be split into two components: a random error caused by noise and a systematic error caused by bias and varying quality of the sensors used. Often the latter source of error is predominant, especially in connection with heterogeneous power meter made with low cost components. The energy conservation principle can be expressed as:

$$\varepsilon_{edge} = \sum_{n=1}^{NODES} \varepsilon_n \text{ [Equation 7]}$$

In our case, with a test bed of three nodes and just one connection to the main grid, the

energy balance is:

$$\varepsilon_{edge} = \varepsilon_1 + \varepsilon_2 + \varepsilon_3 \text{ [Equation 8]}$$

Replacing terms:

$$E_{edge} + N_{edge} = E_1 + E_2 + E3 + N_1 + N_2 + N3 \text{ [Equation 9]}$$

Let's define the total measurement error as:

$$E_{rr} = N_1 + N_2 + N_3 - N_{edge} \text{ [Equation 10]}$$

This figure can also be expressed as:

$$E_{rr} = E_1 + E_2 + E_3 - E_{edge} \text{ [Equation 11]}$$

The percentage error is:

$$E_{per} = 100E_{rr}/E_{edge} \text{ [Equation 12]}$$

During the initial operations of the test bed, we noticed that the estimation of the grid state is accurate for a small percentage of transactions only. The measurement errors are often in the range of 200 %. In order to validate legitimate transactions affected by large errors, it is necessary to estimate the noise components:

$$N_i$$

Where:

$$1 \leq \mathrm{i} \leq \mathrm{NODES}$$

And:

$$N_{edge}$$

The system has a closed solution only in case of a single node accessing the power bus. For all other cases a heuristic estimation of the noise component is required. The systematic error can be predicted using either a Kalman filter when the system is observable or a machine learning approach with a state predictor based on a classifier.

4.3.4 Results: data validation

A number of basic 'health checks' can be performed even before the actual energy budget calculation. First the sign of the energy budget should be the same for the measured and the calculated values. Second any node cannot be inputting and outputting energy at the same time. Third the energy figure in the blockchain entry should match the figure calculated from the additional measurements table. An example of a validated transaction can be seen in Table 3.

Table 3. An example of a validated transaction

Count	Sign	Node	Start time	Duration	Total Energy (KJ)
144	0 = incoming	2	01-Sep-2017 16:14:55	2840	521,84

Validation data:

- Tot energy: -351.4092 KJ
- N1: 141.4831 KJ
- N2: -520.3740 KJ
- N3: 7.4382 KJ
- energy error: -20.04 KJ
- percent error = 5.7 %

Two main parameters are evaluated: the magnitude of the percent error and its sign. The sign is very significant because a positive percent error means that the metering error is favourable for the network and unfavourable for the node. In this case, using a fixed percentage for the threshold (i.e. 10%fixed or variable depending on conditions) the smart contract can be activated automatically and generate a payment. The leftover balance has to be accounted separately and might generate a periodical settlement for each node. The qualitative time series of the power level involved is presented in Figure 55. The block in question is power-negative for Node 2 and for the whole system. However while node 2 is withdrawing power, nodes 1 and 3 are injecting, so the nodes are actually trading energy. The power required by node 2 is provided by node 1, node 3 and the grid. At the end of the transaction node 3 also becomes power negative.

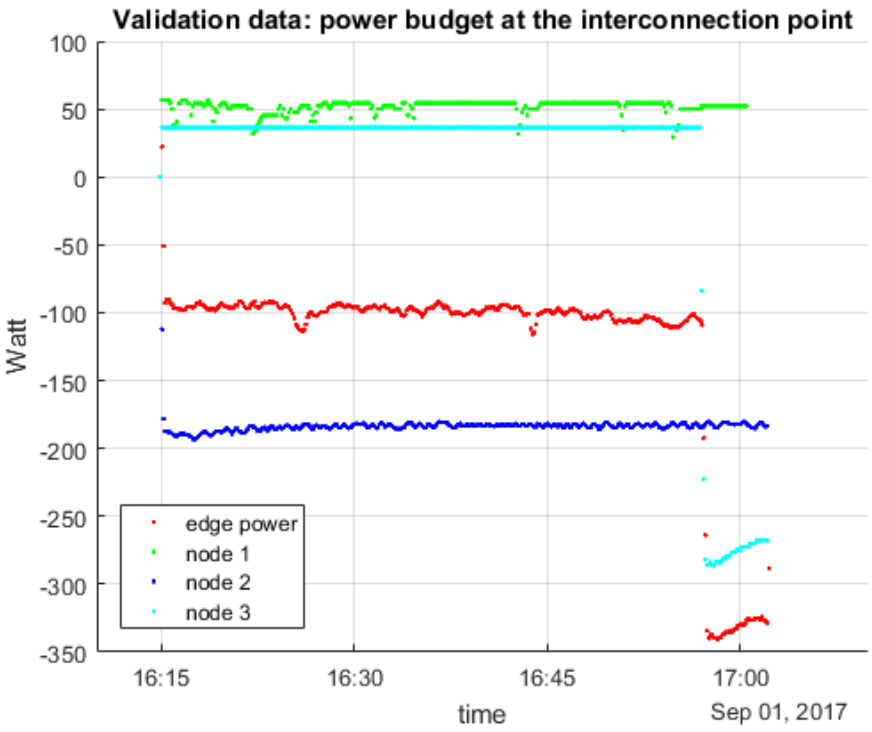


Figure 55. Example of energy transaction validation using trusted meters

4.3.5 Analysis

We evaluated subsets of long-term data series for authentication of smart meters measurements at the physical layer. Figure 56 presents (in red) the evolution in time of the power measurement error as defined in Equation 10. The ideal error would be constant zero, but we can see that it actually varies a lot depending on conditions. In the same plot the self-reported node activities on the power bus are shown in green (withdraw) and blue (injection). Note that the green and blue track show self-reported node activity and not the actual power involved. In the first part where the interval of the activity function is 400 seconds the measurement error is very chaotic and unpredictable due to the continuing transitions or power cycles of the electrical devices. In the second part of the figure, with an activity rate of 3000 seconds, the measurement error is more stable and predictable. During the test bed operation, the activity rate has been optimized in order to have a reasonable amount of transactions without sacrificing too much measurement activity.

Other points that can be observed from the graph are the following:

13. Measurements of single nodes power activity and collective community level measurements can give conflicting results. The red oval with the number 2 in Figure 56 shows a situation where the measurement error is very high and unpredictable.
14. The high frequency access to the power bus and low durations seen in the left part of Figure 56 leads to highly inaccurate measurements and unpredictable rate of transactions' verification.
15. The computed error signals show that the measurement error is correlated with the devices accessing the bus and with the meters employed. The red ovals with the number 4 in Figure 56 for example show two situations where the error is small and were the error is large, depending on which node and which meter is involved.
16. There is a marked improvement in accuracy and consistency of the measurement if the polling rate and activity rate are increased as seen by comparing the left side and the right side of Figure 56.

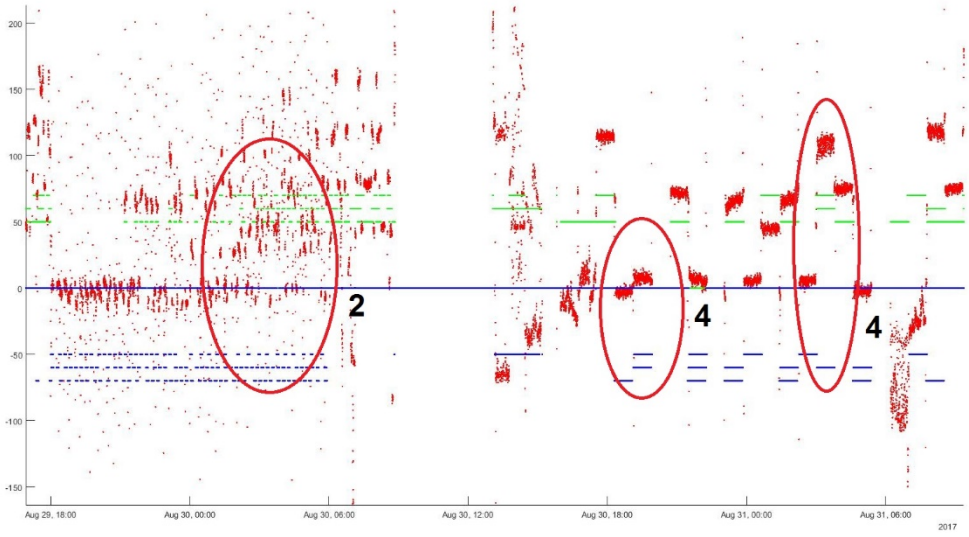


Figure 56. Single node long-term power measurement error vs. network activity

4.3.6 Main findings

A smaller subset of data measurements from the test community is presented in more detail in Figure 57. In red we have the power readings from a single node untrusted meter. In green we have the trusted meter data, measured at the edge of the community, purged from all other nodes' activities. Finally in blue we have the difference between the two: the error signal. The red oval with the number 2 in Figure 57 shows an ideal situation where a smart contract approves a transaction immediately using a deterministic equation (See [Equation 4). The red oval with the number 1 shows a situation where the two measurements do not match but the difference appears to be a systematic and constant error. This can be compensated with proper sensor calibration and metrological improvements. The red oval with the number 3 shows a situation where measurements are offsets by a number that remains constant during the power access but is not constant in time. In this case a further elaboration, probably using artificial intelligence would be necessary in order for the smart contract to provide validation. These points can be deduced from the qualitative analysis of the data sets:

17. A distributed validation scheme for the meter's data is needed for proper accounting and cryptography.
18. Inexpensive hardware can be used for accurate metering and accounting, provided there are redundant sources of measurement.
19. Options remain open for network level authentication and data validation, such as using threshold cryptography.
20. Basic access limitation and control is needed for the power bus, i.e. slotted access, explicit flow control from a master, back off strategies, etc.
21. Usage of machine learning is possible for authentication process and is probably the only viable alternative in presence of very noisy/inaccurate sensors (Figure 57, oval number 3; the pattern is very similar but the values are offset by and unpredictable amount).

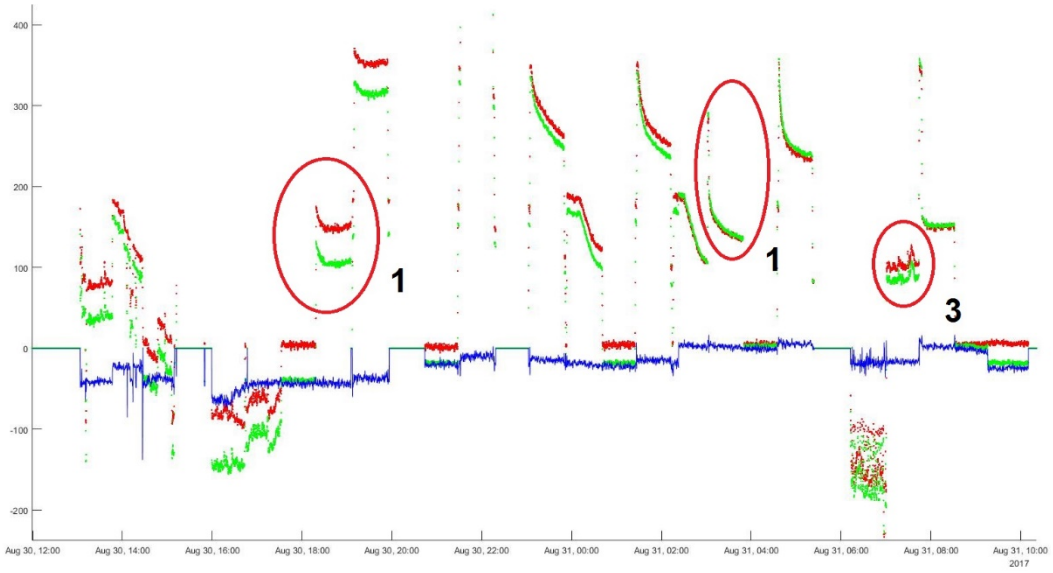


Figure 57. Single node long-term measured power and error

In order to overcome the low accuracy of measurements, focusing on classification approaches that can detect whether power was injected or not, but not necessarily the exact amount would be a viable option. Firstly, such a mechanism might be warranted in any case as part of the grid estimation system, if only for security purposes to raise a warning in case of anomalies. Secondly, if we restrict the local grid operation to only one node operating at a time, a simple injection classification might be sufficient to validate a smart contract process with some legitimacy. The next step in the investigation will look at whether statistical and heuristic estimations are necessary or improving the quality of measurements with better instruments can be sufficient.

4.4 Blockchain implementation

We used Hyperledger Fabric as a blockchain platform, using a smart contract written in JavaScript. We chose this option, as the focus of the energy communities use case implementation is the logic of the system and its real time integration with the actual energy infrastructure that was built in the lab. JavaScript allows for a more high-level flexible logic implementation, giving us the possibility to encapsulate the specific requirements set for this use case, as described in the sections that follow. The clients that trigger the smart contract functions are also written in JavaScript, being invoked with the use of the software node. Finally, the integration with the user nodes of the system, required the execution of the client JavaScript scripts through Python. This was possible with the use of an external library.

Each user node, representing a household, serves as a client to the system and is able to perform blockchain transactions and queries. The client is lightweight and as a result is computationally able to be executed from light devices, such as a Raspberry Pi.

The architecture of the Hyperledger Fabric platform is similar to the one described in the Flexibility use case in chapter 2.1. In fact, we used on purpose the exact same version of Hyperledger Fabric in order to have compatible and comparable solutions. The difference between the two use cases, for what concerns the blockchain part, regards the logic of the smart contract.

4.4.1 Test plan

In order to verify our scenario, we used a constructive approach composed of three test phases:

i. In the first phase smart meter readings are sent directly to the blockchain.

ii. In the second phase transaction verification is introduced, based on the readings of the node. This approach uses a smart contract that checks the energy/power figures self-reported by the nodes against the readings of the trusted meters.

iii. The third phase is the most complete, where energy is verified by the smart contract, taking into account the specific system's architecture and based on the blockchain entries of the nodes and the measurements of the trusted meters.

For all our testing phases, we use the following naming convention: NODE-METER-EQPT. For example 4-ACD-3 means that node 4 owns and operates meter name ACD and it is measuring power circuit 3. Conventionally for each node, power circuit 3 is always power withdrawal from the main bus while circuit 4 is always power injection; these are always positive numbers. The common trusted meters readings are always signed, meaning that they are positive if power if flowing out of the community and they are negative if power is flowing in.

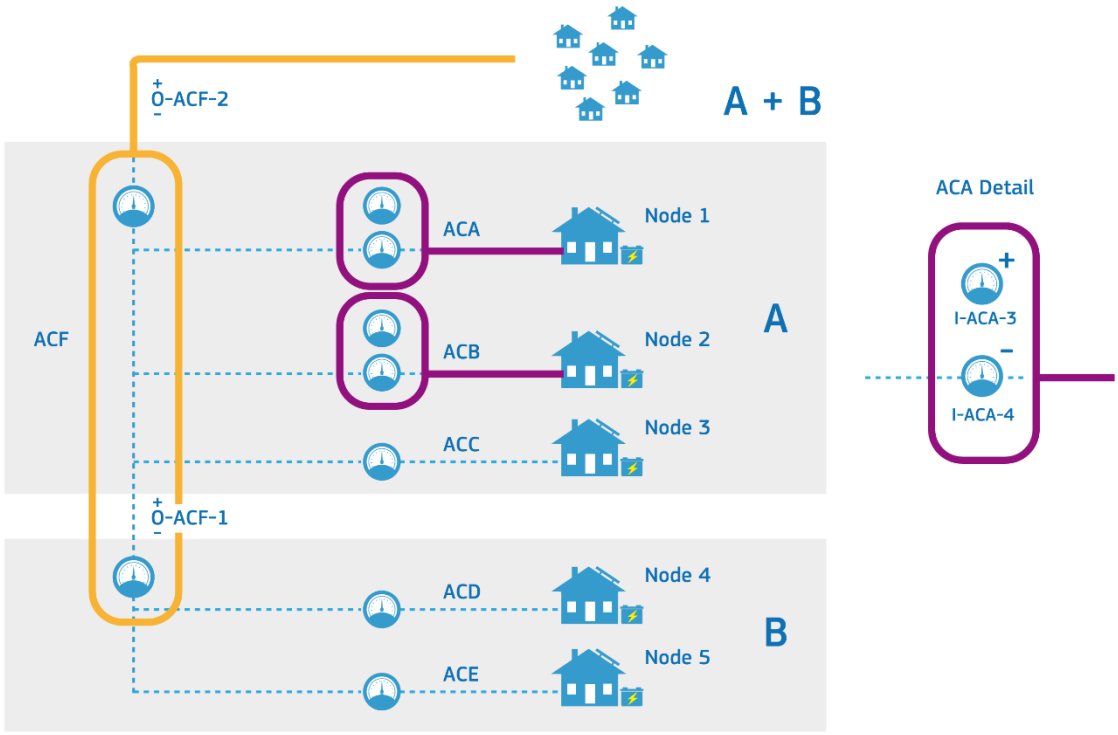


Figure 58. Test bed meters and groups in detail

4.4.2 First testing phase (metering)

In the first phase, we assume five nodes which send energy to the main. Each node is connected to the blockchain network as a client. This means that it is not a full node, verifying and downloading the full blockchain but can communicate with the network by sending blockchain transactions. Every predefined amount of time, the node sends its' smart meter readings to the network in which it registers the energy that has passed through it during this period.

The format of the energy related data in the blockchain transaction has the following fields:

```
energyTransaction = {
        transactionID,
        node,
        startTime,
        endTime,
        type,
        value,
    };
```

TransactionID is a unique identifier referring to the specific transaction, node is the energy node from which the energy transaction originated, startTime is the time that the transaction started at (measured in Unix time using UTC timestamps rounded to the tenth of a second), endTime is the time the transaction ended, type is the type of transaction, and value is the amount of energy measured.

Once the blockchain transaction is broadcasted, the energy transaction is also considered verified and anyone that has access to the blockchain network is able to view it and query it.

This simplified first implementation served as a demonstrator that the current energy infrastructure is able to submit transactions to a blockchain network in parallel with its normal local energy operations. It therefore served as an intermediate step before implementing the more elaborated scenario of the second and third phase.

4.4.3 Second testing phase (transactions)

The second testing phase is more complex in terms of logic. The nodes and the architecture remain the same, as shown in Figure 58. What changes however is the logic of the use case.

When a node sends energy to the network, it now passes to the blockchain all intermediate smart meter measurements. The smart contract:

- is able to read the independent measurements sent by the node's smart meter and thus calculate independently the amount of energy it has sent,
- receives the value that the energy node claims that it sent, and
- compares the two values.

If the difference of the two values is below a given threshold, the energy is considered verified and a new blockchain transaction is sent, indicating this.

In more details, the process starts with the node transmitting to the blockchain network the intention to start sending energy. The blockchain transaction values are in the same format as described in the first testing phase.

At the beginning of the transaction, the start value is set to the timestamp of the first meter reading. The tranasctionID set in this first transaction, will be the same for all the energy measurements that will follow. StartTime is the UNIX time in which the transaction started, while the endTime is the timestamp of the last meter reading. Finally, the node value is the name of the node from which the transactions and the energy originated. Meter is the smart meter owned by the node and eqpt is the circuit measuring the current.

Afterwards, a series of energy measurements are sent in the same format. In this series of transactions, the transactionID and the node are the same. Each measurement has a different startTime and value. Their type is changed to "w", in order to indicate that the type of value sent is in Watts.

The series of transactions conclude with a final transaction that has the "type" set to "end". Moreover, this final transaction indicates in the "value" field the total amount of energy that the energy node claims to have sent to the grid.

All the above transactions are registered in the blockchain and are therefore accessible by the other blockchain peers as well as the smart contract. The next step is for the energy to be verified by the smart contract.

Once the "end" transaction is received, the validation process starts automatically. The smart contract retrieves all the blockchain transactions related to the specific energy transaction and starts calculating independently the total amount of energy sent. The amount of energy is calculated using the formula already detailed in 4.3.3.1:

$$E_{meas} = \sum_{i=i_{start}}^{i_{stop}} P_i(T_i - T_{i-1}) \text{ [Equation 13]}$$

Basically, the smart contract calculates the time difference in seconds between two consecutive energy transactions, and multiplies the time with the "W" value. It then sums up all the individual energy measurements. If the sum is below a given threshold compared to the declared amount of energy as set by the energy node, the transaction is considered verified. In that case a new blockchain transaction is created stating the energy node that sent the energy, the total amount of energy injected, the energy transaction ID and the timestamp of the validation.

4.4.4 Third testing phase (blockchain verification)

The third testing phase is the most autonomous one in terms of automatic transactions' verification. The system does not anymore consider the node measurements trusted, but relies on the measurements of the 0-ACF smart meter in order to verify them.

As it can be seen from Figure 58, there are two trusted meters in the system (in the orange circle): 0-ACF-2 and 0-ACF-1. 0-ACF-2 measures the transactions from nodes 4 (smart meter ACD) and 5 (smart meter ACE), while 0-ACF-1 measures the transactions from 0-ACF-2 (and thus indirectly the measurements of node 4 and node 5), plus the transactions from the other three nodes, i.e. nodes 1 (smart meter ACA), 2 (smart meter ACB), and 3 (smart meter ACC).

The transactions originating from the nodes have the same format as in the previous phases. Moreover, as previously, the transaction has a "start", an "end" and multiple "w" transactions. However, apart from transactions from the five "user" nodes, in this phase transactions are continuously sent in the blockchain from the trusted measurements of the smart meters 0-ACF-1, and 0-ACF-2. Another difference in this phase is that, the smart contract automatically upon receiving a "start" transaction from a node creates a new blockchain entry where it states that this transaction is currently open.

This blockchain entry has the following format:

```
const energyStatus = {
    transactionID,
    node,
    status,
    sumEnergy,
    verificationTime,
};
```

TransactionID refers to the transactionID of the energy transactions that the node is sending, node is the name of the node and status is the current status of the specific energy batch of transactions. The status can be open, pending, verified, or rejected. SumEnergy is not used at this phase, but will be later used, when the specific batch of transactions will be verified to indicate the total amount of energy. The same applies for the verificationTime which is a timestamp of when the transactions were verified.

Therefore, when the start transaction of the node is read by the smart contract, a "open" status blockchain entry is created for this specific batch of energy transactions. Similarly, when the smart contract reads the "end" transaction sent by the same node and with the same transactionID, it will change the status to "pending". This batch of transactions is now ready to be verified when the proper conditions are met.

Each time the status of a transactionIDs series enters in "pending" phase, indicating that the series is considered final from the node's perspective, the smart contract checks if there are other transactions open or pending in the system. If there are other open transactions, i.e. transactions are still ongoing, no further action is taken; the system will wait for these transactions series to be sent from the node. If, however, there are no other open transactionID series, but only pending ones are in the system, the verification process is initiated.

The verification follows a bottom-up approach, starting from smart meter 0-ACF-1 that is able to verify the transactions submitted by nodes 4 and 5. Then once the transactions from these two nodes are verified, the verification continues with the upper layer of the smart meter 0-ACF-2. This node is able to verify nodes 1, 2 and 3 taking into account the verification that has already occurred at 0-ACF-1.

To better show how the verification process if carried out, we have distinguished the following four test scenarios. All scenarios are based on real measurements that were performed in our lab infrastructure.

4.4.4.1 Two nodes withdrawing energy, under the same trusted meter.

In this scenario, nodes 4 and 5 withdraw energy at a given time, which also overlap (Figure 59).

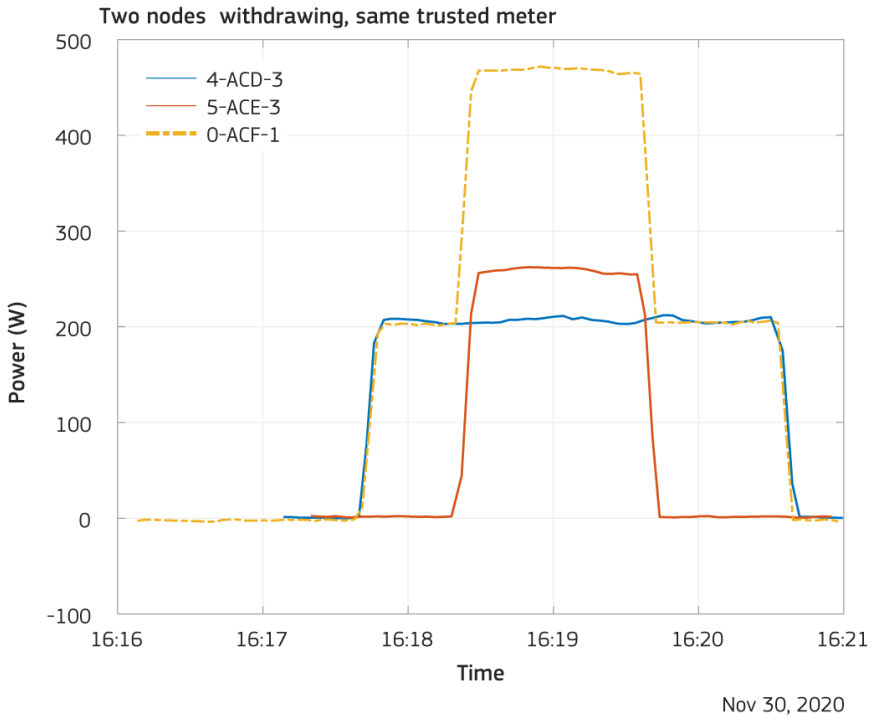


Figure 59. Test transaction 1: two nodes withdrawing, under the same trusted meter

What happens at the smart contract level is the following, following the sequence as seen in the figure:

i. Measurements from 0-ACF-1 are continuously stored in the blockchain, also before the transactions from nodes 4 and 5 start.

ii. Once the transactions from node 4 start, the node starts sending them to the blockchain. As mentioned previously, the first one is the "start" transaction, followed by several "w" measurements, and by the "end" transaction after it has stopped withdrawing energy. An incremental unique transactionID is given to this series, for instance t2.

iii. In parallel with step ii, once the smart contract receives the "start" withdrawing transaction from node 4, with a transactionID t2, it creates a status entry in the blockchain with the values:

```
{transactionID: t2, node: node4, status: open, sumEnergy: 0,
verificationTime: 0}
```

iv. While node 4 is sending the t2 "w" transactions in the blockchain, node 5 starts sending transactions to the blockchain as well. Let's hypothesize the that transactionID for these transactions is t5.

v. The smart contract will detect the transactions t5 from node 5, and will create a new status entry in the blockchain:

```
{transactionID: t5, node: node5, status: open, sumEnergy: 0,
verificationTime: 0}
```

vi. Node 5 will stop withdrawing energy, and thus the node will send the "end" transaction in the blockchain.

vii. The smart contract will detect this, and change the status of the node 5, t5 transactions to:

```
{transactionID: t5, node: node5, status: pending, sumEnergy: 0,
verificationTime: 0}
```

viii. Since a transaction has now been changed to pending, the smart contract checks in the system if other transactions are at this moment open. As t2 from node 4 is open, the smart contract does not proceed at this stage with the verification.

ix. The same process when the withdrawal ends, will happen with node 4, and the transactions t2. Node 4 will send the end transaction, and the smart contract will change the status of the transactions to

```
{transactionID: t2, node: node4, status: pending, sumEnergy: 0,
verificationTime: 0}
```

x. The smart contract now checks again in the system for open transactions. Now, there are none, and since there are two pending transactions, the verification process begins.

xi. The smart contract calculates independently the amount of energy withdrawn by node 4, the amount of energy withdrawn by node 5, and the amount of energy that the 0-ACF-1 smart meter detected. It then compares the sum of energy from node 4 and 5, with that measured by 0-ACF-1. If these two numbers match, with a certain degree of tolerance, the energy transactions t2 and t5 are considered verified. Their status will now change to verified in the corresponding blockchain entry and the energy and verification time will be added to each entry, using the values mentioned above.

It should be mentioned here, that in order to calculate the corresponding energy that 0-ACF-1 measures is not as straightforward as in the case of independent transactions from the user nodes. In the measurements from both 0-ACF-1 and 0-ACF-2, there is no start and end transactions, as the measurements are continuous during the operating of the system. What happens however, is that the smart contract finds the smallest and largest timestamp from the user node transactions that need to be verified, and then identifies the closest measurements to those timestamps from the readings of the 0-ACF meters. By doing so, it sets its own "start" and "end" period for every independent validation that needs to be carried out, not only for this testing phase but also for the ones that follow below.

4.4.4.2 One node withdrawing, one injecting energy, under the same trusted meter

The logic of this second test scenario, is similar to the first one with the difference that node 4 is injecting, while node 5 is withdrawing energy, as it can be seen in Figure 60.

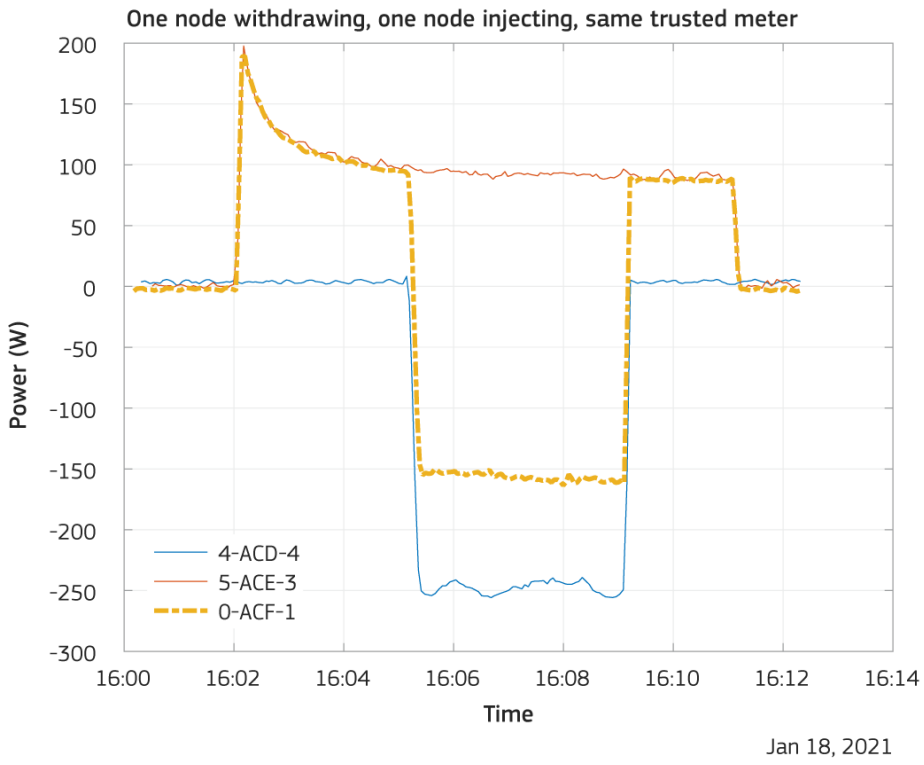


Figure 60. Test transaction 2: one node withdrawing, one node injecting, and same trusted meter

The logic of the operations, as described for the first test transaction do not change. What changes in practice is that the transactions of node 4, will have a negative sign. However, the formulas and the verification process are exactly the same, leading to have both transaction series verified.

4.4.4.3 Two nodes withdrawing energy, under different trusted meters.

In this third scenario, the process becomes more complex. We may have once again two nodes withdrawing energy, they are however located under different trusted meters. The transactions are overlapping, with that of node 4 being longer in duration (Figure 61).

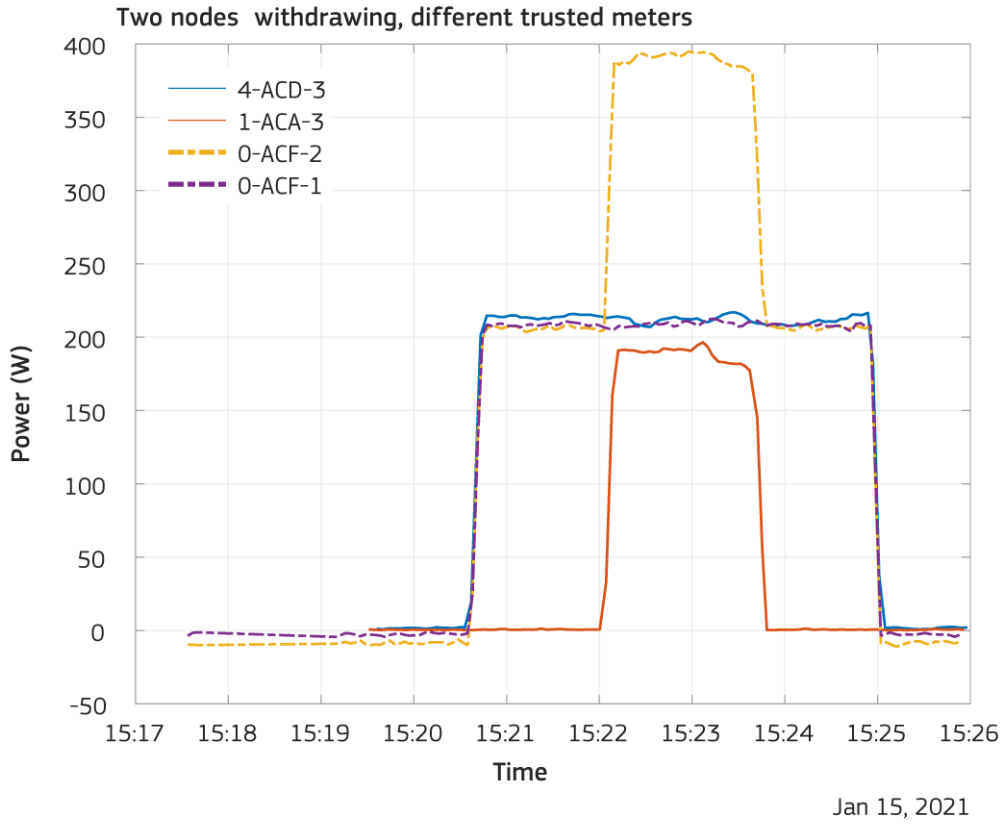


Figure 61. Test transaction 3: two nodes withdrawing, different trusted meters

Without going into the detailed description for the common steps, as explained in the first test transaction, the operations happening in the smart contract are the following:

i. The smart contract, in a similar fashion with the one described in the first test transaction, creates a "pending" status for the transactions of node 4 and node 1, once they are finished. For the needs of this explanation, we can assume that the transactionID of the node 4 transactions is t8, and the one from node 1 is t10. We will therefore have in the blockchain the following two entries:

```
{transactionID: t8, node: node4, status: pending, sumEnergy: 0,
verificationTime: 0}
{transactionID: t10, node: node1, status: pending, sumEnergy: 0,
verificationTime: 0}
```

ii. The smart contract will now start with the verification. However, it will not take all transaction together at one step. It will use a bottom-up approach, starting the verification from the 0-ACF-1 smart meter, and thus by verifying the t8 transactions of node 4. The verification is done in the same way as previously described: the smart contract independently calculates the energy as it was calculated by 0-ACF-1, and as transmitted by node 4. Since in this scenario, there are no transactions from node 5, there is no need to sum the independent node energies. The two numbers that have been calculated are compared, and if the comparison is below a given threshold, the energy transactions t8 from node 4 are now verified.

iii. The verification proceeds at the upper layer, controlled by the smart meter 0-ACF-2. The logic is once again similar to the one before. The smart contract independently calculates the energy from node 1, and from the 0-ACF-2 smart meter readings. However, it now compares the sum of node 1 and the

energy previously calculated for 0-ACF-1, with that of 0-ACF-2. If these two sums are under the predefined margin of error, the energy transactions t10 from node 1 are also considered verified.

4.4.4.4 Three nodes withdrawing energy, under different trusted meters.

This last transaction test is the most complex, as it foresees in parallel transactions occurring in both trusted smart meters, from three user nodes in total (Figure 62).

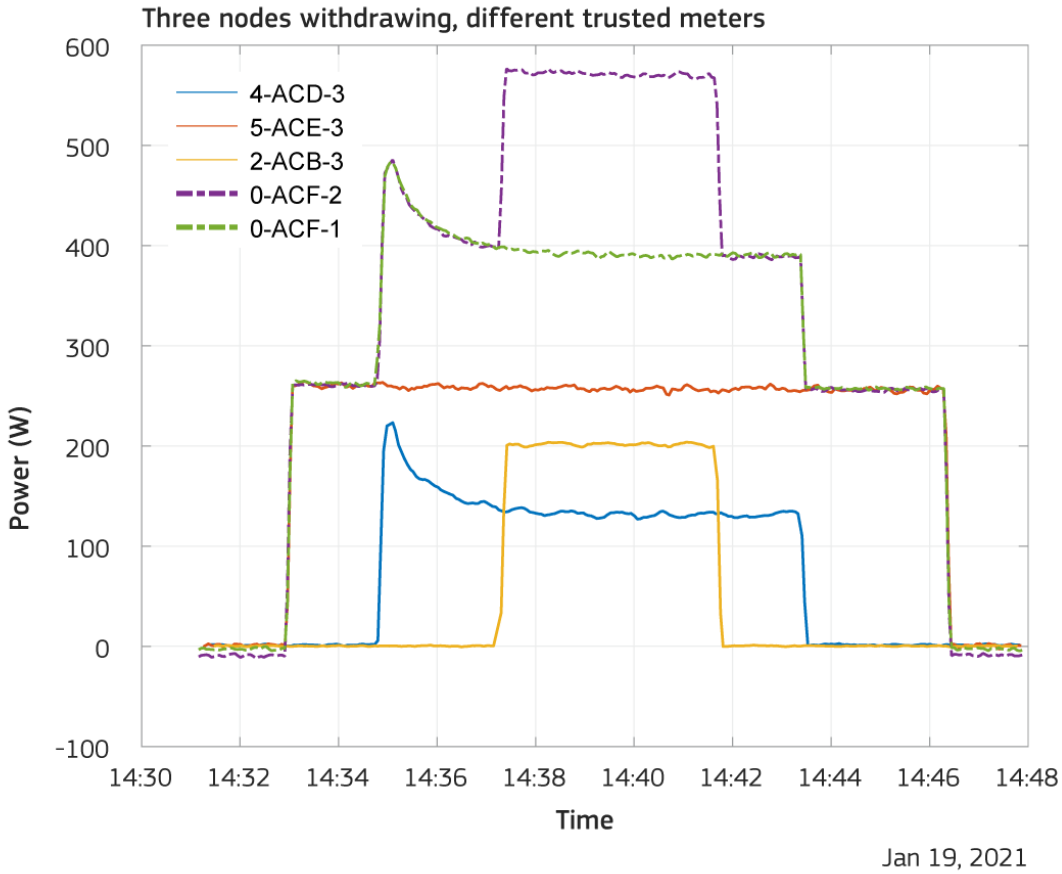


Figure 62. Test transaction 4: three nodes withdrawing, different trusted meters

Even if the scenario looks complicated, in practice the logic is exactly the same with the third test scenario; the verification is split into two parts, starting from the trusted meter 0-ACF-1 for the transactions of nodes 4 and 5, and then to the trusted meter 0-ACF-2 for the transactions of node 2.

4.5 Discussion

The implementation of the energy communities use case, demonstrates the feasibility of the whole experiment. We demonstrated in practice that it is possible to build, operate and automate an energy community, both at the hardware and at the software level. We envisaged the energy community as a trust less fully distributed system without a centralized recognized authority, implemented with the use of DLT and the use of smart contracts. This algorithm is in turn dependant on blockchain data that is essentially self-reported by autonomous and independent entities: the nodes.

The logic implementation over a blockchain system showed that even computationally low-end devices, could be a client to a blockchain system and send transactions periodically. More importantly, it showed that an integration between the “energy domain” and the “blockchain domain” is feasible in terms of technology and logic. The biggest challenge in such use cases, is how to trust the measurements and the “digital twin” of a physical object; in this case energy.

It is a huge challenge to design and operate such system even in a controlled lab environment. For example, establishing if a metering error is due to miscalibration of a sensor or malicious activity is beyond the possibility of an automated smart contract. Statistical noise estimation and modelling, advanced automated instrument calibration, profiling of sensor output with artificial intelligence, generation of adversarial networks are all techniques that might need to be applied in a real-world implementation. This is of fundamental importance in order to gain trust and acceptance from the end users. Even if it almost impossible to ensure a 100% trusted system, with the use of a common set of smart meters that independently register the measured energy, along with each household's smart meter measurements, we can assume that the level of trust achieved is reliable for such operations.

There are of course some considerations which should be expressed. We envision that in a larger scale experiment of this type, some architectural design issues may arise. The most important, is where and how the actual blockchain system will be deployed. Since the smart meters are usually not computationally potent, it would seem inappropriate for them to host the backbone of the blockchain. A potential solution would be for the households that participate to the community, to host a blockchain node each. This would also provide total transparency to the operations of the network, and in the meantime divide the operation workload. Some central services, e.g. the ordering service if Hyperledger Fabric is used, could be hosted centrally in a commonly accepted location. This is in a way how we implemented our experiment, with a centrally located blockchain implementation and community smart meters (0-ACF) that do not directly belong to a household.

From a performance point of view, the results of the flexibility use case also apply in this case. We should note however, that we observed a lower throughput to what concerns the capacity of a low-end client to send transactions to the blockchain system. However, this potential issue is due to the specific implementation we used in the energy communities use case, using a high-level script to implement such transactions. If performance was indeed the scope of the experiment, we would have used a lower level implementation, similar to the one used in the flexibility use case.

5 Implementation Problems

The implementation of the use cases goes beyond the setting up of the blockchain and smart contracts. The difficulty comes from the fact that it is applied to a certain topic. Many factors should be considered for its implementation such as permissions, number of nodes or transactions per second just to mention a few, which impacts on the easiness of the implementation. Moreover the publication on a ledger assumes data acquisition from real physical systems. These may be at a communication level, where protocols are not set up or simply inexistent. During the development of the flexibility case study, we faced that issue in which we found the smart meter which is allocated to the bus bar (where the assets are connected to) without an internet connection, which is quite normal for a smart meter. To overcome this issue, we had to retrieve the data directly from the meter inside the storage container using simpler protocols. Another issue was developing scripts and integrating data in different systems, i.e. at an information level. Different wording was used to the same variable. As an example the variable "Consumed power" was being interpreted as "energy" when transferred to the EPIC lab instead of nominal active power. Ontologies should hence be considered depending on the areas tackled. The SAREF and SAREF4ENER are reference ontologies in this area.

Another problem which the team was faced with, was related to the smart contract implementation. The architecture in the flexibility use case foresees a validation by the system operator (SO), of the data sent from the Aggregator/FSP, acknowledging that the asset power is aggregated/summed and that the ID of the assets are original. For this an "AND" policy had to be used, where the Aggregator/FSP AND the SO both validate this information. In practice this function was not enabled in Hyperledger Fabric when the study started and it was eventually facilitated as it was still under development. A list of the main difficulties during the implementation can be summed as:

- Semantic incongruence
- Communication problems with legacy systems
- Data retrieved is not cleaned and cannot be used directly
- Intermediate steps for data treatment
- Smart contract lack of ability for multiple approvals/arithmetic's triggering

The interaction with the existing systems was indeed a challenging factor. In fact it is often pointed out as one of the main issues to be solved. In particular, the challenge of integration with legacy systems, was extracting the information from where it was being generated.

This is the reason for which the energy communities use case focused on the logic and the integrations of the "energy" and the "blockchain" worlds. We had to use in-house built smart meters for this purpose, which nonetheless proved the feasibility of the experiment.

The choice of DIY low cost technology for the energy community use case was driven by a fundamental compromise between price and accuracy. The choice of an Arduino-based solution offered by a commercial producer was beneficial from an IT integration point of view but also from a strictly metrological point of view. However this approach required us to build a full protocol stack from the physical reading of the sensors to the blockchain node architecture. This approach would not be feasible for a commercial application so further standardization of protocol layers seems to be fundamental for this technology to be adopted widely.

For example, in a market-ready system the verification performed should not simply calculate the energy sum of the different batch of transactions, but should rather divide the evaluated time period into smaller time instances and evaluate each of them separately.

Another limitation that we observed was related to the hardware we used for the node controllers and thus the client to the blockchain platform. Since our client for communicating with Hyperledger Fabric was built on JavaScript, we observed that its execution was significantly slower compared to its execution on a "normal" computer. The difference may be that of a few seconds, but would still be enough in case the system and verification methods require instant measurements. Still we consider this an issues that is easy to overcome, as the Arduino smart meters are wirelessly connected and thus the node controller could be implemented in a more computationally capable device, such as a "normal" computer. Changing the client implementation, and using a lower level language could also improve the system's performance.

5.1 Real World Adoption considerations

From a proof of concept to real integration, the readiness of the technology must be assessed. If a DLT solution for a particular use case is to be ready for the market integration, it must check a few boxes before being commercially used. Consideration in terms of equipment, security, privacy and infrastructure should be well understood.

Further digitalisation is a prerequisite for smart grid operations and for making flexibility data and metering available through DLTs. The costs of this transition are expected to be high in the beginning. Smart meters, chargers and other assets need to be able to communicate to the internet, either directly, through a home energy management system (HEMS) or gateway.

The nodes and consequent validation are also a concern to be taken into consideration. In a decentralized system there must be a form of governance that should assure the implementation of upgrades/changes and also a minimum infrastructure requirement. This is important to assure a minimum level of throughput and other operational/security factors, such as security in reaching consensus, robustness, and resilience. In this sense public ledgers have enough size to offer these requisites; on the other hand a permissioned/smaller system may have several potential points of failure.

The way to acquire data is also a subject to be attended. Even though a DLT assures the integrity and untampered data once it is published, the same must be secured from the origin of data. Certificates of origin tend to be the solution, but their use might depend on the use case.

There are several solutions to implement blockchain solutions similar to Hyperledger, however the easiness of implementation is still not straightforward requiring often dedicated specialized personnel to do it. A broader dissemination is needed to facilitate typical architecture design, node implementation and connection. Furthermore, very often, the data generated by assets need to be treated, and even though a middle step requiring some programming skills shouldn't be a problem, the next step which is often a smart contract deployment is not so easy to set up. Smart contracts could then be ruled by common best practices and should be allowed to have judicial strength if needed.

Moreover some key questions need to be considered especially related to business development:

- How is a company applying security to its application and is it a priority?
- Who defines the level of privacy, decentralization, and security?
- Who has access to the ledger and how is its access controlled?
- How are updates to the software or application agreed and made?
- Should organizations think about how customers' feedback can be obtained?
- How can organizations engage with customers?

Blockchains' use is not only a technological change. Its implications will have an impact in management, accounting, law and marketing. The questions will linger around how a company can promote DAPPs or other decentralized services, where its control is decentralized and consumers are empowered while still maintaining control of marketing strategies, target audiences and managing disputes. It will present itself more and more as a solution to solve several problems but its satellite implications will still be solved in years to come, as markets decide how to work within this new environment.

Standardisation is yet another challenge facing the blockchain industry. As sectors such as automotive make their way into DLT, ensuring that blockchains offer an industry-wide benefit will require collaboration on a scale rarely seen. Finally the integration of DLT solutions with legacy systems poses a tremendous hurdle for the industry. Not because the technical hurdles cannot be overcome, but because there are relatively few use cases where it has been done.

6 Cybersecurity analysis

The growth of the emerging technologies studied in this report (namely distributed ledger technology, blockchain and smart meters) has drawn the attention of cybercrime. The cybersecurity analysis of the energy community and flexibility use cases is thus an essential step towards insurance for strong security and data protection. It is thus necessary to help in understanding, managing, verifying and mitigating the cybersecurity attacks across the two use cases.

6.1 Attacks and their systemic effects

Table 4 presents the attacks that can be performed on the use cases. The attacks scenarios are grouped into four main types:

1. **Network attacks**, which correspond to attempts to gain unauthorized access to or disturb a system via network vulnerabilities.
2. **Network traffic data attacks**, which focus on the objective of stealing or manipulating data emitted during network communications.
3. **Infrastructure attacks**, which include both the exploit of vulnerabilities of the system devices and the theft or manipulation of data stored in the system devices.
4. **Blockchain attacks**, which are similar to infrastructure attacks, but only targeting the blockchain.

Note that, in what follows, “node” refers to a prosumer node (e.g., household, a farm, or a factory), a TSO, a DSO or an aggregator.

Table 4. Attacks descriptions.

Attack type	Scenario	Description
Network attack	DoS/DDoS attack	An attacker floods a node or smart meter with requests from multiple sources, leading it to become overwhelmed to the point of slowing down substantially or even crashing.
	Port scanning attack on nodes	An attacker scans a node in order to find potential open ports to get access to the node.
	Wireless attack on smart meters	An attacker tries some well-known dedicated attacks on the Wi-Fi, Bluetooth, BLE, ZigBee connection of a smart meter to get access to it.
	Spoofing attack on smart meters	An attacker impersonates a smart meter to send fake data within the system.
Network traffic data attack	Data theft	An attacker obtains the data content of the exchanged messages, or their metadata (e.g., source or destination of the messages).
	Data manipulation	An attacker modifies the data contained in the exchanged messages.
	Data repudiation	A node denies sending data in exchanged messages.

Infrastructure attack	Bug exploit	An attacker tries to find any exploitable bug on a node or smart meter.
	Malware injection	An attacker deploys a malware on a node or smart meter in order to gain access to it.
	Digital access hack	An attacker tries to login, either as a user or as an admin, on a node or smart meter.
	Data theft	An attacker obtains the data stored on a node or smart meter.
	Data manipulation	An attacker modifies the data stored on a node or smart meter.
Blockchain attack	Digital access hack	An attacker tries to login, either as a user or as an admin, on the dedicated blockchain.
	Bug exploit on smart contract code	An attacker uses some code flaws or bugs on the smart contract to steal or modify data within the dedicated blockchain.
	Data theft	An attacker obtains the data stored on the dedicated blockchain.
	Data manipulation	An attacker modifies the data stored on the dedicated blockchain.

The actual impacts of the attacks on both the flexibility and energy community use cases are described through the definition of the **cascading attack scenarios**. They are the subsequent potential attacks that can be performed after succeeding each of the attack scenarios described in Table 4. The link between attacks and cascading attacks is depicted in Figure 63. The attacks in red are the final purposes of an attacker. Those in grey are the preliminary ones, as they prepare the attacker to achieve a final red attack. The arrows show the relation between the attacks. Note that the final red attacks do not necessary need preliminary ones in order to be achieved.

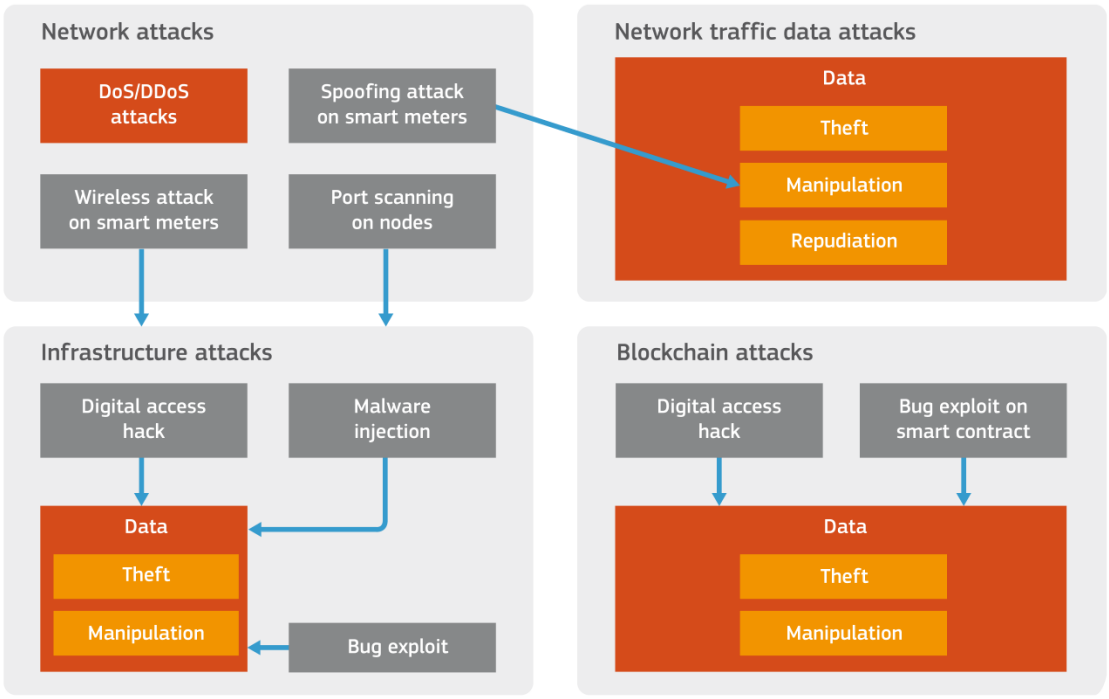


Figure 63. Cascading attack scenario(s) (Red attacks are final purposes. Grey attacks are preliminary. Arrows show the cascading attacks).

Those attacks might not have any real impact on a global scale of a given system if they are applied to a single node or smart meter. However, they might be catastrophic when they involve the blockchain or an important number of nodes and/or smart meters. For example, unauthorised blockchain transactions might lead to changes to the ledger that would be propagated to all the other entities. Changes that in principle cannot be reverted in the blockchain. This fact is called **systemic effect** in this report. Table 5 sums up the cascading attack scenarios and presents the systemic effects of both the flexibility and energy community use cases, trying to make a link between the main system components put at risk and the consequences in the use case.

Table 5. Cascading attacks and systemic effects of the attacks.

Attack type	Attack scenario	Cascading attack scenario(s)	Systemic effects	
			Energy community use case	Flexibility use case
Network attack	DoS/DDoS attack	--	Systemic effect: **HIGH** Probability of occurrence: **MEDIUM** A DoS/DDoS attack on many nodes or smart meters has a direct impact on the functioning of the energy system, making unavailable its main functionalities. In the case of the energy community, it has to be highlighted that the concept of community is to be mainly autonomous, so the disruption of activities has direct impact people and is potentially harmful.	
	Port scanning attack on nodes	Infrastructure - bug exploit - malware injection - data theft - data manipulation	Systemic effect: **HIGH** Probability of occurrence: **HIGH** A port scanning attack on many nodes can be easily performed and potentially lead to an intrusion through many entries of the system. Intrusion in nodes can open to unauthorised operations on the blockchain, such as performing ledger transactions aimed at giving some sort of advantage (e.g. economic) to the intruder.	
	Wireless attack on smart meters	Infrastructure - bug exploit - malware injection	Systemic effect: **HIGH** Probability of occurrence: **MEDIUM**	

		- data theft - data manipulation	Depending on the devices, a wireless attack on many smart meters can be easily performed and potentially lead to an intrusion through many entries of the system. Intrusion in smart meters would certainly be finalised at altering data readings in order to jeopardise the functioning of the system. Accuracy of smart meter readings is fundamental in both use cases. Their alteration in the flexibility use case would make an aggregator unable to satisfy the request and have direct consequences on the requestor. In the communities instead, it would completely jeopardise the balance from both electrical and economic points of view.	
	Spoofing attack on smart meters	Network traffic - data manipulation	Systemic effect: **HIGH** Probability of occurrence: **LOW** A spoofing attack on many smart meters has a direct impact on the functioning of the energy system. This can be used to distort the values of the entire system.	
Network traffic data attack	Data theft	--	Systemic effect: **MEDIUM** Probability of occurrence: **HIGH** The theft of data content of many exchanged messages has an impact on the privacy of the related users. This helps the attacker to know the consumption profile of the related users. It does not have a direct impact on the functioning of energy system, but it would ruin the user's perception of trustability of the system.	
	Data manipulation	--	Systemic effect: **HIGH** Probability of occurrence: **MEDIUM**	Systemic effect: **MEDIUM** Probability of occurrence: **LOW**

			The manipulation of data content of many exchanged messages has a direct impact on the functioning of the energy community system. Fake shared energy values are transmitted to the nodes, the smart meters and/or the blockchain, thus distorting the entire system.	The manipulation of data content of many exchanged messages might not have too much impact on the functioning of the flexibility system. This comes from the fact that the system is already divided in energy communities with their own aggregator that acts like a protection shield. If the data manipulation attacks are performed on the communications of the aggregators, then the result is similar to one of the energy community system.
	Data repudiation	--	Systemic effect: **HIGH** Probability of occurrence: **LOW** The repudiation of data contained in one exchanged message has a direct impact on the functioning of the energy community system. This sort of attacks is generally performed by insiders, i.e., dishonest users that would like to take advantage of the energy system. These can be users not paying for buying energy or not selling energy at a fair trade, in the case of energy community system. These can be aggregators willing to fraud the flexibility system. The trust and trustworthiness of the system might highly be in danger.	
	Bug exploit	Infrastructure	Systemic effect: **HIGH**	Systemic effect: **MEDIUM**

Infrastructure attack		- malware injection - data theft - data manipulation	Probability of occurrence: **MEDIUM** The exploitation of a bug on many nodes or smart meters has a moderate impact on the functioning of the energy community system. This can be used for launching other attacks.	Probability of occurrence: **MEDIUM** The exploitation of a bug on many nodes or smart meters might not have too much impact on the functioning of the flexibility system. This comes from the fact that the system is already divided in energy communities with their own aggregator that acts like a protection shield. If the bug exploits are performed on the aggregators, then the result is similar to one of the energy community system.
	Malware injection	Infrastructure - data theft - data manipulation	Systemic effect: **HIGH** Probability of occurrence: **MEDIUM** The injection of a malware into many nodes or smart meters has a moderate impact on the functioning of the energy community system. This can be used to steal or modify data contained in the attacked devices.	Systemic effect: **MEDIUM** Probability of occurrence: **MEDIUM** The injection of a malware into many nodes or smart meters might not have too much impact on the functioning of the flexibility system. This comes from the fact that the system is already divided in energy communities with their own aggregator that acts like a protection shield. If the malware injections are performed on the aggregators, then the result is similar to one of the energy community system.

	Digital access hack	Infrastructure - data theft - data manipulation	Systemic effect: **HIGH** Probability of occurrence: **LOW** The ability to hack into many nodes or smart meters as a user might have disastrous consequences. This might be even worse if the hack succeeds for a super-user (like an admin), who has more privileges than a standard user.	Systemic effect: **MEDIUM** Probability of occurrence: **LOW** The ability to hack into many nodes or smart meters as a user might not necessarily have disastrous consequences. This comes from the fact that the system is already divided in energy communities with their own aggregator that acts like a protection shield. If the digital access hacks are performed on the aggregators, then the result is similar to one of the energy community system.
	Data theft	--	Systemic effect: **MEDIUM** Probability of occurrence: **LOW** The theft of data contained in many nodes or smart meters has an impact on the privacy of the related users. This helps the attacker to know the consumption profile of the related users. It does not have an impact on the functioning of energy system.	
	Data manipulation	--	Systemic effect: **HIGH** Probability of occurrence: **LOW**	Systemic effect: **MEDIUM** Probability of occurrence: **LOW**

			The manipulation of data contained in many nodes or smart meters has a direct impact on the functioning of the energy community system. Fake shared energy values are transmitted to the nodes, the smart meters and/or the blockchain, thus distorting the entire system.	The manipulation of data contained in many nodes or smart meters might not have too much impact on the functioning of the flexibility system. This comes from the fact that the system is already divided in energy communities with their own aggregator that acts like a protection shield. If the data manipulation attacks are performed on the aggregators, then the result is similar to one of the energy community system.
Blockchain attack	Digital access hack	Blockchain - data theft - data manipulation	Systemic effect: **HIGH** Probability of occurrence: **MEDIUM** The ability to hack into the blockchain as a user might have disastrous consequences. This might be even worse if the hack succeeds for a super-user (like an admin), who has more privileges than a standard user.	
	Bug exploit on smart contract code	Blockchain - data theft - data manipulation	Systemic effect: **HIGH** Probability of occurrence: **MEDIUM** This type of attack is comparable to software bugs, as the smart contract is ultimately a piece of code running in the blockchain nodes. Bug exploit on smart contracts can lead to theft and alteration of data that can propagate to all the nodes. Also, it can result in behaviour and decision not compliant to initial rules set in the system (e.g. modified parameters and thresholds required by laws and regulations).	

	Data theft	--	Systemic effect: **MEDIUM** Probability of occurrence: **HIGH** The theft of data stored on the blockchain has an impact on the privacy of the related users. This helps the attacker to know the consumption profile of the related users. It does not have an immediate direct impact on the functioning of the energy system, but it can certainly decrease users trust and lead to the abandoning of the system and therefore lead to.
	Data manipulation	--	Systemic effect: **HIGH** Probability of occurrence: **LOW** The manipulation of data stored on the blockchain has a direct impact on the functioning of the energy system. Fake shared energy values are transmitted to the nodes and the smart meters, thus distorting the entire system. In particular for what concerns the flexibility use case, correct data measurements from Aggregator's assets are crucial in being able to satisfy the request and guaranteeing stability in the electricity network.

6.2 Attacks mitigation

Table 6 presents the mitigations that can be put in place to prevent the corresponding attacks or at least limit their impact. The attacks mitigations are characterised by (i) the verification processes to carry out on the two use cases, and (ii) their expected outputs. These outputs represent the measures that should be put in place so that the corresponding attacks would have a limited impact on the studied use cases.

Table 6. Attacks mitigations

Attacks		Attacks mitigation	
Type	**Scenario**	**Verification**	**Expected output**
Network attack	DoS/DDoS attack	Check the risk boundaries with a traffic generator sending out Internet packets.	The node or smart meter should be resilient up to a certain level (to be defined) of packets reception.
	Port scanning attack on nodes	Check the risk by running a port scanning tool like Nmap.	According to the specific task of the node, no unnecessary TCP or UDP port should be open.
	Wireless attack on smart meters	Check Wi-Fi and Bluetooth versions. Check the risk by running some specific tools: - Aircrack-ng, KRACK, Kr00k for Wi-Fi - BtleJack, GATTacker, BtleJuice, Crackle for Bluetooth/BLE - KillerBee for ZigBee	Wi-Fi should use WPA. Bluetooth version should be 4.1+. Aircrack-ng should not recover any Wi-Fi password. The KRACK and Kr00k Wi-Fi attacks should not work. Combining the hacking tools BtleJack, GATTacker, BtleJuice, Crackle should not break the Bluetooth/BLE communications. The KillerBee framework should not help in hacking ZigBee.
	Spoofing attack on smart meters	Check the real energy consumption measurements and compare them with the exchanged messages.	The values should be consistent.
Network traffic data attack	Data theft	Check the implementation of data-in-transit encryption measures.	The latest version of TLS and IPsec should be used. The data-in-transit should be independently encrypted with a strong cryptographic algorithm approved by NIST and ENISA (e.g., AES, Camellia, RSA or ECC) with minimum appropriate key lengths (e.g., 128 bits for symmetric keys, 3072 bits for RSA, 256 bits for ECC). In case of symmetric key encryption, a strong key exchange protocol should be used (e.g., the Elliptic Curve Diffie-Hellman protocol with Ephemeral keys).

	Data manipulation	Check the implementation of data-in-transit integrity measures.	The latest version of TLS and IPsec should be used. The data-in-transit should independently be protected with strong hash algorithm (e.g., SHA-2 or SHA-3) or digital signature (e.g., DSA, ECDSA or EdDSA) approved by NIST. In case of digital signature, the Certificate Authority should be a trusted party that delivers X.509 certificates. The PKI management should be described on a CPS (Certificate Practice Statement) as defined in the RFC 3647 related to X.509 PKI.
	Data repudiation	Check the implementation of data-in-transit non repudiation measures.	Digital signature approved by NIST (e.g., DSA, ECDSA or EdDSA) should be used. The Certificate Authority should be a trusted party that delivers X.509 certificates. The PKI management should be described on a CPS (Certificate Practice Statement) as defined in the RFC 3647 related to X.509 PKI.
Infrastructure attack	Bug exploit	Check the risk by running a fuzzing test tool.	No fuzzing test tool (e.g., the ones proposed by OWASP) should have any effect on a node or smart meter.
	Malware injection	Check OS/firmware update, firewall, anti-virus.	The OS/firmware version should be the most recent one. The firewall should run the most recent software version. The firewall should also be well-configured (e.g., block all traffic by default and only authorise specific traffic, monitor user access). The anti-virus should run the most recent software version. The anti-virus should be as much comprehensive as possible (e.g., in terms of known viruses, protection suites).
	Digital access hack	Check the implementation of authentication measures.	At least a two-factor authentication should be required (e.g., user password with a one-time-password received on a smartphone). The authentication protocol should follow the ISO/IEC 9798 standard (either a challenge-response or a zero-knowledge). In case of password use, its policy should follow the recommendation SP 800-63B

			from NIST in order to provide a high security level.
	Data theft	Check the implementation of data-at-rest encryption measures.	The data-at-rest should be encrypted with a strong cryptographic algorithm approved by NIST and ENISA (e.g., AES, Camellia, RSA or ECC) with minimum appropriate key lengths (e.g., 128 bits for symmetric keys, 3072 bits for RSA, 256 bits for ECC). The encryption keys should be stored in a well-protected environment (e.g., on a secure memory or tamper-proof device, like smartcard or HSM).
	Data manipulation	Check the implementation of data-at-rest integrity measures.	The data-at-rest should be protected with strong hash algorithm (e.g., SHA-2 or SHA-3) or digital signature (e.g., DSA, ECDSA or EdDSA) approved by NIST. In case of digital signature, the Certificate Authority should be a trusted party that delivers X.509 certificates. The PKI management should be described on a CPS (Certificate Practice Statement) as defined in the RFC 3647 related to X.509 PKI.
Blockchain attack	Digital access hack	Check the implementation of authentication measures.	At least a two-factor authentication should be required (e.g., user password with a one-time-password received on a smartphone). The authentication protocol should follow the ISO/IEC 9798 standard (either a challenge-response or a zero-knowledge). In case of password use, its policy should follow the recommendation SP 800-63B from NIST in order to provide a high security level.
	Bug exploit on smart contract code	Check code bugs by running some recent tools targeting blockchain (e.g., AnChain, ChainSecurity).	The bug tools should not find any issue.
	Data theft	Check the implementation of data-at-rest encryption measures.	The data-at-rest should be encrypted with a strong cryptographic algorithm approved by NIST and ENISA (e.g., AES, Camellia, RSA or ECC) with minimum appropriate key lengths (e.g., 128 bits for symmetric keys, 3072 bits for RSA, 256 bits for ECC). The encryption keys should be stored in a well-protected environment (e.g., on a secure memory or tamper-proof device, like smartcard or HSM).

	Data manipulation	Check the implementation of data-at-rest integrity measures.	The data-at-rest should be protected with strong hash algorithm (e.g., SHA-2 or SHA-3) or digital signature (e.g., DSA, ECDSA or EdDSA) approved by NIST. In case of digital signature, the Certificate Authority should be a trusted party that delivers X.509 certificates. The PKI management should be described on a CPS (Certificate Practice Statement) as defined in the RFC 3647 related to X.509 PKI.

7 Conclusions

In this report we demonstrate the feasibility of all the identified use-cases. Our lab setup and experiments showed that it is feasible to operate such systems from a performance, architectural and system's logic perspective.

In the flexibility and e-mobility use cases, we showed that high throughput, in terms of number of transactions per second, can be achieved, which has much higher performance than needed, especially considering that a real world flexibility DR events performs transaction in settlement period typically lasting 15 minutes. The hardware's resources, i.e. CPU and memory, showed normal usage during the experimentation, while an important factor to consider is the network topology and the network bandwidth, especially in more complicated architectures. Moreover, with our implementation it was demonstrated that financial settlements can be facilitated in a shorter period compared to the actual situation. We also foresaw a communication between the TSO and the DSO, as it is incentivized in the Clean Energy Package (CEP).

In the energy communities and smart metering experiments, we focused on the integration of the lab setup and the blockchain system. We showed that it is possible to use low-end devices to communicate and send transactions to the blockchain system. Moreover, we proposed a solution for minimizing the trust levels that need to be placed in end-users' smart meters. By using community smart meters, which are centrally located in the system and not directly controlled by a household, we developed the logic in a smart contract that performs the verification based on the readings of these meters. On top of that, the verification, being in the smart contract, is performed automatically and continuously upon energy injections and withdrawals have finished. Our implementation shows that an integration between the "blockchain domain" and the "energy domain" is feasible from both a technological and an implementation logic point of view. There are however still many issues to solve before being able to have a "ready to the market" set of solutions. Workpackage 6 will summarise the policy actions needed to unlock the potential of blockchain technologies in the energy sector.

References

[1] GO, 'The Go Programming Language', *Documentation*. https://golang.org/doc/ (accessed Feb. 01, 2021).

[2] 'Flexibility in the energy transition – A Toolbox for Electricity DSOs', Brussels, Belgium, 2018. Accessed: Feb. 19, 2021. [Online]. Available: https://www.edsoforsmartgrids.eu/flexibility-in-the-energy-transition-a-toolbox-for-electricity-dsos/.

List of abbreviations and definitions

DLT	Distributed Ledger Technologies
DR	Demand Response
DSO	Distribution System Operator
EV	Electric Vehicle
GDPR	General Data Protection Regulation
ICT	Information Communication Technology
IoT	Internet of Things
IP	Internet Protocol
IT	Information Technology
SCADA	Supervisory Control and Data Acquisition
TSO	Transmission System Operator
V2G	Vehicle to Grid
REST	REpresentational State Transfer

List of figures

List of tables

GETTING IN TOUCH WITH THE EU

In person

All over the European Union there are hundreds of Europe Direct information centres. You can find the address of the centre nearest you at: https://europa.eu/european-union/contact_en

On the phone or by email

Europe Direct is a service that answers your questions about the European Union. You can contact this service:

- by freephone: 00 800 6 7 8 9 10 11 (certain operators may charge for these calls),

- at the following standard number: +32 22999696, or

- by electronic mail via: https://europa.eu/european-union/contact_en

FINDING INFORMATION ABOUT THE EU

Online

Information about the European Union in all the official languages of the EU is available on the Europa website at: https://europa.eu/european-union/index_en

EU publications
You can download or order free and priced EU publications from EU Bookshop at: https://publications.europa.eu/en/publications. Multiple copies of free publications may be obtained by contacting Europe Direct or your local information centre (see https://europa.eu/european-union/contact_en).

KJ-NA-30780-EN-N

The European Commission's science and knowledge service

Joint Research Centre

JRC Mission

As the science and knowledge service of the European Commission, the Joint Research Centre's mission is to support EU policies with independent evidence throughout the whole policy cycle.

EU Science Hub

ec.europa.eu/jrc

@EU_ScienceHub

EU Science Hub - Joint Research Centre

EU Science, Research and Innovation

EU Science Hub

Publications Office of the European Union

doi:10.2760/55525

ISBN 978-92-76-40550-4