Blockchain: Research and Applications 2 (2021) 100010

Contents lists available at ScienceDirect

Blockchain: Research and Applications

journal homepage: www.journals.elsevier.com/blockchain-research-and-applications

Resilient design of distribution grid automation system against cyber-physical attacks using blockchain and smart contract

Abhinav Sadu *, Akshay Jindal, Gianluca Lipari, Ferdinanda Ponci, Antonello Monti

Institute for Automation of Complex Power System, RWTH Aachen University, 52062 Aachen, Germany

ARTICLE INFO

Keywords:
Blockchain
Distribution grid automation
Smart contract
Multiple attribute decision making
Resilience

ABSTRACT

The current Distribution Grid Automation (DGA) Systems are being heavily dependent on the Information and Communication Technologies (ICT) infrastructure for its proper operation. The DGA architectures are predominantly centralized and usually deployed on a dedicated hardware. This increases the risk of blackouts under a coordinated cyber-physical attack. The compromise of the dedicated hardware that hosts the central coordinator of the DGA automation results in a blackout. Though many countermeasures have already been proposed for tackling different types cyber and physical attacks on the ICT infrastructure, very few measures have been proposed to ensure the availability of the grid operation functions, even when it is compromised. This study proposes an automatic, distributed approach based on Blockchain and Smart Contract that ensures the availability of the core DGA functions even if the central coordinator that operates the grid is compromised. This is done by virtualizing and migrating/re-initialising these functions from the dedicated hardware that was compromised to another. Additionally, a Multi-Attribute Decision Making based method is incorporated into the Smart Contract that helps in selection of the optimal hardware that can host the function considering its limitations (hardware and software). Finally, a proof of concept implementation of the proposed solution is presented that utilizes the Calvin IoT (Internet of Things) platform, Flow programming tool and Hyperledger fabric and its performance is evaluated.

1. Introduction

Traditionally the distribution grid automation systems have a centralized architecture where the data from the different field devices like the measurement units and the Remote Terminal Units (RTUs) are collected centrally by the Supervisory Control And Data Acquisition (SCADA) System [1]. The collected data is then used for the different monitoring and control applications like the State Estimation (SE), Volt/Var control, network congestion management, Optimal power dispatch, fault location isolation and optimal service restoration and so on by Distribution grid Management System/Energy Management System (DMS/EMS). These applications ensure the safety and reliability of the power grids. However, both the SCADA and the DMS/EMS are generally deployed in dedicated servers in a control center. Any attack on these servers resulting in their failure would cause the loss of the operational capabilities with possible consequent blackout. For e.g, the coordinated cyber-attacks performed on the Distribution Grid Automation (DGA) system of the Ukranian DSO exploited this vulnerability [2]. Several decentralized distribution grid automation architectures were proposed in literature, in particular the IDE4L project [3,4]. In the proposed architecture, the intelligence for grid operation was allocated to individual HV (High Voltage), MV (Medium Voltage) and LV (Low Voltage) substations for managing their respective downstream grid, using the Substation Automation Unit (SAU). This reduces the computational and communication burden to the SCADA/DMS. Furthermore, each SAU is responsible for operating a specific segment of the grid and

* Corresponding author.
E-mail addresses: abhinav.sadu@siemens-energy.com (A. Sadu), glipari@eonerc.rwth-aachen.de (G. Lipari), fponci@eonerc.rwth-aachen.de (F. Ponci), amonti@eonerc.rwth-aachen.de (A. Monti).

Production and Hosting by Elsevier on behalf of KeAi

https://doi.org/10.1016/j.bcra.2021.100010
Received 2 December 2020; Received in revised form 7 February 2021; Accepted 12 March 2021

interacts with the control center and other SAUs for coordinated control of the whole power grid.

The failure of a single SAU results in the failure of a segment of the grid. The SAU is equally vulnerable as the SCADA/DMS in CA, as it is also deployed on dedicated computational hardware. Though several countermeasures have been proposed to deal with phishing attacks, credential theft, distributed denial-of-service (DDoS) attacks, killDisk attacks and unauthorized VPN/Remote Terminal access [5,6], they don't provide mechanisms that ensure availability of their functions once they have been compromised. In order to improve the availability of the grid operation functions even when the SAU is compromised, solutions were developed and presented in Ref. [7]. The authors propose a method to virtualise the grid operation functions that can be migrated from one hardware to another using Calvin IoT platform. They also investigate the effectiveness of virtualization of the functions and the latency in migration of the functions. The system is so designed that a central entity identifies the failed SAU and coordinates the migrated/re-initialized of the operation functions into an another healthy SAU in the network. Similarly, the authors [8] propose a centralized architecture using Node-Red for improving the availability of a Phasor Data Concentrator (PDC) that processes Phasor Measurement Unit (PMU) measurements collected through a wide area network. However, in all of the previous work presented the coordination of the migration of the functions was achieved by a central entity. This introduces again a single point of failure of the system. Furthermore, the coordination system is not byzantine tolerant and sensitive to status data manipulation. Additionally, the approaches do not provide a methodology to optimally place the grid operation functions considering the hardware and software capabilities of the computational resource that would host the function.

Therefore, in order to mitigate the drawbacks of the previously presented work a completely distributed coordination scheme is proposed that uses Blockchain and smart contracts to improve the resilience of the distribution grid automation system, specifically SAU in the context of IDE4L architecture. The major contribution of the manuscript is as follows:

- This study explains how the blockchain technology can be utilized in enabling the resilient functioning of the IT infrastructure supporting SAUs.
- An explanation of how blockchain based smart contracts can perform secure and distributed migration of applications is presented.
- A Proof of Concept (PoC) implementation of the proposed solution is presented. The Hyperledger Composer [19] and Hyperledger Fabric are utilized for implementing the blockchain and Smart contract component of the PoC. Then the implemented blockchain application is integrated with the Calvin platform [17] via the flow programming tool called Node-Red [20] and REST API. Then, the scenario in which application (or actor) migration would get triggered automatically by the devices (or runtimes) is proposed.
- A Multiple Attribute Decision Making (MADM) based algorithm is proposed for choosing the destination machine (or runtime) to migrate the selected applications (or actors) is proposed.
- The working functionality (i.e. migration process) of the implemented prototype is tested and the performance evaluation i.e. variation in transaction latency under certain conditions, is performed.

A more generic version of the solution has been applied for a patent and has received positive feedback from the German Patent and trademark office [21].

2. Exemplary DGA system: IDE4L – its shortcomings

A decentralized automation architecture has been proposed within the IDE4L project [3,4], in order to improve the performance of the traditional centralized ADA system. In the proposed architecture, the individual MV and LV substations manage their respective downstream grid, using the (SAU). The SAU enables the distribution of the grid operation intelligence to individual MV and LV substations. A SAU is a cyber-physical unit that has specific hardware and software components. From the hardware perspective a SAU is a computational resource that has processing capabilities, data storage capabilities and has appropriate communication interfaces required to communicate with other SAUs, IEDs (Intelligent Electronic Device), RTUs and DMS. The different software components of the SAU form the three major layers as depicted in Fig. 1. The interfacing layer is containing all the communication protocol translators that enable SAU to interact with different IEDs, RTUs and SAUs. The software components of the interfacing layer retrieve the data from the storage layer (mostly a database) and encapsulate in the specific communication protocol and send it (actuation command, control set point, etc) to the appropriate receiver (IED-Actuator and RTUs). Furthermore, the raw data (measurements, status set points, etc) received from the IEDs, RTUs and SAUs, after de-encapsulating the messages from them is stored in the instances of storage layer. Typically, the instance of a storage layer is a database. The third layer, application layer, of SAU hosts all the monitoring, control protection algorithms that are necessary for grid operation, namely, state estimation, Volt VAR control, FLISR, etc.

The decentralized automation architecture with SAUs for an exemplary MV grid is depicted in Fig. 2. The IP based secure wide area communication and standardized data modelling schemes help in realizing such a decentralized automation architecture proposed in IDE4L project. In this architecture each SAU is responsible for operating a specific segment of the grid. This reduces the computational and communication burden to the SCADA/DMS. It interacts with the IEDs configured as measurement devices (IED-Measurement), IEDs configured to control actuators and protection devices (IED-Actuator), the DMS and other SAUs (in the MV and LV grid) for coordinated control of the whole distribution grid. Depending upon the application (monitoring(control/protection) the type of data, the rate of data exchange, levels of security encryption and the communication protocol used varies. The SAU is so designed that all the software components of the SAU could be deployed in heterogenous computational hardware ranging from a single board computational devices like Raspberry PI to high performance computational servers [4].

The SAU enables the decentralization of the grid operation and thus reduces the risk of the complete blackout as each SAU is responsible for its segment. However, it should be noted that each SAU is also deployed in a dedicated hardware that is susceptible to targeted cyber-physical attacks. Loss of a SAU (hardware/Software component) means loss of grid segment operability. In the future with active distribution grids, the SAU would be the most critical component coordinating the distributed generation and load demand connected to the distribution grid. Therefore, additional resilience measures have to be deployed that ensures high availability of the SAU, specifically the algorithms that form the application layer. The proposed methodology ensures that the algorithms hosted by the SAU that are responsible for the operation of its grid segment are made available, even when the specific SAU is compromised, and thus improving the resiliency of the DGA.

3. Proposed methodology for improved resiliency of DGA

In this section the necessary pre-requisites, an overview of the proposed solution, and a detailed explanation of how blockchain and smart contract are used for implementing the solution are presented. In this study, the proposed methodology is explained in the context of increasing the resilience of the IDE4L automation architecture. Particularly, the improvement in the availability of the grid operation functions hosted by the SAUs is presented.

3.1. Necessary pre-requisites and assumptions

For successfully implementing the proposed solution, the following pre-requisites must be fulfilled.

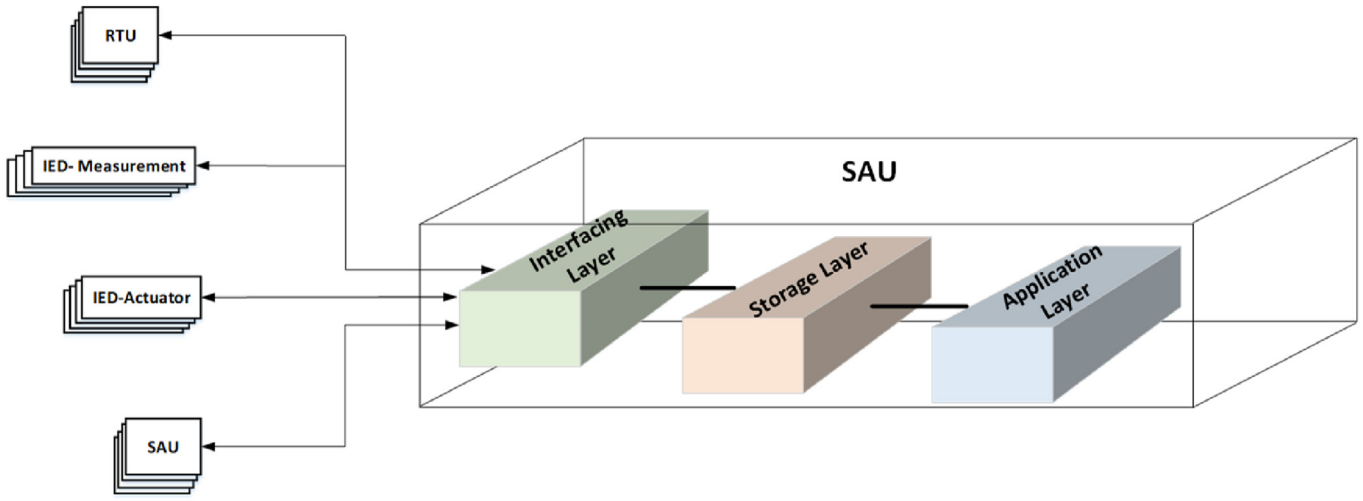


Fig. 1. Components of SAU (substation automation unit). RTU: Remote Terminal Unit.

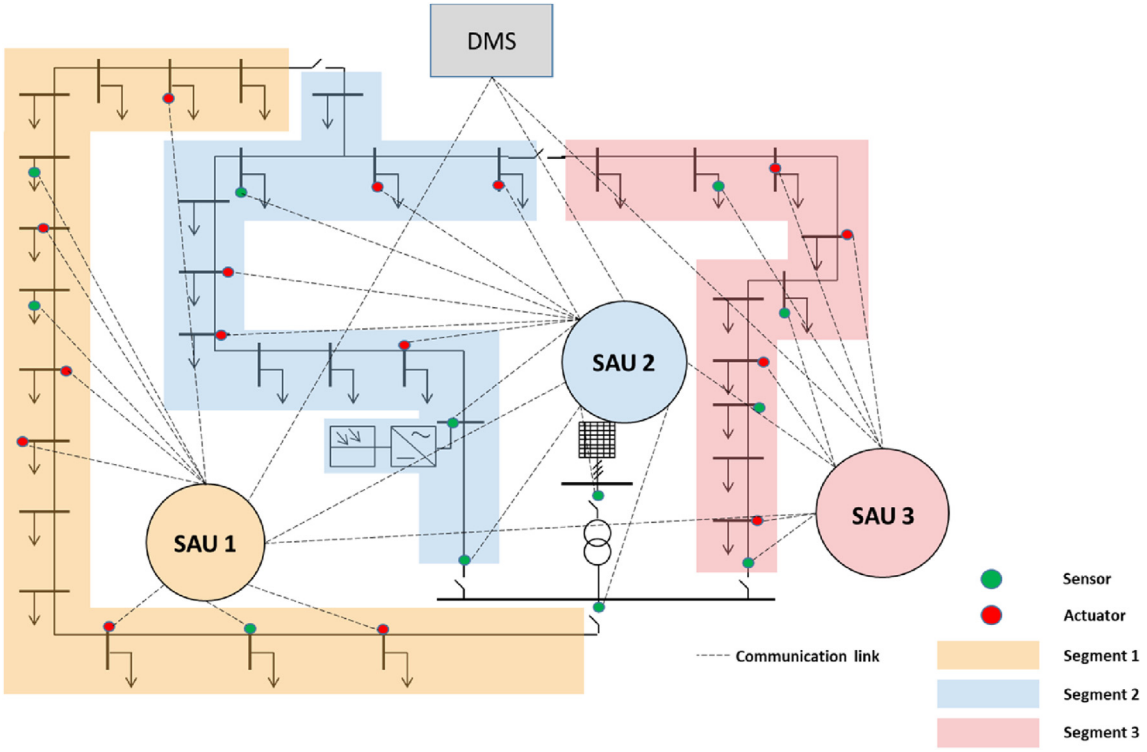


Fig. 2. IDE4L based distribution grid automation with (substation automation units): An Overview. DMS: distribution management system.

- The communication network between the SAUs and the IEDs (Measurement and Control) deployed, should be so configured that every SAU is able to receive measurements and send control commands to all IEDs irrespective of the grid segment that they are deployed in.
- The Monitoring, Control and Protection Functions should be designed as executables that are platform and OS independent. This enables seamless initialization of the functions after the migration/re-initialization in a healthy SAU. The [7] proposes a method that enables seamless virtualization of grid operation functions and initialization using CALVIN
- It is assumed that each SAU hosts services that enables identification of available communication peripheries and monitoring of available computational resources (% CPU availability, number of cores, clock rate) in real time, bandwidth utilization and available RAM in real time.
- Each SAU hosts a heartbeat service that enables the other SAUs in the network to recognise if a specific SAU is alive or is not anymore reachable within a network.

3.2. *Overview of the proposed solution*

In this study, the main objective of the proposed solution is to design a resilience measure that ensures higher availability of the grid operation functions (Monitoring, Control and Protection algorithms) hosted by SAU even when the specific SAU is compromised by a cyber-physical attack. A cyber-physical attack can be a targeted attack (e.g terrorist attacks) or caused by natural calamities. These attacks can result in a hardware failure or software failure that compromises the functioning of a SAU, thus hampering the operability of a distribution grid segment. The proposed solution provides higher availability of the grid operation

functions by migrating/re-initialising all the grid operation functions of the compromised SAU to a healthy SAU in the network of SAUs. An example to show the benefits of the proposed solution is depicted in Fig. 3. Fig. 3(a) shows the normal operation of distribution grid operated by three different SAUs. Each SAU hosts a set of monitoring functions depicted as green square, Control function depicted as yellow square and the Protection function depicted as grey square. As depicted in Fig. 3(b) Segment 3 of the distribution grid fails due to the failure of the SAU 3. However, when the proposed solution is adopted, the algorithms are migrated to the healthy SAU.

However, with the proposed solution, the monitoring, control and protection functions of the failed SAU are migrated to the other healthy SAUs. Additionally, from Fig. 3(c) it can be seen that the functions of the healthy SAUs are also redistributed among them. This migration/re-initialization is done by selecting a target SAU considering the hardware and software capabilities like the available computational and communication resources that are required for successfully hosting a specific grid operation function.

For a successful migration/re-initialization of the grid operation functions, the following components are of absolute importance.

(1) An immutable ledger of the real time status of all SAUs
   a. Hardware and software capabilities
   b. Grid operation functions are currently hosted by them.
   c. The grid segment they currently operate
(2) An immutable ledger of the requirements of each grid operation function
   a. Required computational and communicational resource
(3) An automatic triggering of an immutable logic to migrate/re-initialize the grid operation functions when an SAU fails considering the capabilities of the different SAUs (hardware and software) and the requirements of the grid operation functions.

In this study, a blockchain is used for pre-requisite (1) and (2). Whereas, a smart contract is used to implement a logic for optimally choosing the healthy SAU to which the grid operation functions hosted by the failed SAU would be migrated. A detailed explanation of the configuration of the blockchain and the Smart contract would be presented in the subsequent sub-sections. Furthermore, the necessary pre-requisites for implementing the proposed solution would also be presented.

3.3. Blockchain configuration

Blockchain was first introduced as a underlying technology of a P2P (peer to peer) electronic cash system i.e. Bitcoin [9]. Blockchain is a digital distributed ledger which is replicated and shared among all the nodes in the network [22]. It is controlled in a decentralized manner by multiple nodes running a consensus protocol [22]. It is a sequence of blocks, where each block is identified by its cryptographic hash. The first block is called genesis block, which contains an initial set of transactions and then the hash of this block is calculated by taking its transactions and a timestamp, as input. Furthermore, for every new block that is generated afterward, the hash is determined by taking previous block's hash together with its transactions and timestamp, as input [10]. As shown in Fig. 4, each block hash is pointing towards previous block's hash which results into chain of blocks [11].

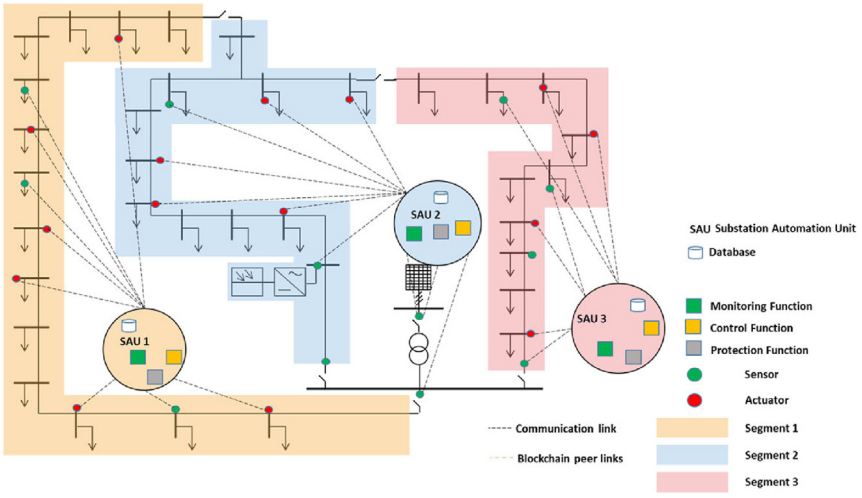
It is an append-only database which means that once the information/data is recorded on the block then it cannot be altered afterward. If an attacker would attempt to modify any transaction (or anything inside the block), the hash of the respective block would change and this would subsequently change the hash of all the blocks added after this block. Thus, blockchain is a temper-proof or immutable data storage.

Therefore, in this study blockchain is used for the following purpose.

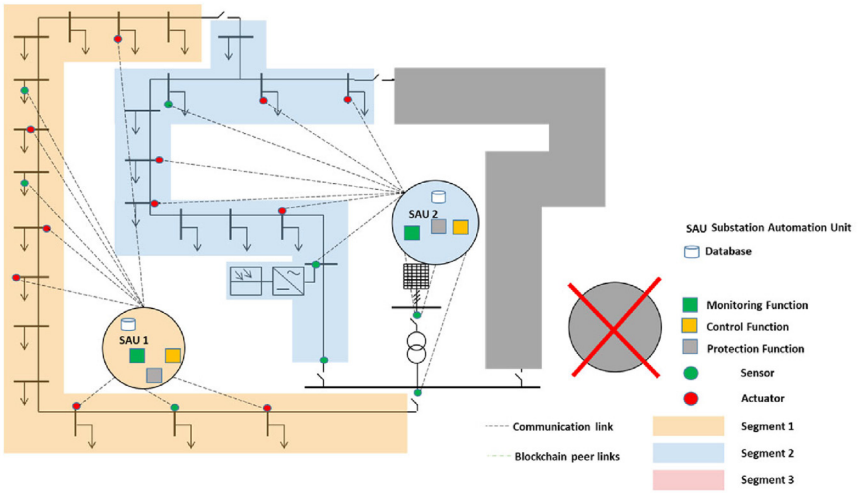
(1) Store the attributes and operational capabilities of the different SAUs in a blockchain.
   a) Hardware resources
      i) Available computational resources
      ii) Available storage resources
      iii) Communication interfaces
   b) Software resources
      i) Available communication protocol translators
      ii) Firmware compatibility (with the compromised SAU)
   c) Grid operation function specific
      i) Reachability to specific IEDs (sensors and actuators)
      ii) Average communication latency between the SAU and specific IEDs that communicated with the compromised SAU
   d) Administrative attributes
      i) Unique ID
      ii) Location
      iii) Current grid operation functions hosted
      iv) Operation jurisdiction (Grid segment it operates)
      v) Priority index for migration coordination: This index helps in determining the Master SAU that takes over the initialization of the migration/re-initialization. The SAU with the highest index updates the blockchain with the ID of the lost SAU and initiates the transaction to trigger the smart contract.
(2) Requirements of the grid operation functions to be migrated/re-initialized
   a) Required minimum computational resource
   b) Necessary communication protocols
   c) List of thresholds that have to be obeyed
      i) Maximum communication latency between the SAU and IED allowed
   d) Weights reflecting the importance of SAU resources (computational, storage and communication resources) used for optimal selection of SAU for migrating the grid operation function.
   e) List of IEDs (Sensors, actuators) that they interact with
   f) Communication parameters of the IEDs
   g) Administrative attributes
      i) Unique ID
      ii) Location

There are three different types of blockchains namely the public, consortium and private blockchain [12]. In a public blockchain any node can join a blockchain network and perform transactions. A private blockchain is restricted and only authorized nodes can participate in the network. A private blockchain is recommended when specific scalability, privacy, and other regulatory norms have to be met. A private blockchain is also preferred when all the blockchain operations are performed internally within an organization and the read permissions have to be restricted to a specific set of nodes. A consortium blockchain is a kind of private blockchain but managed by more than one organization where a selected set of nodes from different organizations have read permissions and determine the consensus.

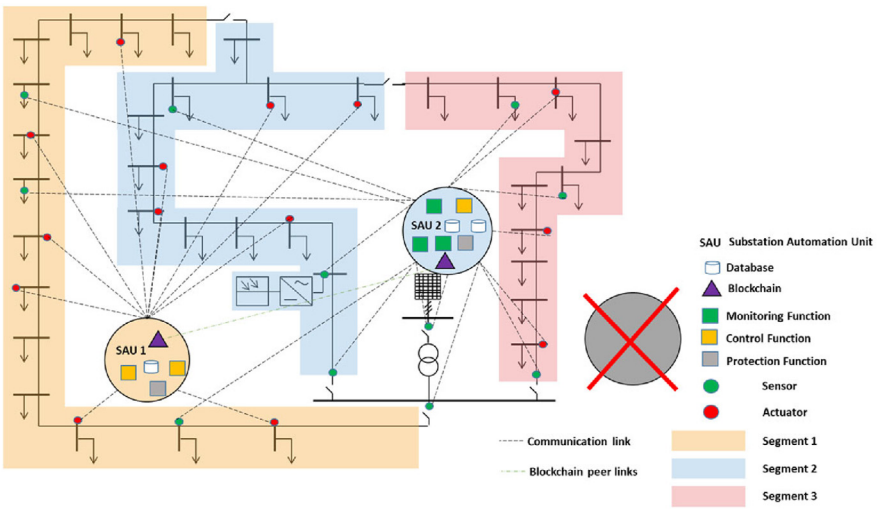
In this study a private blockchain, to be more precise a permissioned blockchain as defined in Ref. [23] is used for the migration of the grid operation functions from one SAU to another. This is because the ICT infrastructure owned to operate the grid is owned by a specific organization. Each SAU is a participant and the asset exchanged is the grid operation function. Furthermore, no unauthorized SAUs are to be allowed as participants for operating the critical distribution grid infrastructure.



(a)



(b)



(c)

Fig. 3. a) Normal operation mode with 3 SAUs (substation automation units). b) Failure of operation of Segment 3 when SAU 3 fails, c) Improved availability of SAU 3 with proposed solution.

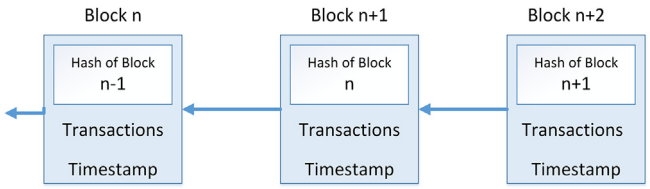


Fig. 4. Blockchain structure.

3.4. Smart contract configuration

A smart contract ensures an automatic exchange of the asset (grid operation functions) from one participant (SAU) to another upon an occurrence of a specific transaction (occurrence of failure of SAU) via blockchain. Where the assets are the grid operation functions, the exchange process is migration/re-initialization and the participants are the authorized SAUs. A smart contract enables exchange of the assets between the participants automatically according to a pre-defined logic.

Smart contracts automate the workflows or processes. They are activated upon receipt of a transaction. Once activated they perform autonomously on a blockchain node. They reside on blockchain nodes and hence are decentralized and cryptographically secured. Though, this is true for HLF based blockchain where chaincodes are installed as containers within the peers, it might not be the case for other blockchain systems where smart contacts are saved within block as lines of code for example in Ethereum. Therefore, alteration or changes in the smart contract code is impossible without being noticed.

A smart contract is triggered by a transaction. It then executes automatically in a specified way on each node of the network, based on the data inserted in the submitted transaction and smart contract's world state i.e. the data stored on the blockchain [23]. Smart contracts eliminate the need of a third-party to facilitate the exchanges between transacting parties (or devices) as all network nodes execute the contract and reach a consensus on the produced output [13]. In case, a node is malicious or altered, then it will produce disparate results and prevent the network from reaching a consensus. So, due to its non-deterministic nature, the transaction will be rejected. Additionally, all transactions are digitally signed and stored in an immutable ledger which preserves data integrity and enables historical tracking or data verifiability [14]. Hence, because of all these characteristics, blockchain based smart contracts give us an opportunity to improve the grid automation resiliency.

The different steps involved in the secure migration/re-initialization of the grid operation functions (from now on called as actors/applications) from one SAU (from now on referred as Runtime) to another for a system depicted in Fig. 5. Runtime Failure Migration is as given below.

(1) All Runtimes are equipped with the heartbeat application.
(2) All Runtimes (SAUs) periodically submit transactions to the blockchain updating their operational statuses (capabilities, available resources and applications hosted).
(3) The heartbeat actor (application), running on each runtime, helps in identifying functional SAUs in the network.
(4) A set of Calvin Master nodes with the help of the heartbeat actor send messages to the Runtimes in the same distribution grid area. The list of these Runtimes is obtained by querying into the blockchain. Failure of reception of heartbeat messages from the failed runtime makes the Master Runtimes publish a list of failed runtimes after a pre-defined timeout. A smart contract logic checks if more than 50% of the master runtimes have published the loss of the same SAU IDs. An event is generated to notify the identification of the lost SAUs. This step reduces the risk of corruption of the master runtime in identifying the lost SAUs.
(5) On the reception of this event, all the rest of the Runtimes stop their periodic updates of their statuses to blockchain.
(6) A new operational runtime needs to be selected which will be responsible for re-deploying the applications of the failed runtime.
(7) The master runtime with the highest priority index then submits a blockchain transaction.
(8) Smart contract will process the transaction from the master node and choose the new optimal runtime for the placement of an actor of the failed runtime. It will generate results that include the destination runtime ID, application ID (grid operation function to be migrated/reinitialized) and the application's state (last saved state before the failure).
(9) The master node will analyse the application's state, and then initiate the deployment of the application on the chosen destination runtime.
(10) After finishing re-deployment, the runtime with the new migrated functions will submit a blockchain transaction to update the status
(11) The master node will repeat steps from 6 to 10 until all the applications of failed runtime have been re-deployed
(12) The master node will trigger an event to re-start their periodic status update with a final migration end transaction

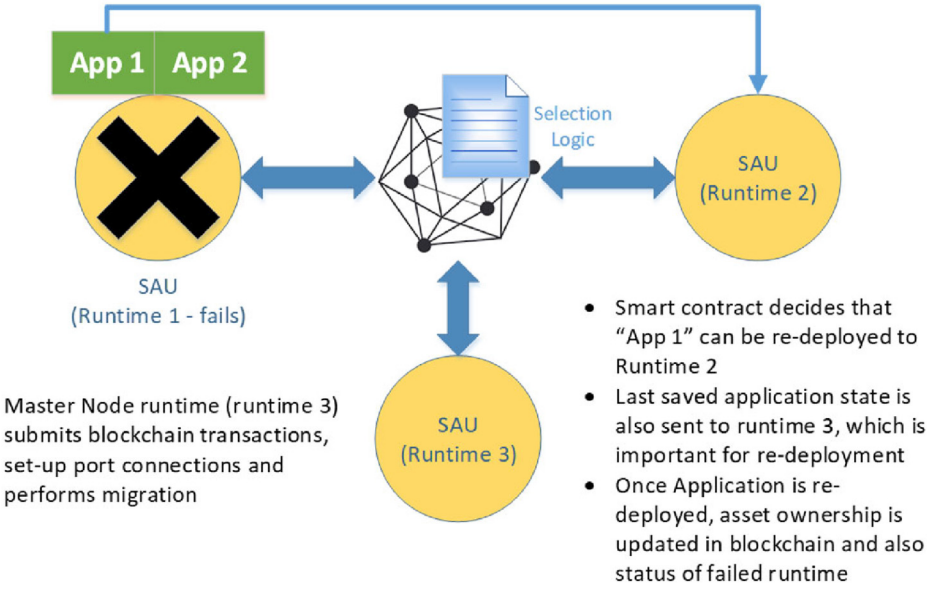


Fig. 5. Runtime failure migration. SAU: substation automation unit.

As depicted in Fig. 5, when SAU 1 (Runtime 1) fails the Runtime with the highest priority index (Runtime 3) becomes the master node and coordinates the migration of all the applications hosted in Runtime 1 to Runtime 2 using smart contract. However, it should be noted that if the Runtime with the highest priority index fails then the node with the next highest priority index becomes the master node. In this study, a MADM approach is used to design the pre-defined logic of the smart contract.

3.5. Smart contract logic: A MADM based selection of optimal destination runtime

Selecting an optimal destination runtime for actor migration is a two-step mechanism, i.e. selection of capable runtimes and application of multiple attribute decision-making to choose the most optimal runtime from the set of capable runtimes.

3.5.1. Selection of capable runtimes

It is assumed that all the runtimes (or SAUs) are on the same network, with an established connection between them. The goal of the first step is to select runtimes that satisfy the list of requirements as described below. Whenever a runtime triggers migration, the smart contract retrieves the parameters' values, mentioned in the requirements list (Section.3.5.1.1), from the blockchain ledger. Therefore, these values should be updated periodically by other blockchain transactions, which facilitates the smart contract's decision-making on the latest timestamped data.

3.5.1.1. List of requirements.

(1) Runtime status should be operational.

(2) Actor deployment requirements should match with runtime attributes.
- As shown in Fig. 6 an application's requirements could be its execution location (or address) and should match with the runtime's location (or address). When an application needs to be migrated, the smart contract logic will consider only those runtimes, for placement, which will be on the same connected network (or same static location).

(3) Runtime capabilities should contain all the functionalities needed by an actor for its execution.

(4) A runtime should have enough computational resources. This should be decided after carefully analysing the computational requirements of the different applications. The determination of such thresholds is out of scope of this study. However, the threshold values of available CPU % >20%, Available RAM >512 MB; Bandwidth utilization <80%, Round Trip Time (RTT) of less than 250 ms between the Runtime and the IED.

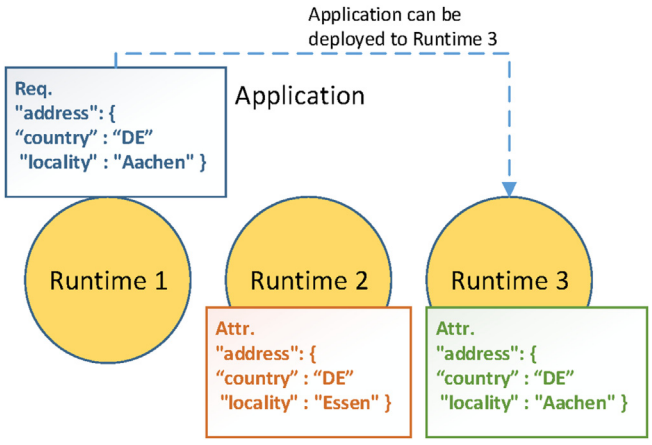


Fig. 6. Requirement matching with attributes and migration.

3.5.2. Selection of optimal runtime from the set of capable runtimes

The decision to select the most appropriate destination runtime for application migration is done by MADM methods. The MADM is extensively used in solving problems when there exists a set of feasible alternatives which need to be analysed and evaluated with respect to a set of, usually conflicting, attributes. The aim of MADM is to determine the best alternative or rank the alternatives [15]. The MADM approach adopted in this study is a two-step algorithm. In the first step depending upon the relative pair-wise weight of the criteria an absolute weight for each criterion is deduced. In the second step, these weights are used to determine the relative closeness of the available solution to the ideal solution. For the first step, the AHP (Analytical Hierarchy Process) method is used to compute the weight for each attribute by carrying out pairwise comparisons of the attributes, which is done by the decision-makers or experts. The importance of an attribute is directly proportional to its weight. It breaks down the problem into a hierarchical structure of the goal, attributes (or criteria) and alternatives as shown in Fig. 7.

The steps for calculating the weights for the attributes are explained below.

(1) Create a pairwise comparison matrix A. The matrix A is a m × m matrix where m is the number of attributes (or criteria). In matrix A, each item a_{ij} represents the importance of j^{th} attribute in relation to i^{th} attribute. The numerical value assigned to each item in matrix A is derived from Table 1 [16].

$$\Gamma = \begin{bmatrix} 1 & a_{12} & \dots & a_{1n} \\ 1/a_{21} & 1 & \dots & a_{2n} \\ . & \dots & 1 & . \\ 1/a_{n1} & 1/a_{n2} & \dots & 1 \end{bmatrix} \quad (1)$$

(2) Calculate the priority vector(PV) for each application depending on the requirements of the application can be done as shown in Eq. (2)

$$PV_j = \frac{\sum_{l=1}^{m} \bar{p}_{jl}}{m} \; where \; \bar{p}_{jk} = \frac{\Gamma(j,k)}{\sum_{l=1}^{m} \Gamma(l,k)} \; and \; m : number \; of \; criteria \quad (2)$$

In the second step the priority vector obtained from the AHP process for each application is then used to rank the set of capable Runtimes using the Technique for Order Preference by Similarity to Ideal Solution (TOPSIS) method. A generic set of equations that govern the TOPSIS based ranking of the alternatives are given below.

1. For t alternatives and m number of criteria calculate the decision matrix D as shown below in Eq. (3)

$$D = \begin{bmatrix} d_{11} & \dots & d_{1m} \\ \vdots & \dots & \vdots \\ d_{t1} & \dots & d_{tm} \end{bmatrix} \quad (3)$$

(2) Then calculate the weighted normalized decision matrix as in Eq. (4), integrating the priority vector, calculated in Eq. (2).

$$v_{ij} = PV_j r_{ij} where \; r_{ij} = \frac{d_{ij}}{\sqrt{\sum_{i=1}^{m} d_{ij}^2}} \quad (4)$$

3. Calculate the positive ideal solution (A^+) and the negative ideal solution (A^-) where B is a set of Benefit criteria and C is a set of Cost criteria.

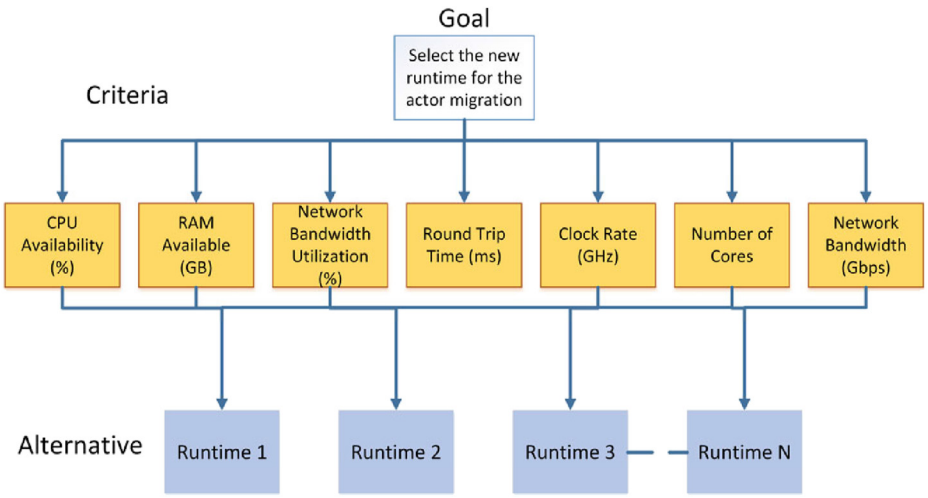


Fig. 7. Decision problem.

Table 1
Pairwise comparison scale.

Intensity of Importance (a_{ij})	1	3	5	7	9	2,4,6,8
Interpretation	Equal Importance	Moderate Importance	Strong Importance	Very Strong Importance	Extreme Importance	Intermediate values between the two adjacent judgments

$$\boldsymbol{A}^{+} = \left\{v_1^{+}, ..v_m^{+}\right\} where\ v_j^{+} = \left\{\max\left(v_{ij}\right) if\ j \in B; \min\left(v_{ij}\right) if\ j \in C\right\} \tag{5}$$

$$\boldsymbol{A}^{-} = \left\{v_1^{-}, ..v_m^{-}\right\} where\ v_j^{-} = \left\{\min\left(v_{ij}\right) if\ j \in B; \max\left(v_{ij}\right) if\ j \in C\right\} \tag{6}$$

The purpose to calculate the $\boldsymbol{A}^{+}$ and $\boldsymbol{A}^{-}$ is to measure the distance of the alternatives from the positive ideal solution and negative ideal solution. The best alternative would be the one that is as close to the positive ideal solution and as far as from the negative ideal solution.

(4) In order to rank the alternatives, the relative closeness is then calculated as in Eq. (7)

$$C_t^{+} = \frac{S_t^{+}}{S_t^{+} + S_t^{-}}\ where\ S_t^{+} = \sqrt{\sum_{i=1}^{m}\left(v_{ij} - v_j^{+}\right)^2}\ and\ S_t^{-} = \sqrt{\sum_{i=1}^{m}\left(v_{ij} - v_j^{-}\right)^2} \tag{7}$$

(5) Rank the alternatives according to their relative closeness. Higher the value of the C_t^{+} better is the alternative.

Smart contract selects the runtime (or alternative) with the highest relative closeness and then the selected actor is migrated to this runtime. Therefore, whenever there is an actor which needs to be migrated, smart contract performs logic based on the MADM approach to determine the best or the optimal runtime for application placement.

4. Implementation

In this section, an introduction of Hyperledger Fabric and Hyperledger composer is provided that is used for implementation. Furthermore, a detailed explanation of the system configuration of the distribution grid automation devices (SAUs) and their associated roles as different Blockchain entities is provided. Finally, a complete transaction flow diagram of a migration process from the detection of loss of an SAU till the complete migration of its grid operation function to a healthy SAU is presented.

4.1. Hyperledger Fabric & Hyperledger composer: configuration

4.1.1. Hyperledger fabric

Hyperledger Fabric [24] is a platform for implementing permissioned blockchain applications, written in general purpose languages such as Java, Go, or Node.js. Permissioned blockchain means that all the members that participate in the network are associated with an identity provided by the membership service provider (MSP). In the Fabric, there is no in-built cryptocurrency, and it has an append-only ledger which is replicated on all the peers and can track the history of executed transactions. Furthermore, the chaincode (or smart contract) implements the business or application logic which is installed on each peer. It allows interaction with the ledger and facilitates the exchange of assets between the transacting members. In the Fabric, a single blockchain network is a channel. There could be multiple blockchain networks possible which mean different channels, among the network participants. Each channel involves certain transacting members/participants and peers, who are authorized to access that channel. Every channel has its own shared ledger and is isolated from the other channels. A chaincode is installed on the peers and instantiated on the channel and it is possible that the same chaincode is deployed on multiple channels, but each instance is isolated. Whenever a transaction occurs within a channel, a consensus takes place by the peers on the channel. Transactions occurring in one channel are not visible to members of the other channels. Thus, the Fabric provides confidentiality (or data partition mechanism) among members or participants on the same Fabric network.

In the Hyperledger Fabric, there are three types of nodes [24]:

- Client: An application that submits the transaction proposal to the endorsing peers, and later broadcasts the endorsed transactions to the ordering service.
- Peer: A node that manages the ledger as well as the chaincode. There are two types of roles that a peer can take up:

i. Endorser: A peer which executes the chaincode for the submitted transactions, endorses (cryptographically signed) the results, and also has the properties of Committer peer.
ii. Committer: A peer which verifies the endorsements and validates the transactions.

- Ordering Service: A node that arranges the transactions into blocks and then delivers the blocks to all the peers for validation. The Fabric offers different implementations of ordering service:
  i. Solo: Centralized, mainly used for prototype development.
  ii. Kafka: Offers Crash Fault Tolerance (CFT).
  iii. BFT-SMarT [25] Offers Byzantine Fault Tolerance (BFT).

In the Hyperledger Fabric [26] the general flow of transactions starts when two or more participants join the network (or a channel). They agree upon the details of the chaincode, which is then deployed on all the peers in the channel. In addition, in the Endorsement Policy, the condition of endorsement (cryptographically signing the data) is specified, for example, a condition like A and B or C and D means either Peer A and Peer B should endorse the transaction, or Peer C and Peer D should endorse the transaction. These peers become endorsers. As depicted in Fig. 8 [24], the Client sends the transaction proposal to the peers mentioned in the endorsement policy. The transaction proposal includes the Chaincode ID, Client ID, Timestamp, and Transaction payload. Each endorsing peer executes the specified chaincode (or smart contract) and generates a read-write set based on their current blockchain state. Then each peer signs the results (contain read-write set, endorser ID, transaction ID) and returns them to the client. At this stage, peers do not perform any updates to the ledger. The Client sends the transaction (results), satisfying the endorsement condition, to the ordering service. The ordering service collects multiple transactions and groups them into blocks depending upon configuration parameters (such as Batch Timeout and Batch Size). The Ordering service just arranges the transactions but does not see the details of transactions. It delivers the blocks to all the peers for validation and commitment to the ledger. All peers (Committers as well as Endorsers) receive the blocks and for each transaction in a block, they verify the endorsements (signatures) and validate if the read set is still valid based on the current ledger state. Then they accept or reject the transaction and, in the end, the block is appended to ledger. Furthermore, the Client is also informed by peers when the transaction is accepted or rejected, and when the block gets committed to the ledger.

4.1.2. Hyperledger composer

The Hyperledger Composer [27] is an open-source framework which simplifies and accelerates the development of blockchain applications. Using the Composer, a blockchain business network can be designed rapidly, built, and deployed on top of the blockchain platform i.e. Hyperledger Fabric. A Composer business network definition consists of the following components:

- Model File: In the Composer blockchain network, there are mainly four entities or resources, such as Assets, Participants, Transactions, and Events. Participants submit the transactions in order to exchange assets between each other and also the participants can subscribe to events which can be emitted from the transaction logic. All of these entities are defined in this file using the Composer modeling language.
- Script File; The logic (or transaction processing function) for each transaction is defined in this file and it is coded in JavaScript. Basically, it's a smart contract of the blockchain application.
- Query File: Queries can be defined to fetch filtered data of resources (assets, participants and even transactions) from the ledger. Queries can be executed by transaction processing functions or the Composer REST server.
- 4. ACL: Rules defined for participant roles (present in the Composer model) which describe the permissions to perform operations (i.e. create, read, update and delete) on the network resources (assets, participants, or transactions).

A Composer business network is defined for the selected scenario as shown in Fig. 9. Runtime is a participant (representing the SAU) which is uniquely identified with uuidNode. Each runtime contains the list of actors (grid operation functions), actorList, running on it. An actor is an

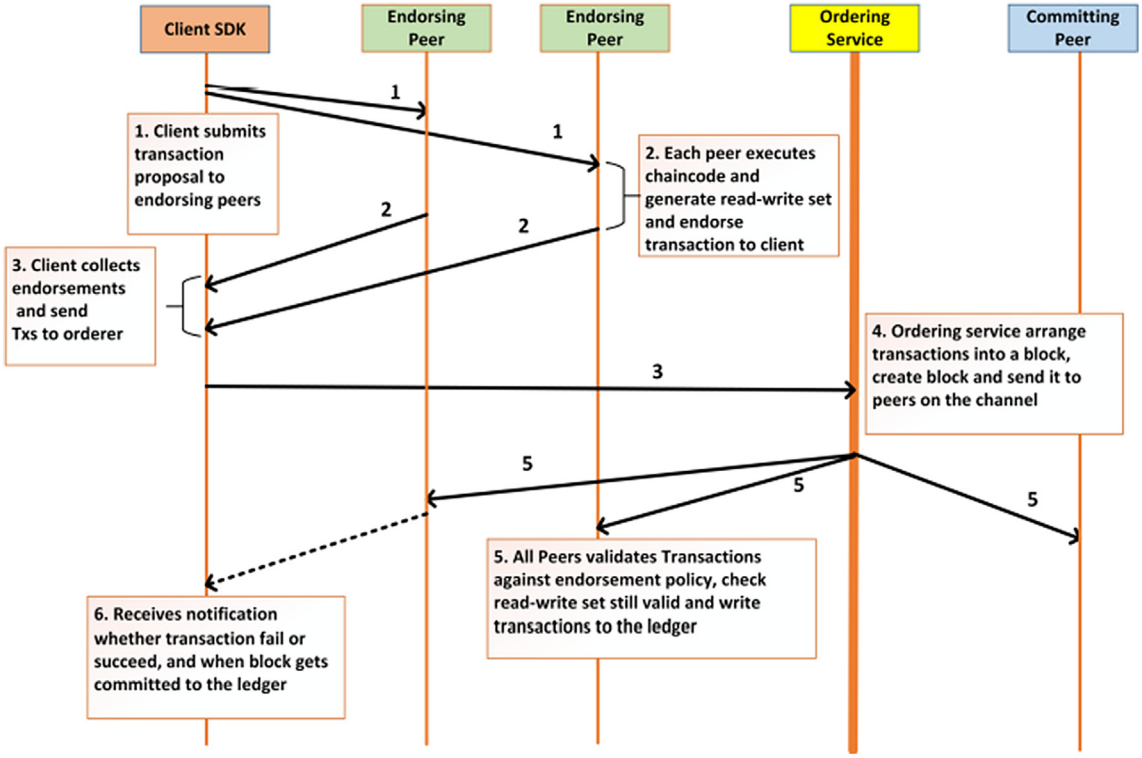


Fig. 8. Transaction flow in Fabric.

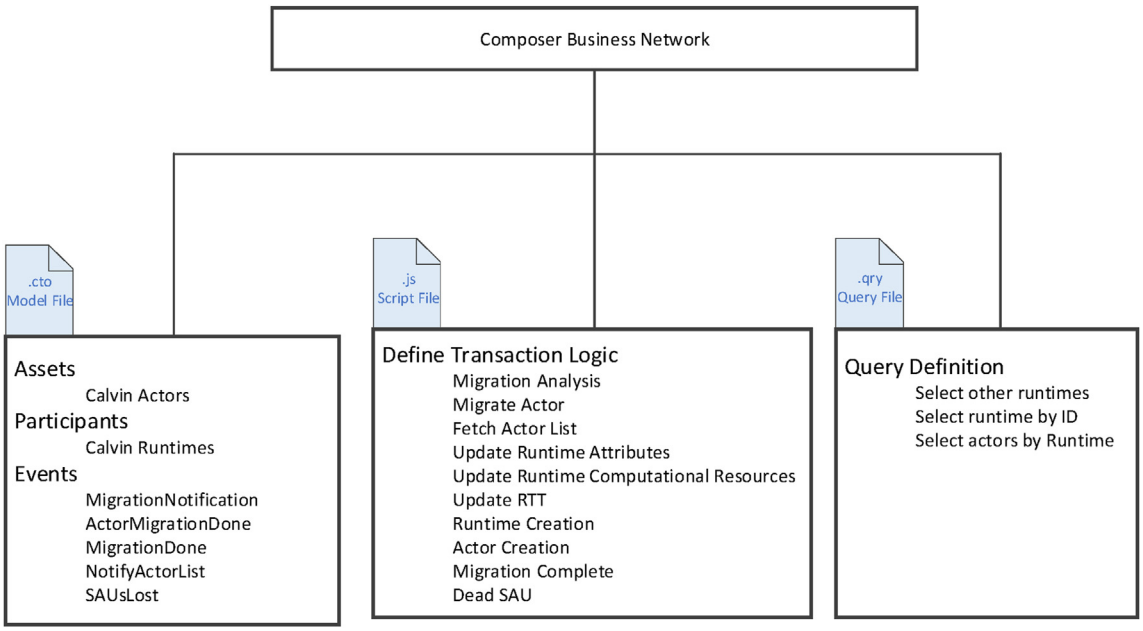


Fig. 9. Composer business network.

asset which is uniquely identified with uuidActor and each actor is linked with one runtime, represented by runtime field, which means that the actor is being executed on that particular runtime at that moment. In addition, there are many other variables in the actor and runtime structure which are used in the transaction processing logic that are mentioned in Section.3.3.

In the Composer network, events are defined in the model file and are emitted by the specified transactions in their respective transaction processing function. In this model, an external application, i.e. Node-RED, is subscribed to defined events in order to get some important information when a specified transaction is committed to the ledger and utilizes the emitted data to process another transaction. Different transactions are defined in .js script file.

4.2. System configuration

The Fabric network consists of different entities namely the peer nodes, ordering service nodes and client nodes from different organizations. To implement the proposed solution the SAUs are considered to act as both client and a peer in the Fabric network. Each SAU is installed in a Primary substation (Medium Voltage Substation) (from Now on called as PSAU) and Secondary Substation (LV substation) (From now on called as SSAU). According to the IDE4L architecture, several SSAUs report data from their secondary substation to a set of PSAUs for coordination of the automation of a section of the distribution grid. Furthermore, PSAUs interact with each other to coordinate the complete distribution grid automation. Since the PSAUs play a major role in coordination, the computational resources, memory storage resources and network connectivity of the PSAUs are generally higher than that of the SSAUs. Furthermore, the number of PSAUs in a distribution grid is lower than the number of SSAUs. This is due to the inherent radial design of distribution grids.

Taking these aspects of the PSAUs into consideration, the peer hosted by the PSAUs are configured as an endorsing peer and that of the SSAUs are configured as committing peer. To separate the critical ordering functions from the endorsers, separate ordering service nodes need to be deployed. A detailed explanation on the deployment specifics of the ordering nodes is out of scope, however, it is assumed that appropriate configuration of OSNs and the Kafka/Zookeeper clusters, as suggested in Ref. [28], is done to enable efficient management of the broadcasted transactions from the PSAUs and SSAUs.

An overview of the entities involved in the Fabric network is as given Fig. 10 The transaction flow in Hyperledger has four major phases [28].

- â€¢ Endorsement phase: The clients in SSAUs and the PSAUs generate transaction proposals signed with their credentials to all PSAUs in the same distribution grid area. The endorsing peer of the PSAU checks if the client is authorized to invoke such a transaction and signs the transaction response and replies it to the respective clients. The client checks if the transaction response bears the signature of the endorsing peer.
- â€¢ Ordering phase: After the check the client generates a well-formed transaction and broadcasts it to the ordering service. An Ordering Service Node (OSN) participates in the consensus protocol and cuts a block of transactions which is delivered to the peers by a gossip communication protocol.
- â€¢ Validation phase: All peers (endorsing and committing) check for the identity of the orderer from the blocks of transaction that were received from the ordering service and perform validation as mentioned in Ref. [28].
- â€¢ Ledger update phase: Once the validation is done the transaction is updated in the local ledger.

4.3. Software architecture of the prototype

The architecture of the implemented prototype is shown in Fig. 11. Calvin IoT [17] platform is used for developing and deploying SAUs functionalities and its applications. Calvin is a distributed IoT framework made available as an open source package by Ericsson. It combines the idea of the Actor model and flow based programming [18]. CALVIN provides a simplified framework which eases the development of the application for distributed systems. The framework consists of the three-architecture layers including runtime, actors and an application. An application in the CALVIN framework can be implemented as a combination of the actor. A detailed explanation of the implementation of the grid operation functions as Calvin actors is provided in Ref. [7]. Hyperledger Composer [19] is used for building blockchain business network whereas Node-RED [20] tool is used to integrate blockchain with Calvin platform. Node-RED is specifically used to integrate Calvin REST API with Hyperledger Composer blockchain network. Moreover,

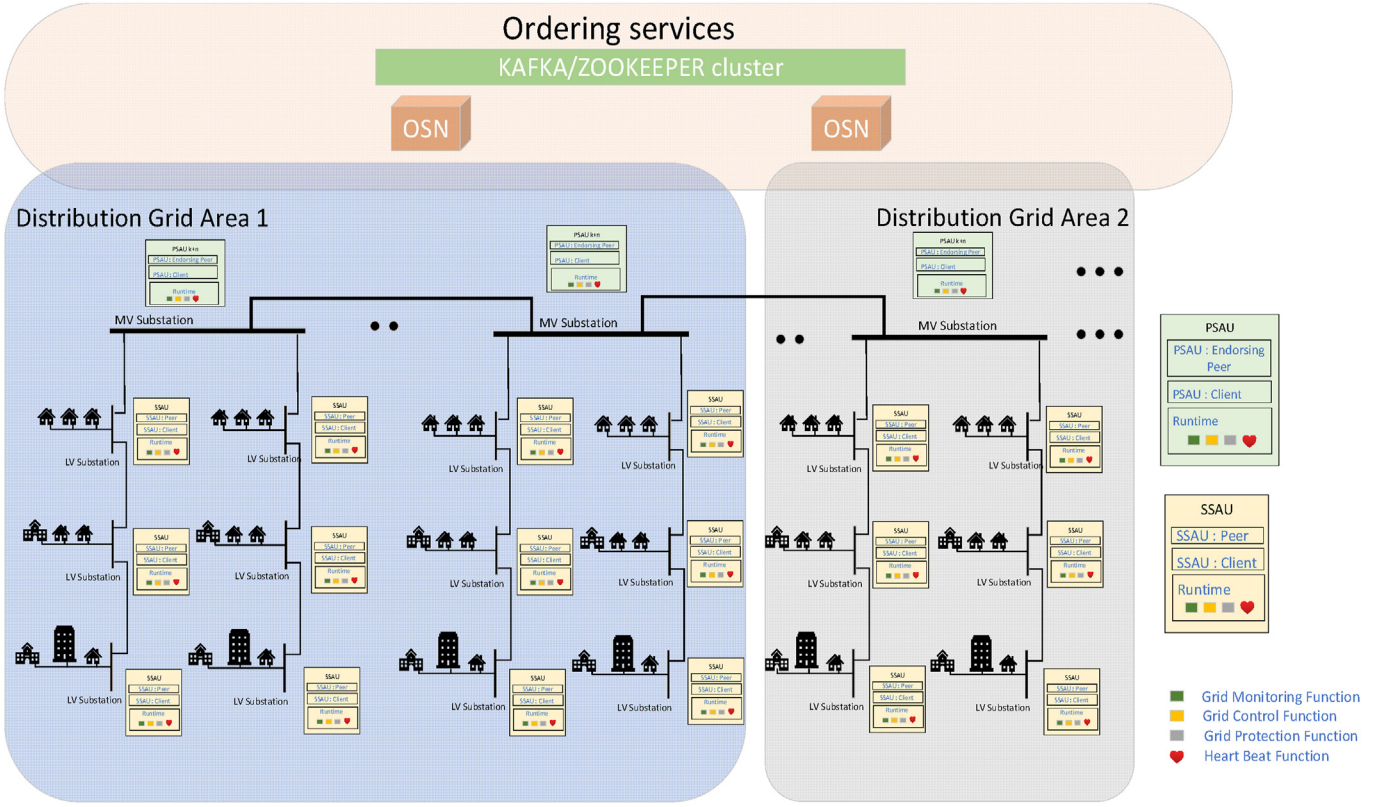


Fig. 10. Macro architecture of Fabric entities for resilient distribution grid automation design.

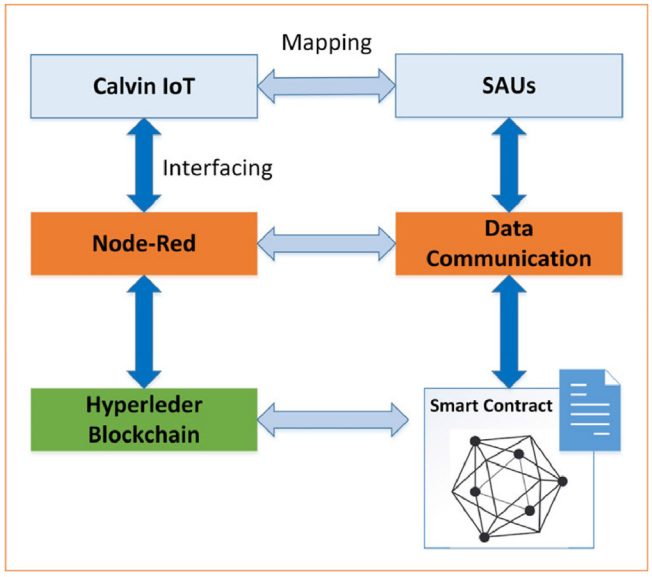


Fig. 11. System design IoT: Internet of Things; SAUs: substation automation units.

external an package i.e. node-red-contrib-composer is used which provides nodes required to interact with Composer network in order to perform activities as follows:

- â€¢ Submit blockchain transaction
- â€¢ Read, update and delete assets and participants
- â€¢ Subscribe to events

The internal mapping is shown in Fig. 11. For each Calvin runtime, a participant is created in the blockchain and for each Calvin application

Table 2
Hardware configuration and required development tool.

Operating System	Ubuntu Linux 16.04 LTS (64-bit)
Memory	8 Gb
Processor	Intel core i-3
Clock rate	2.20 GHz
Docker engine	Version 17.3
Node-Red	8.9
Hyperledger Composer	V 0.19.1
Fabric network	V1.1.0
Npm	V5.x
Python	2.7.x
Code Editor	VSCode

(grid operation functions denoted as actors), an asset is created. In Node-RED, in-built nodes are used to receive data from Calvin runtimes or vice-versa, and to submit transactions to Composer blockchain network. The specifics of the computational hardware and the associated software used for implementing the prototype are tabulated in Table 2.

4.4. Migration of the grid operation functions

The complete migration of the grid operation functions is carried out in three steps.

- â€¢ Step 1: Identification of Lost SAU
- â€¢ Step 2: Initiation of the Migration Transaction to trigger the Smart contract for optimal selection of SAU.
- â€¢ Step 3: Perform the migration of the grid operation function
- â€¢ Step 4: Updating the ownership of the grid operation functions to the new SAU by using a smart contract logic
- â€¢ Step 5: Finally with a smart contract logic a check is made to determine if either all functions of the lost SAU have been migrated or no more Functions could be migrated due to lack of

sufficient new SAUs to hos them. After a successful check, a Migration Done event is emitted to end the migration process.

The Step 1 is carried out by the CALVIN Runtimes and the blockchain clients. The Step 2 is performed by the blockchain clients and the Step 3 is performed by the CALVIN runtimes considering the output of the smart contract.

For the first step, a heartbeat-based identification of the lost SAU using the CALVIN framework is implemented. Each SAU is CALVIN Runtime (in addition to being a blockchain peer and a client) that hosts the grid operation functions and the heartbeat function as actors. A detailed explanation the virtualization of all grid operation functions with CALVIN is presented in [7]. The heartbeat function retrieves the list of SAUs in its distribution grid by querying the blockchain and sends a ping command to all of them periodically. A lost SAU is identified by another SAU when the latter does not receive an acknowledgement message back from the former. A detailed explanation of generating the ping command and processing the response from all the other SAUs using the CALVIN is presented in [7].

Generally, the Master Calvin node is responsible for orchestrating the migration of the actors from one runtime to the other using the API of CALVIN framework. Though this increases the risk of single point of failure, sufficient redundant master nodes ensure reliable orchestration of the migration of grid operation functions (Calvin Actors). This study just utilizes CALVIN for a proof of concept implementation, all the architectural drawbacks such as this single point of failure should be avoided by utilizing fail tolerant migration orchestration frameworks. However, to partially tackle this single point of failure and possible corruption of the data sent by a single master node a set of master nodes are utilized in validating the SAUs that are lost in the First step.

For the identification of the lost SAUs, all the master nodes retrieve the list of SAUs from the ledger which are deployed in the same distribution grid area and send periodic heartbeat signals to all of them. For retrieving the data from the ledger, the Calvin Runtimes configured as Master nodes post a request via the Node-Red to retrieve the information from the blockchain. The master nodes then send a heartbeat signal to all the SAUs using the Calvin API. The information retrieval from the ledger and the sending the heartbeat signal is done periodically. When an acknowledgement message from specific SAUs is not received within the pre-defined timeout time, the specific SAUs are declared dead by the master nodes and a DEAD SAU Transaction with the ID of the Lost SAUs is published. A smart contract logic is used to determine if more than 50% of the master nodes publish DEAD SAU transaction for the same SAU ID to emit the SAUs Lost Event is emitted after appending the list of the DEAD SAUs in the ledger. This process reduces the risk of single point of failure in determining the lost SAUs. However, once the Lost SAUs are determined, upon the reception of the SAUs Lost event, the master node with the highest priority index then submits the Migration analysis Transaction and initiates the Step 2. A flow diagram of this process is shown in Fig. 13.

In Step 2 the smart contract is triggered by the reception of the Migration analysis Transaction. It determines the optimal SAU for hosting the functions of one of the lost SAUs and emits the Migration Notification Event with the ID of the lost SAU and the ID of the destination SAU.

In Step 3, the Migration Notification is received by all the blockchain clients (PSAUs and SSAUs). All the SAUs stop their periodic update of the states (that are regularly requested by the blockchain client from the Calvin Runtime) to the blockchain. The blockchain client of the Calvin Master Node will raise a request the blockchain client to query the latest state of the grid operation functions from the blockchain. The blockchain client queries the ledger (using the query: Select Actors by Runtime) and provides the states of the different grid operation functions to the Calvin master node The Calvin master node then initializes the grid operation function in the new SAU. The detailed steps involved in the selection of the optimal SAU and the migration of the grid operation functions of the lost SAU is depicted in Fig. 14.

Finally, the new SAU Runtime, to which the grid operation functions were migrated, posts to its blockchain client about its new status change. The blockchain client then submits a Migrate Actor transaction(with the ID of the Lost SAU, ID of the new SAU, ID of the grid operation function and time of start of the migration) for each grid operation function (Calvin Actor) that was migrated. This triggers a smart contract logic that transfers the ownership of the migrated grid operation functions to the new SAUs. An Actor Migration Done event is emitted notifying the end of the migration of a grid operation function. After this update is done, the smart contract checks if all the grid operation functions of the lost SAUs have been migrated, if they have been then it emits the Migration Done Event, else, Step 2 to Step 4 are repeated. Finally once the Migration Done event is received by the master node, it sends the final transaction: Migration Complete to trigger the event: Commence Normal operation, that enables all the other runtimes to resume their periodical actions. The detailed sequence of processes involved in the Step 4 is depicted in Fig. 15.

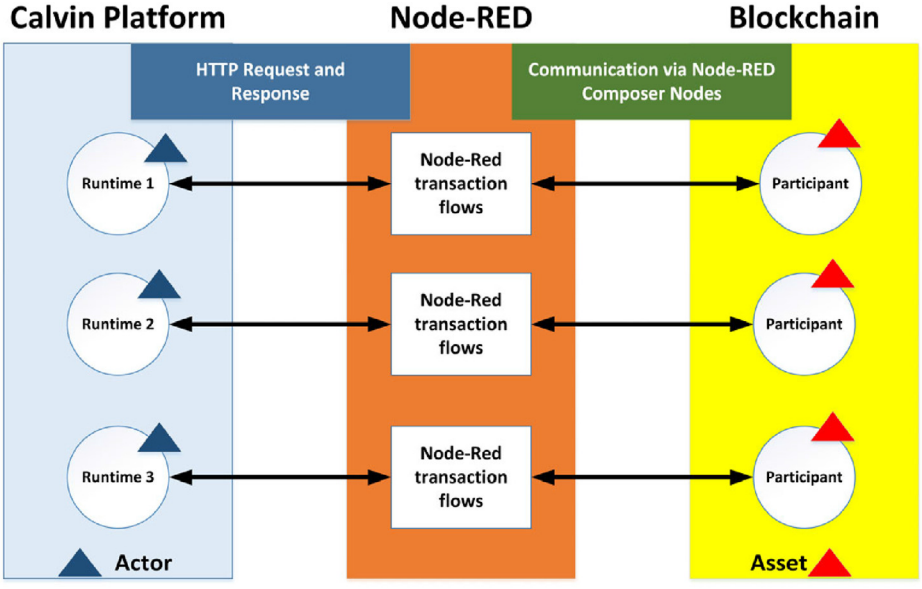


Fig. 12. Mapping between calvin and blockchain via Node-RED.

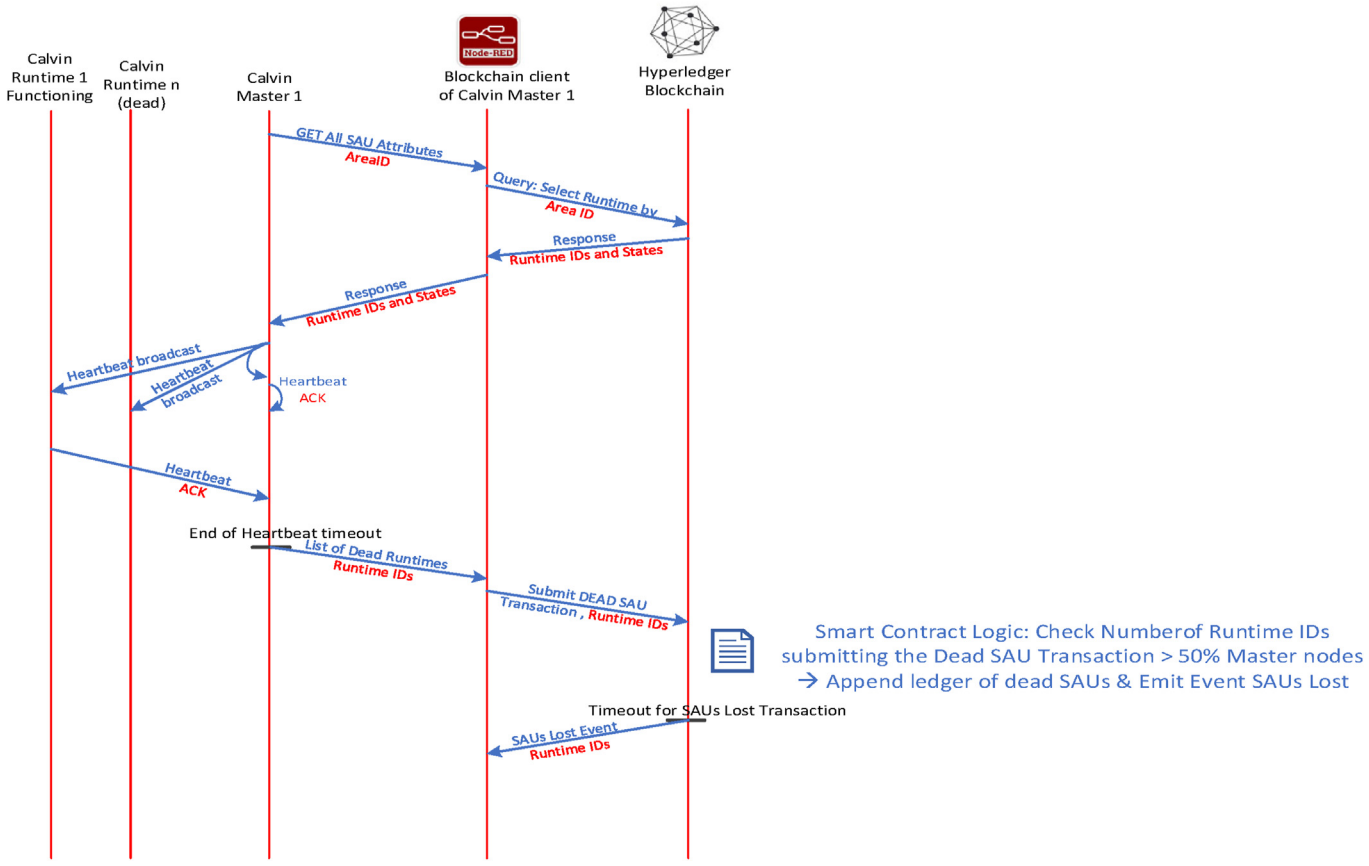


Fig. 13. Identification of lost SAUs (substation automation units) flow diagram: Step 1.

5. Test case and results

5.1. Test scenario description

For evaluating the actor migration process, a four peer blockchain network is created using the Hyperledger Fabric and then the Composer application is deployed to the blockchain network. Using Node-RED flows, four runtimes (i.e. participants) are created in the ledger and then four actors (i.e. assets) are created for each runtime. Furthermore, data is updated for all the runtimes and actors by submitting the transactions via Node-RED flows. For the actor to be migrated, the AHP pairwise comparison matrix is created by the decision-maker and then the priority vector are as shown in Table 3 and Table 4 respectively.

Along with the actor's state information, the priority vector is also added in the ledger (or asset registry) and considered to be fixed for the tested scenario. In addition, for testing purposes, the values of computation resources are assumed constant for each runtime as presented in Table 5.

A subset of the actor's state as fetched from the ledger, namely the actor ID, actor name, current runtime and its name, a test scenario is created as shown in Fig. 16 for evaluating the migration process.

Three test cases are presented. The first test case showing the optimality of the MADM based runtime selection is presented. In test case 2 the performance metrics namely the latency and throughput of the prototype implementation based on Hyperledger fabric are presented. The impact of network scaling and batch time out time is explained.

Table 4
Priority vector: Input to TOPSIS.

Attribute	Weight
%CPU Availability *Clockrate	0.1323
RAM	0.2663
Bandwidth Utilization	0.1323
RTT	0.3584
Cores	0.0554
Network Bandwidth	0.0554

Table 3
AHP (Analytical Hierarchy Process) pairwise comparison matrix.

Comparison Matrix	% CPU Availability *Clockrate	RAM	Bandwidth Utilization	RTT	Cores	Network Bandwidth
%CPU Availability *Clockrate	**1**	**1/3**	**1**	**1/3**	**3**	**3**
RAM	**3**	**1**	**3**	**1/2**	**4**	**4**
Bandwidth Utilization	**1**	**1/3**	**1**	**1/3**	**3**	**3**
RTT	**1**	**2**	**3**	**1**	**5**	**5**
Cores	**1/3**	**1/4**	**1/3**	**1/5**	**1**	**1**
Network Bandwidth	**1/3**	**1/4**	**1/3**	**1/5**	**1**	**1**

TOPSIS: technique for order preference by similarity to ideal solution; RTT: round trip time.

Table 5
Resource data for testing: TOPSIS Data Matrix.

Resources	% CPU Availability *Clockrate (GHz)	RAM (GB)	Bandwidth Utilization (%)	RTT	Cores	Network Bandwidth
Runtime 1 (SAU 1)	**72*2.2**	**13**	**70**	**13**	**4**	**1**
Runtime 2 (SAU 2)	**85*1.4**	**7**	**55**	**14**	**2**	**10**
Runtime 3 (SAU 3)	**67*2.6**	**10**	**28**	**9**	**2**	**10**
Runtime 4 (SAU 4)	**50*2.1**	**18**	**51**	**6**	**4**	**10**

TOPSIS: technique for order preference by similarity to ideal solution; RTT: round trip time; SAU: substation automation unit.

5.2. Test case 1: performance of MADM based optimal runtime selection

The Migration Analysis transaction is triggered via Node-RED flow created for a Runtime 3 which then executes the smart contract on all four peers. From the smart contract's output (i.e. emitted event tabulated in Table 6), it can be seen that Runtime 1, Runtime 2 and Runtime 4 were qualified as the target runtimes for actor migration. Smart contract applied TOPSIS technique on the data matrix as shown in Table 5, created from selected runtime's data and then chose the runtime with highest relative closeness. Therefore, from Table 6, it can be seen that Runtime 2 has the highest relative closeness and is chosen as the new destination runtime for actor placement.

5.3. Test case 2: performance evaluation

In order to evaluate the performance, two peers and four peers blockchain networks are setup using Fabric and also parameters such as batch timeout and batch size are modified to change the Fabric network configuration. The Batch Timeout is the waiting time after the arrival of the first transaction for further transactions before creating a block and the Batch Size parameter contains *max_message_count* which sets the maximum number of transactions per block and *absolute_max_bytes* which limits the size of the block. In Composer business network running over Fabric, four runtimes are created and then four actors for each runtime are created. For the latency calculations, the data is collected for each transaction as follows:

The transaction deployment time is the time (ISO 8601 format) when the transaction is triggered via runtime's Node-RED flow. The transaction completion time is the time (ISO 8601 format) when the block containing respective transaction gets committed to the ledger. Latency is defined as the time difference between transaction completion time and transaction deployment time. The average latency is defined as the mean of latency of transactions present in a data set created for a specific type of transaction. In the further sections, the variation in the average latency is explored by changing the number of blockchain peers, batch timeout and the number of transactions per block.

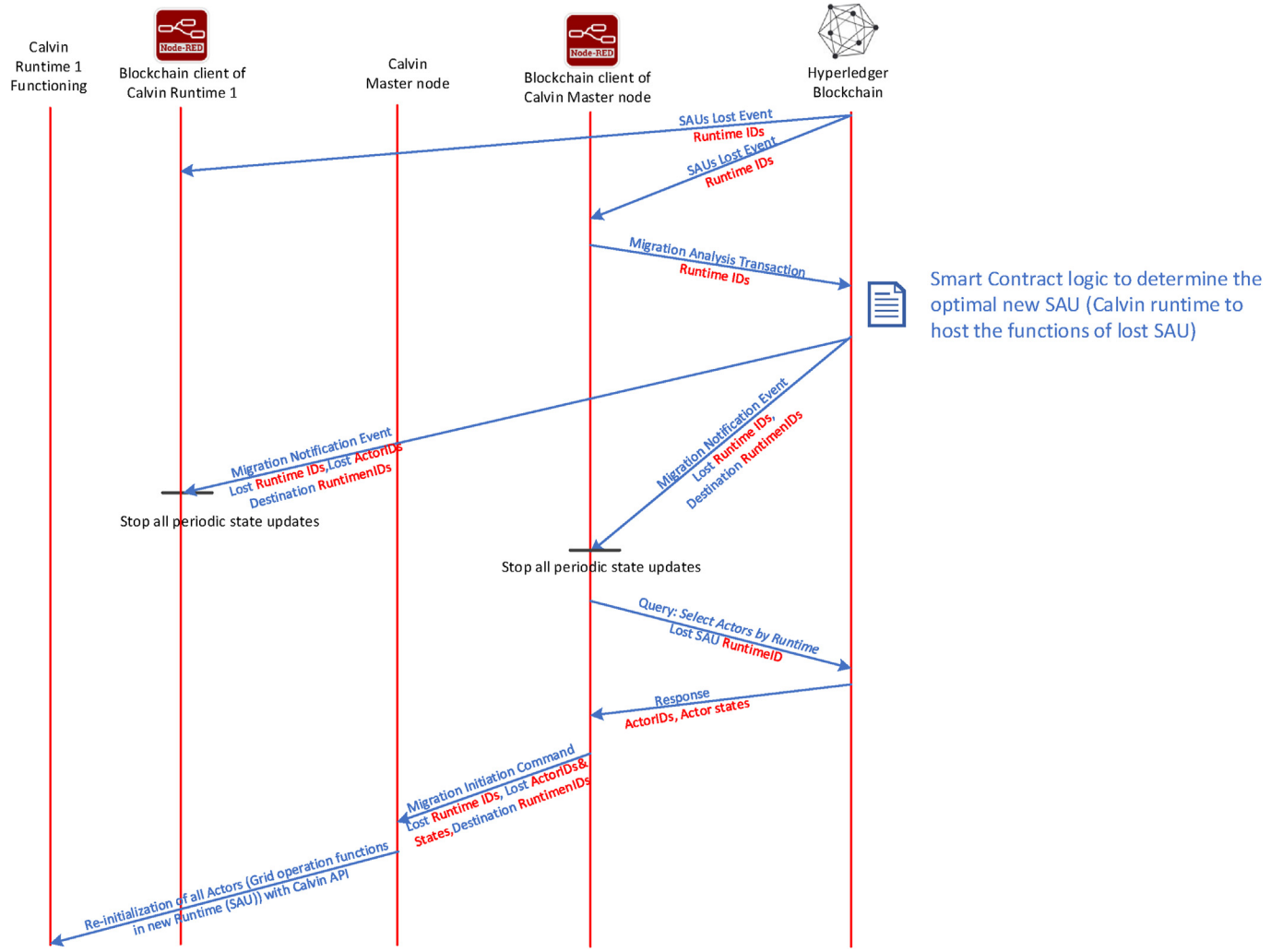


Fig. 14. Identification of optimal SAU (substation automation unit) and migration of grid operation functions: Step 2 and Step 3.

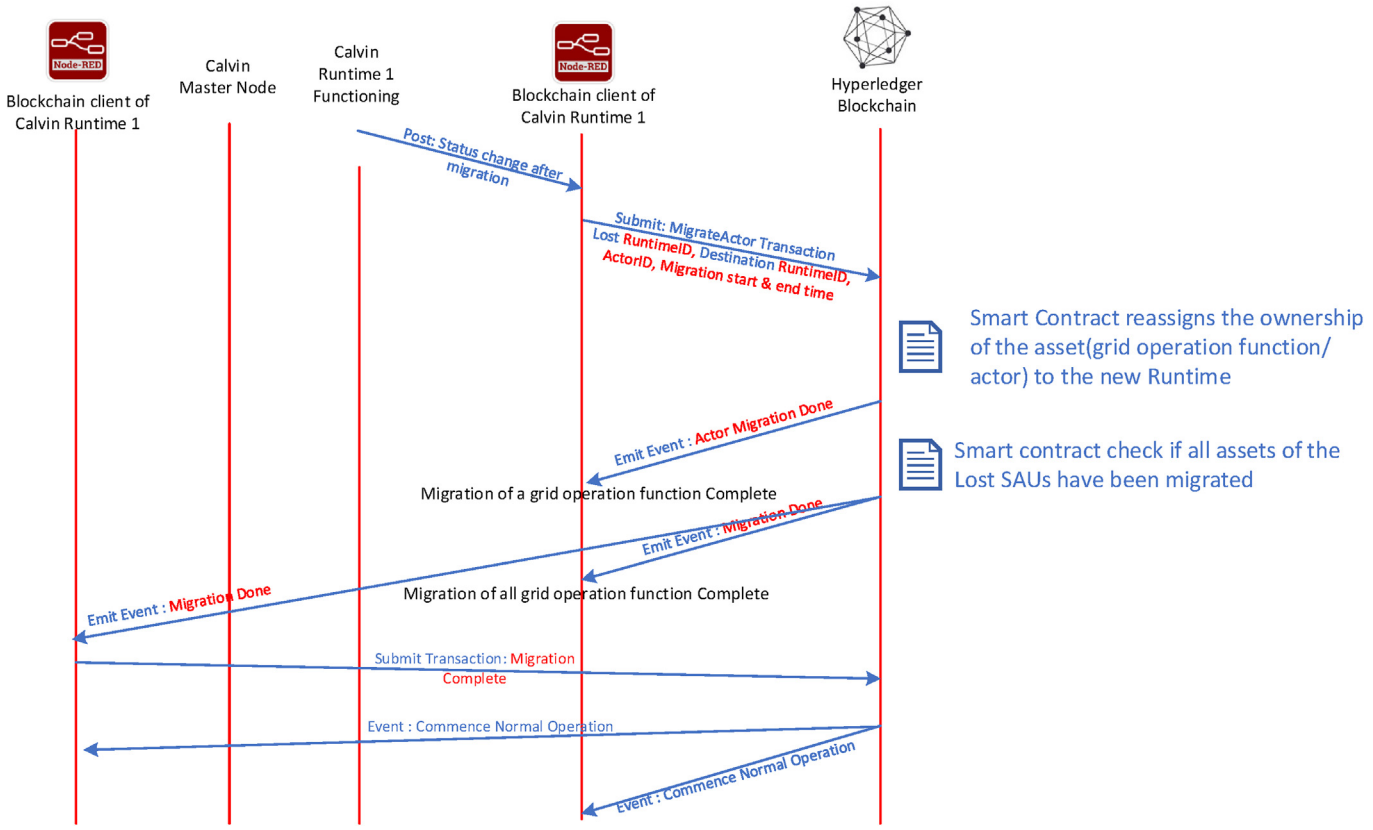


Fig. 15. Step 4 Updating the new statuses after migration.

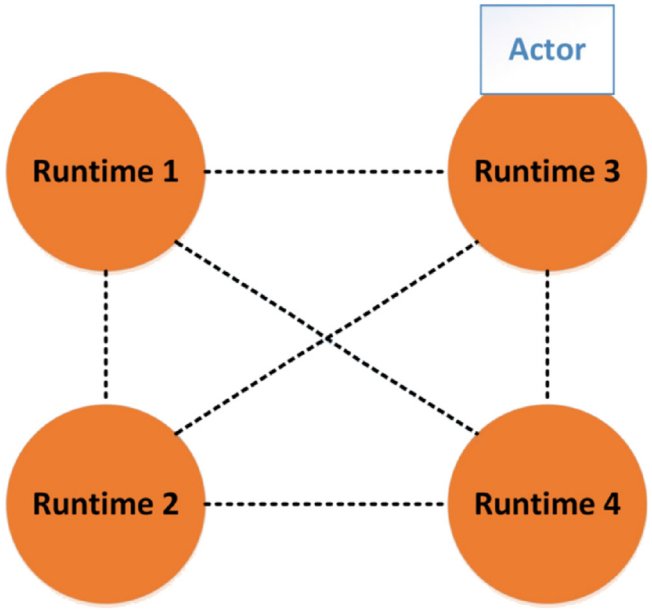


Fig. 16. Test scenario.

Table 6
Relative closeness.

Resources	Relative closeness
Runtime 1 (SAU 1)	**0.3275**
Runtime 2 (SAU 2)	**0.6400**
Runtime 4 (SAU 4)	**0.6007**

SAU: substation automation unit.

5.3.1. *Test case 2: Impact of network scaling*

This test is performed with only one runtime invoking the transactions via Node- RED, therefore, there is only one transaction per block and there are no parallel transactions from other runtimes. By configuring two peers and four peers blockchain networks with configuration i.e batch timeout (2 s), a data set containing 25 records (or blocks) for each transaction is created for both networks. Then the average latency is calculated individually for each data set. Thereafter, the comparison between the average latency calculated for both networks for each transaction is shown in Fig. 17. It is observed that the latency increases with respect to the number of blockchain peers. The reason could be that more time is required by the four peers network in reaching a consensus. Since the evaluation is performed locally, the machine resources are now shared by four peers, therefore, this could slow down the processing of transactions at each peer. All blockchain peers process the transaction and generate an output and only if the results from all the peers are identical, will the transaction be committed to the ledger. Consequently, the time to receive confirmations from all peers in a four peer network is more than a two peer network. When peers are located separately, resource allocation may not be a problem, but network delays could affect the transaction latency. Therefore, keeping in mind the above analysis, the effect of scaling the blockchain network should be taken into account while designing the blockchain platform for grid applications.

5.3.2. *Test case 2: Impact of batch time out*

A four peers Fabric network with two different configurations i.e. batch timeout set to 1 s and 2 s, are setup for this evaluation and only one runtime is invoking the transactions via Node-RED in this test. For both configured networks, a data set containing 25 records (or blocks) for each transaction is created and then average latency is calculated individually for each data set. Thereafter, the comparison between the average latency calculated for both networks for each transaction is shown in Fig. 18. It is observed that decreasing the batch timeout would improve

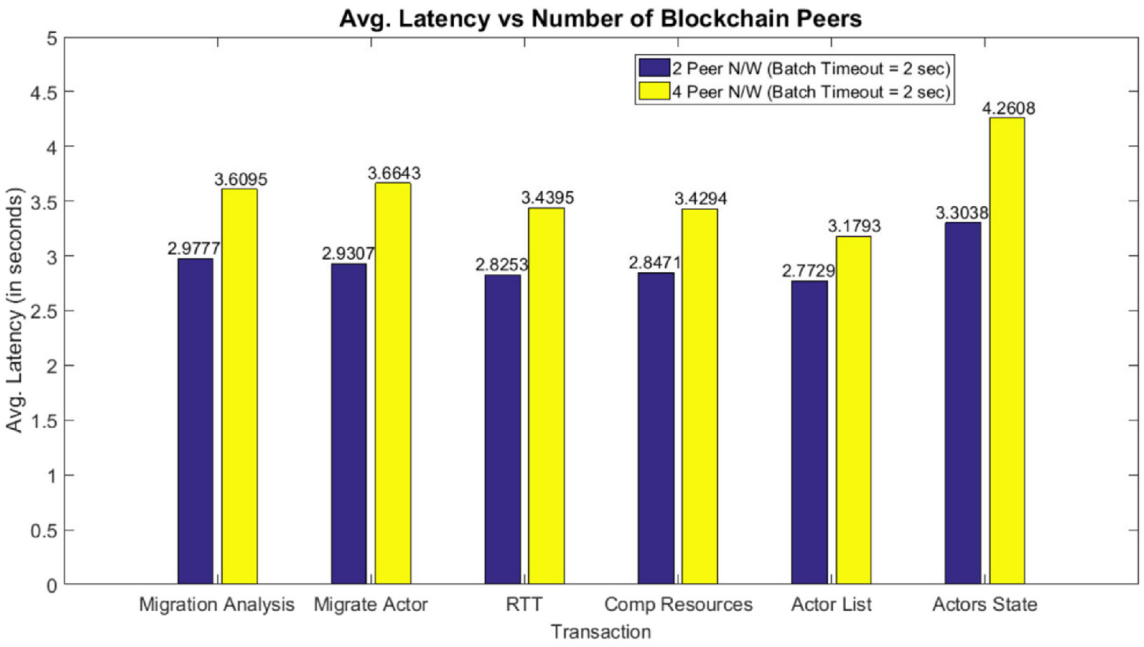


Fig. 17. Average Latency vs Number of peers.

the latency. For example, in the implemented model, the migration transactions (i.e. Migration Analysis and Migrate Actor) are always processed as a single transaction per block therefore there is no need to wait for further transactions to be added to the block. Hence, batch timeout value equal to 1 s would be good if there is a need to minimize actor migration time. However, this also reduces the throughput of the network. Thus, while developing a blockchain platform for grid applications, it needs to be taken into consideration that batch timeout value would affect the latency as well as throughput.

6. Conclusions

A novel approach for improving the resiliency of the grid automation system by utilizing a blockchain-based smart contract was introduced and a prototypical implementation was demonstrated. This technique could reduce the service downtime by migrating applications of attacked or failed nodes, to new destinations. A blockchain development framework i.e. Hyperledger Composer (along with Hyperledger Fabric) integrated with the distributed IoT environment i.e. Calvin, was utilized to realize the concept, validate the functioning of the prototype and investigate the performance. The implemented application was validated by performing application (or actor) migration for a predicted test scenario. It was observed that the transaction latency increases with an increase in the number of blockchain peers since peers took more time in reaching a consensus. Secondly, the transaction latency was evaluated with respect to batch timeout. It was observed that decreasing the batch timeout improves the transaction latency. Lastly, it was observed that the transaction latency increases with an increase in number of the

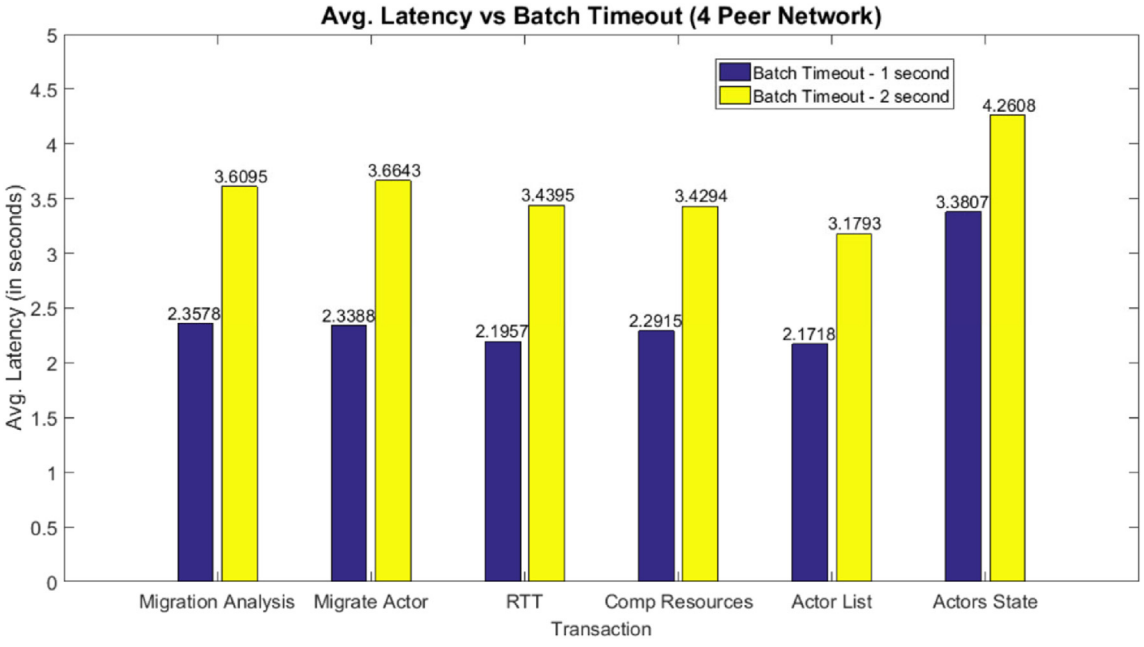


Fig. 18. Average Latency vs Batch Timeout.

transactions per block. This is due to the fact that as the number of transactions per block increases, the block committing time also increases. It can be concluded that it is possible to migrate the grid control functions to different physical systems by utilizing the smart contract solution. However, the performance of this solution would depend upon the configurational and runtime parameters. Therefore, while designing real world applications, the impact of these evaluated parameters as well as other possible parameters, such as network delays, should be taken into consideration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] F.F. Wu, K. Moslehi, A. Bose, Power system control centers: past, present, and future, Proceedings of the IEEE 93 (11) (Nov. 2005) 1890–1908.

[2] R.M. Lee, M.J. Assante, T. Conway, Analysis of the cyber attack on the Ukrainian power grid : defnse use case, 2016. Available online : https://ics.sans.org/media/E-ISAC_SANS_Ukraine_DUC_5.pdf.

[3] IDE4L, deliverable 3.2, Architecture design and implementation, 2015 [Online]. Available: http://ide4l.eu/results/.

[4] A. Angioni, A. Kulmala, D.D. Giustina, et al., Design and implementation of a substation automation unit, IEEE Trans. Power Deliv. 32 (2) (April 2017) 1133–1142.

[5] G. Loukas, D. Gan, T. Vuong, A review of cyber threats and defence approaches in emergency management, Future Internet 5 (2013) 205–236.

[6] E. Gelenbe, F.-J. Wu, Future research on cyber-physical emergency management systems, Future Internet 5 (2013) 336–354.

[7] A. Sadu, L. Ostendorf, G. Lipari, et al., Resilient design of distribution grid automation system with CALVIN, in: 2018 IEEE International Energy Conference (ENERGYCON) ; 3–7 Jun 2018, Cyprus, IEEE, Piscataway, NJ, USA, 2018, pp. 1–6, https://doi.org/10.1109/ENERGYCON.2018.8398833. Limassol.

[8] G. Di Orio, G. Brito, P. Maló, et al., A cyber-physical approach to resilience and robustness by design, Int. J. Adv. Comput. Sci. Appl. (2020), https://doi.org/10.14569/IJACSA.2020.0110710, 11. 2020.

[9] Satoshi Nakamoto, Bitcoin, 2018. Available online: https://bitcoin.org/bitcoin.pdf. (Accessed 11 May 2018).

[10] Haseeb Rabbani, What is hashing & digital signature in the blockchain?, 2018. https://blockgeeks.com/what- is- hashing- digital- signature- in the- blockchain/. Blockgeeks. (Accessed 30 May 2018).

[11] M. Alharby, A. van Moorsel, Blockchain-based smart contracts: a systematic mapping study, in: CoRR abs/1710.06372, 2017 arXiv: 1710. 06372. Available online: http://arxiv.org/abs/1710.06372.

[12] M. Tahir, M.H. Habaebi, M. Dabbagh, et al., A review on application of blockchain in 5G and beyond networks: taxonomy, field-trials, challenges and opportunities, IEEE Access 8 (2020) 115876–115904, https://doi.org/10.1109/ACCESS.2020.3003020.

[13] K. Christidis, M. Devetsikiotis, Blockchains and smart contracts for the internet of things, IEEE Access 4 (2016) 2292–2303, https://doi.org/10.1109/ACCESS.2016.2566339.

[14] M. Mylrea, S.N.G. Gourisetti, Blockchain for smart grid resilience: exchanging distributed energy at speed, scale and security, in: 2017 Resilience Week (RWS) ; 18–22 Sep 2017; Wilmington, DE, USA., Piscataway, NJ, USA, 2017, pp. 18–23.

[15] Zhongliang Yue, A method for group decision-making based on determining weights of decision makers using TOPSIS, Appl. Math. Model. 35 (2011) 1926–1936.

[16] R.W. Saaty, The analytic hierarchy process—what it is and how it is used, Math. Model. 9 (3) (1987) 161–176.

[17] Calvin base, 2018. Available online: https://github.com/EricssonResearch/calvin-base. (Accessed 21 April 2018).

[18] Per Persson, Ola Angelsmark, CALVIN– merging cloud and IoT, in: Procedia Computer Science, vol. 52, 2015, pp. 210–217.

[19] Hyperledger composer, 2018. Available online: https://hyperledger.github.io/composer/latest/introduction/introduction.html. (Accessed 21 April 2018).

[20] Node-red, 2018. Available online: https://nodered.org/. (Accessed 21 April 2018).

[21] A. Sadu, A. jindal, G. Lipari, et al., Verfahren und Vorrichtungen für eine Lastzuweisung und Überwachung für eine zuzuweisende versorgungssicherheitskritische Ressource in einem Netzwerk, German Patent (2019). Patent DE 10 2019 203 874.3, applied on 03.

[22] C. Fan, S. Ghaemi, H. Khazaei, et al., Performance evaluation of blockchain systems: a systematic survey, IEEE Access 8 (2020) 126927–126950, https://doi.org/10.1109/ACCESS.2020.3006078.

[23] A. Baggio, F. Grimaccia, Blockchain as key enabling technology for future electric energy exchange: a vision, in: IEEE Access, vol. 8, 2020, pp. 205250–205271, https://doi.org/10.1109/ACCESS.2020.3036994.

[28] P. Thakkar, S. Nathan, B. Viswanathan, Performance benchmarking and optimizing hyperledger fabric blockchain platform, in: 2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, Milwaukee, WI, USA, IEEE, Piscataway, NJ, USA, 2018 Sep 25–28, pp. 264–276, https://doi.org/10.1109/MASCOTS.2018.00034.

[24] E. Androulaki, A. barger, V. Bortnikov, et al., Hyperledger fabric: a distributed operating system for permissioned blockchains, EuroSys '18; 23-26 Apr; Porto, Portugal, ACM: New York, NY, USA, 2018, pp. 1-15.

[25] J. Sousa, A. Bessani, M. Vukolić, A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform, 2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN); 25-28 Jun 2018, Luxembourg, Luxembourg, IEEE: Piscataway, NJ, USA, 2018, pp. 51-58.

[26] M. Scherer, Performance and Scalability of Blockchain Networks and Smart Contracts, MS Thesis, Umeå University, Umeå, Sweden, 2017.

[27] Hyperledger Composer, 2018, https://hyperledger.github.io/composer/latest/introduction/introduction.html. (Accessed: 21 April 2018).