Applied Energy 281 (2021) 115963

Contents lists available at ScienceDirect

Applied Energy

journal homepage: www.elsevier.com/locate/apenergy

A framework for designing and evaluating realistic blockchain-based local energy markets

Konstantinos Christidis [a,*], Dimitrios Sikeridis [b], Yun Wang [a], Michael Devetsikiotis [b]

[a] Department of Electrical & Computer Engineering, North Carolina State University, Campus Box 7914, Raleigh, NC 27695-7914, United States
[b] Department of Electrical & Computer Engineering, The University of New Mexico, Albuquerque, NM 87131, United States

ARTICLE INFO

Keywords:
Transactive energy
Local energy market
Blockchain
Decentralized market
Peer-to-peer trading
Renewable energy

ABSTRACT

A growing customer base for solar-plus-storage at the grid edge has resulted in stronger interest at the regulatory level towards energy markets at the distribution level. Local energy markets (LEMs) running on blockchains are being studied as a possible direction, but the relevant literature treats the blockchain component as a black box. We make the case that this approach is flawed because the choices in this layer affect the market's performance significantly. We explicitly identify the design space that the blockchain layer introduces, and analyze how the design choices made therein affect the performance, governance, and degree of decentralization of these markets. As an exercise, we consider three distinct configurations for a next-generation LEM, and compare their performance on both the blockchain and the market layer via a case study. We demonstrate that simple changes in the data model can decrease the market efficiency by up to 90%. We also show that changes in the way bids get encrypted may result in economic improvements, but they do so at the risk of subverting the proper operation and resilience of the market. The simulations for our case study are conducted via a framework that we developed and open-sourced as part of this work.

1. Introduction

Solar photovoltaics (PV) and battery storage markets keep growing. For the past year through Q3, solar PV accounted for 39% of all new electricity-generating capacity installed in the US [1]. In Q3, 2.6 GW_{dc} of capacity was added – up 45% from the quarter one year-ago –, and the residential sector had its largest quarter ever with 700 MW_{dc} — up 20% year-over-year. In addition, throughout 2018, the median installation price (USD/W) across all market segments fell by 5%, continuing a 5-year trend [2] (Fig. 1a).

The story for battery storage technology is similar: in Q3, 264.6 MWh of battery storage was deployed in the US, marking a 59% quarter-over-quarter increase [3]. Moreover, the residential sector had a record-breaking quarter, with 40 MW_{dc} of newly installed capacity [4]. Across the entire US market, annual storage deployments are expected to reach 5.4 GW_{dc} in 2024, with the market reaching a size of 5.4Bn USD, growing more than 8x from 2019 (645M USD) [3]. Price-wise, costs for lithium-ion battery packs have fallen 85% from 2010 to 2019, to 176 USD/kWh [5], driven mostly by mass production of batteries for electric vehicles.

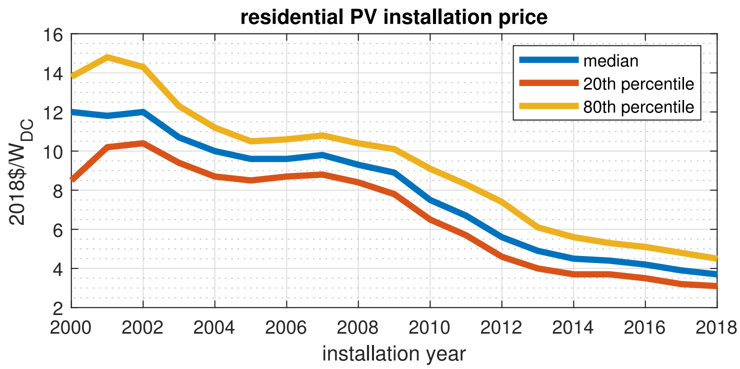
PV panels and battery storage make a powerful combination; a battery can store excess energy generated by the PV panels during daytime, and make it available to its owner during off-peak hours. As an example, a 5 kWp (kiloWatt peak) PV installation with a 4 kWh battery can double a household's consumption of PV power from 30% to 60% [6]. Fig. 1b tracks the increasing percentage of households that opt for both solar and storage across 6 US states.

Effect on utilities. A growing customer base for solar-plus-storage ultimately means less electricity consumed from the grid. Even under the least optimistic modeling scenarios [7], the levelized cost of energy (LCOE) for solar-plus-storage that can meet 100% of a site's load will be cheaper than grid electricity for millions of commercial ratepayers in New York and California by the end of the decade, with grid parity reaching residential customers several years later.

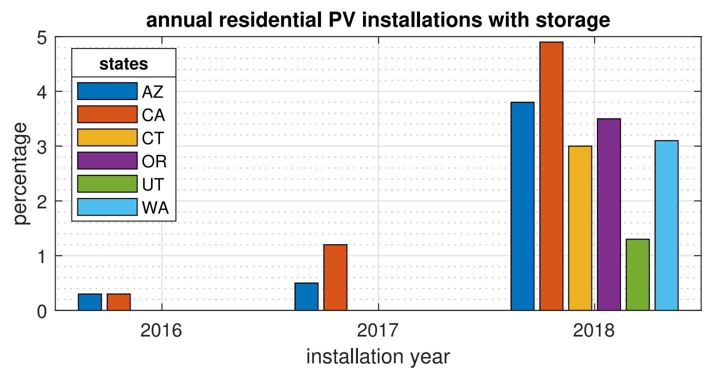
Grid parity aside, demand for power is *already* slowing down. Utilities are already seeing minimal, stagnant, or even negative load growth in their service territories [8]. The reference case (AEO2020) from the most recent Annual Energy Outlook released by the EIA (Energy Information Administration), projects an annual growth in electricity demand that averages just about 1% throughout the projection period (2019–2050) [9].

* Corresponding author.
E-mail address: kchrist@ncsu.edu (K. Christidis).

https://doi.org/10.1016/j.apenergy.2020.115963
Received 2 June 2020; Received in revised form 24 September 2020; Accepted 30 September 2020
Available online 21 October 2020
0306-2619/



(a) Residential PV installation price from 2000 to 2018 [1].



(b) Percentage of annual residential solar-plus-storage installation relative to annual residential plain solar installation [1].

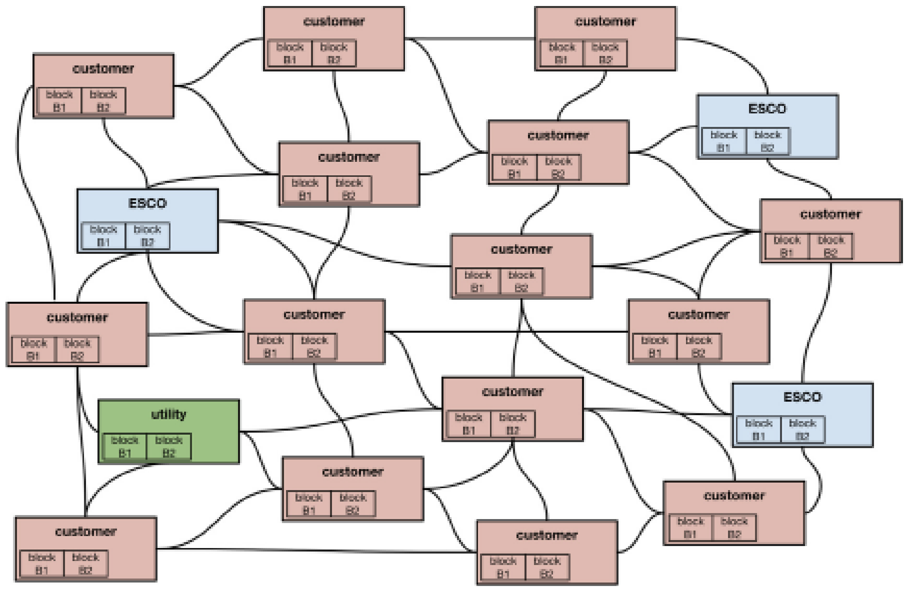


Fig. 2. Conceptual diagram for a blockchain-based local energy market that spans the area served by a distribution substation. Third-party energy service companies (ESCOs) sell services such as distributed generation, storage, or voltage regulation to consumers. [15] contains an extensive list of transaction-based energy services that can be exchanged on the distribution level, along with technical requirements to fulfill them, and a qualitative description of benefits to participating stakeholders. In this depiction, each participant in the system is represented by their own blockchain node. Black lines form the communication network that connects all nodes in the system.

other in exchange for energy. All operations run on a *single* middleware controller, on which the single smart contract is to be deployed; that node is the only one that can transfer coins from one user to another. This adds unnecessary centralization. Instead, the deployment and management of the controller node should be done collaboratively by a consortium consisting of all the nodes in the system. The authors seem to be using the blockchain exactly as one would use a centralized database.

The work in [24] presents "PETra", a blockchain-based transaction management platform for transactive energy systems (TES) aiming for participant privacy and safety by ensuring operation within stable bounds. However, on the privacy front, the authors merely propose the standard practice in blockchain platforms that has participants using a new private/public key-pair for every transaction. Additionally, the system safety concern is *not* a blockchain problem; system operators rely on the market mechanism to collateralize the sale when the supply is low, and they rely on slack buses when supply is high.

The author in [39] design a blockchain-based system that allows electric vehicles (EVs) to exchange energy with each other, and utilize a consortium blockchain. However, the usefulness of a proof-of-work consensus mechanism is unclear given that the Sybil attack vector is not there; see our work in [25] for more on this. In addition, the use of an auctioneer contradicts the claim that this trading happens "without reliance on a trusted third party" — the exact same functionality can easily be decentralized via a smart contract. Finally, there are no specifics regarding the blockchain platform's configuration; we have no idea how this affects the experiment and the results.

The work in [40] presents a blockchain-based protocol that allows electric vehicle owners to choose a tariff dynamically via Pedersen commitments [41] without being tracked over time. Charging stations post offers on the blockchain, and EVs issue a binding and hiding commitment to one of the bids by pushing a hash value to the blockchain. However, if privacy is a concern, using a different key-pair every time a tariff query is sent will suffice; this is standard practice in most blockchain setups, and is acknowledged in passing in Section 4.1 of the referenced work. In addition, the practicality of the proposed solution is questionable; there is no reasonable market assumption under which *multiple* charging station owners will *reserve exclusively* a charging slot for a customer that may ultimately not show up.

A local energy marketplace designed as part of "NOBEL", a European project, is presented and simulated in [42]. The market's design follows the stock exchange model with "sell" and "buy" order books, with the difference that the trading periods are discrete fixed-size time slots. For the evaluation scenario, the authors use "zero intelligence" agents because their behavior can help reach a high level of order matching.

In [43], the authors test out four variations of a smart contract, all implementing a uniform-price, double-auction mechanism for a decentralized energy market. Each smart contract is testing a different way to calculate the market-clearing price, covering the spectrum from off-chain to on-chain calculations. The focus is on the number of blocks generated during the simulation runs, the number of transactions packed in each block (presumably as the index for effectiveness), and on the gas cost for each transaction. The only security-related reference in the implementation section has to do with the fact that transactions are signed which is (a) pretty much a given for a blockchain network, and (b) does not constitute the basis of any security comparison. Besides the criticism above, two major differences compared to our work are that we tackle the scenario of a closed-order book (addressing the privacy concerns that the [43] would presumably cover), and we also track *how* the effectiveness of the *market* (i.e., its allocative efficiency [42]) is affected by the underlying blockchain implementation.

Another work by the same authors, that is closer to our line of work, is [44]. When it comes to experiments, the *main* difference compared to [43] is in the sensitivity analysis that is performed during the simulations; the number of market participants, the auction period, and the block generation time are not fixed. What brings this paper closer to our line of work however, is that for every experiment conducted they track the effectiveness of the market, unlike what is done in [43]. In addition, they conclude that blockchain-based systems designed for applications with high demand on synchronization need to have a block generation times x times smaller compared to a full application round. The difference to our work is that we do closed-order book auctions, and our experiments cover both the decentralization and the data contention (or concurrency) axes.

In [45] the authors run simulations of a day-ahead and real-time local energy market on a private Ethereum blockchain network with

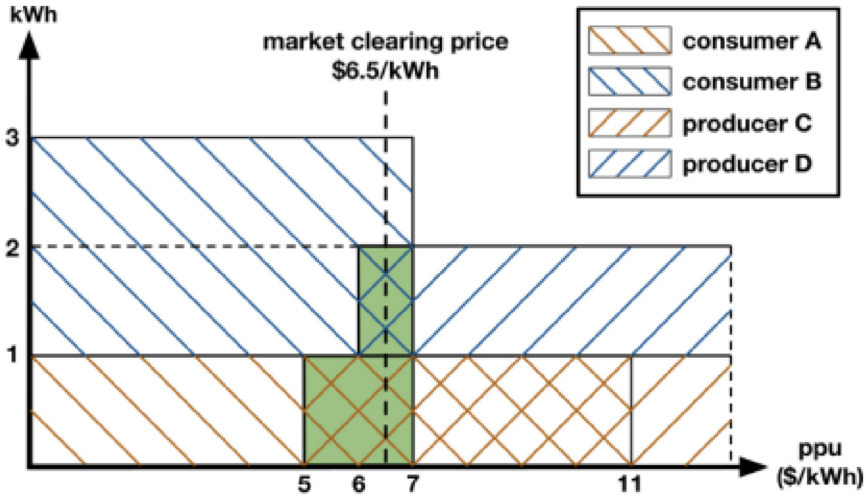


Fig. 3. An example market clearing price (MCP) calculation in a double auction. Consumers A and B have placed bids for 1 and 2 kWh with reservation prices of \$11/kWh and \$7/kWh respectively. Producers C and D have placed offers for 1 kWh each, with reservation prices of \$5/kWh and \$6/kWh respectively. We sort the demand and supply according to their reservation prices — highest (lowest) reservation price for bids (offers) goes first. The green-shaded area identifies the amount of energy that can be traded (height), and the possible prices at which it can be traded (width). Concretely here, it shows that we can trade 1 kWh at any price in the [5,6] \$/kWh interval, or 2 kWh at any price in the [6,7] \$/kWh interval. We choose as MCP the price that maximizes the amount of tradeable energy. When more than one price satisfies this condition – as is the case here – we use the average. In our case, the market clears at \$6.5 since the amount of tradeable energy is maximized in the [6,7] interval. 2 kWh are traded at this price.

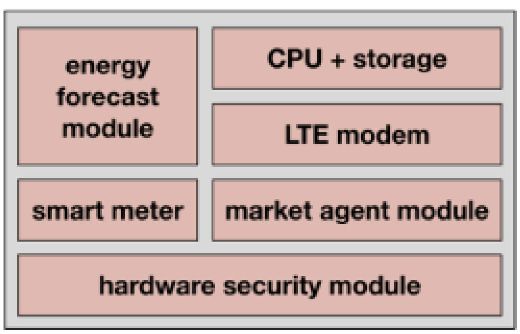


Fig. 4. Main modules in the transactive energy enabled (TE-enabled) smart meters that all prosumers use in our system model (Section 3.1).

on miners releasing cryptocurrencies into the network, and thus allowing for faster contract validation with less computation power" as one of their reasons for doing so [61].

A utility in Fukushima, Japan debuted a blockchain-enabled microgrid last year [62] to experiment with energy exchanges between the 1000 participating households. The participants were also given smart power strips that can be disabled based on signals from the backing blockchain — this is, for instance, the case when the supply cannot keep up with demand.

Vienna-based startup Grid Singularity has conducted trials where customers pay-as-they-go for the energy that they need [63]. Their target is developing countries [64]; pre-paid metering in South Africa for instance is important since 80% of its population is unbanked.

Liton is a licensed energy supplier in Germany with a blockchain-based marketplace that connects consumers (700 households) directly with energy producers. They use Ethereum because they want a permisionless blockchain, but the long transaction times are a paint point ("20 to 30 s to tell a customer whether they can buy energy or not" [65]).

Finally, we point the reader to the survey work done in [66]; the authors have identified more than 140 blockchain projects and research initiatives in the energy space, breaking them down by field of activity and blockchain platform used.

3. System model

3.1. The energy market

3.1.1. Market participants

The local energy market spans the service area of a distribution substation. It consists of N households. All of them are equipped with solar panels, and can therefore produce their own energy (prosumers). The prosumers will sell that excess energy to a neighbor, if there is demand; otherwise they will sell it back to the grid for a lower price. For simplicity, and without lack of generality, we do not consider a residential energy storage mechanism.

3.1.2. Market mechanism

Energy is exchanged using a double auction market, implemented via a closed order book, with discrete market closing times, and price–time precedence, as is the case in [24,36]. A uniform market-clearing price (MCP) is determined for each time slot[7]; the lowest bid price that can still be served given the aggregated supply determines the market clearing price [36]. Fig. 3 shows an example of how the MCP is calculated.

3.1.3. Smart meters

Bids are placed and received by the households via their TE-enabled smart meters, inline with the setup adopted in [24]; each smart meter is equipped with substantial on-board computational resources, and can also communicate with other smart meters. Besides their standard functionality – ability to measure line voltages, power consumption and production, and communicate these to the distribution system operator –, the smart meters carry an energy generation/consumption forecast module, and a market agent module (see Fig. 4).

For simplicity, we assume that the forecast module has perfect accuracy for the next slot, as in [37]. Whenever the agent generates a bid, the encompassing meter posts it on the blockchain that underpins the market — it therefore doubles as a blockchain node.

Via the hardware security module (HSM) installed on the device, the smart meter periodically posts to the blockchain network the *actual* energy consumed from, and delivered to the grid.

[7] We refer the reader to [35], for an open-source library we developed for MCP calculation.

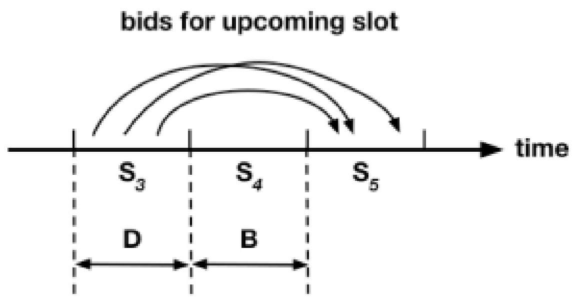


Fig. 5. In the example shown here, bidding for slot s_5 takes place during slot s_3. In general, bidding for slot s_i takes place during slot s_{i-2} (Section 3.1.6).

3.1.4. Bidding

On a given slot, each market participant can a place of maximum of one bid ("buy" order) and one offer ("sell" order). Each order consists of the quantity of energy being bought or sold, the slot it applies to, and a reservation price.[8]

The bidders are rational: the reservation prices they pick for their bids are greater than the low price that the grid is offering to them for their surplus (the feed-in tariff [67]), and smaller than the high price that the grid is selling energy to them for (the retail rate). The MCP then is bound between the feed-in tariff [67], and the retail rate offered by the grid.

Prosumers sell *all* of their output on a given slot, instead of using it to satisfy their own needs first. This assumption does not necessarily work in favor of the individual generator, as they may end up bying energy for a higher price than the one they sell their own production for, but it contributes positively to the welfare of the local market. Our motivation for this choice is purely practical: we want to increase the opportunities for matching bids by increasing the tradeable energy, since one of our goals is to evaluate the trade clearing performance of the platform.

All unmet energy needs are balanced by the grid; excess production is returned to the grid using the feed-in tariff [67], and unmet demand is satisfied at the retail rate. This is also the case when the market cannot clear for a slot due to issues with the backing transaction management platform.

3.1.5. Billing

Billing is handled by an ESCO on the network. The ESCO audits the blockchain, looking at the amount of energy that a prosumer *committed* to produce/consume on a given slot, along with the market clearing price for that slot, and the *actual* energy the producer produced/consumed on that slot, as communicated via their HSM-carrying smart meter (Section 3.1.3). It then reconciles those two streams of data in order to credit/debit each prosumer accordingly. The actual exchange of money between the prosumers and the ESCO takes place offline, i.e. it is identical to how things work today between electricity consumers and their utility providers.

3.1.6. Time-frame of trading

Bidding for slot s_i lasts for a fixed duration D and closes a certain amount of time B before that slot is about to become active. For simplicity, we adopt $D = B$ as in [68], i.e., bidding for slot s_i takes place during slot s_{i-2}, and the market clearing price is calculated and communicated to the market participants at the end of s_{i-2}, allowing them to adjust their deficit or surplus for the upcoming slot with the grid, as seen in Fig. 5.

[8] The maximum (minimum) price at which the buyer (seller) is willing to trade [24].

3.2. The blockchain layer

3.2.1. Blockchain stack

The blockchain network in our model is *permissioned*.[9] The participants in the LEM *must* be vetted and whitelisted to ensure they belong to the same distribution service area. This "Know Your Customer" (KYC) task is handled by the overseer of the market — the utility (see Section 1).

A permissioned network configuration means we do not need Proof-of-Work (PoW), as it solves a problem (Sybil attack [69]) we do not have. Consequently, we have no need to incentivize the miners financially, which translates to *no need for cryptocurrency support*. Instead, we resort to classic Byzantine fault tolerant (BFT) solutions, and can use a mechanism such as BFT-SMART [70] as our consensus engine.[10]

We choose Hyperledger Fabric, the system we developed in [30], as the blockchain implementation for the LEM's TMP. Besides supporting the choices above (BFT using the ordering service developed in [71]), it allows us to write smart contracts in a standard, general-purpose programming language, and supports a fine-grained trust model.[11] We leverage both of these features in the text that follows.

3.2.2. Network configuration

There are three key roles in Fabric:

1. Ordering nodes, which package incoming requests (transactions) into blocks and disseminate them to the rest of the network.
2. Peer nodes, which execute transaction proposals, validate incoming transactions, and maintain the ledger.
3. Client nodes, which submit transaction proposals to the peers, and transactions to the ordering nodes, once the appropriate endorsements have been collected.

In our model, the ordering service is delegated to the utility and the ESCOs. The usual BFT caveats apply; these nodes are vetted by the utility so that the probability of one third (or more) of them simultaneously colluding are minimized. In any other case, the network stops producing new batches of bids[12] and the market stops.

We expect participation in the ordering service to be backed by contracts that require a deposit; in case of malicious behavior, the deposit is seized by the utility. Preventing such behavior ensures the network's *liveness*; it is not however enough to prevent censorship of bids (transactions). The ordering service node that leads the current view can still selectively ignore bids, and *will maintain its leadership* for as long as it produces new blocks in a timely manner.

To prevent that behavior, the market needs to ultimately switch to a BFT protocol that follows the rotating coordinator paradigm [72].[13] Then, leaders are rotated periodically, and since not all leaders are Byzantine, any censored transactions *will* eventually go through.

As we will see in Section 4.1.1, energy transactions rarely happen on scales shorter than five minutes,[14] and physical limitations may not allow this length to get any shorter. Such timeframes – which map

[9] As opposed to *public* or *permissionless*; see our taxonomy of blockchains in [25].

[10] Refer to our previous work in [25] for an exposition into the terms used here.

[11] Via endorsement policies at the application (smart contract) level. This allows us to dictate who can invoke what.

[12] In blockchain terms: *blocks*.

[13] N.B. that there is no such implementation for Fabric as of the time of this writing.

[14] "Timescale for energy market services transactions: auctions or agreements for energy exchanges are typically made minutes to day ahead in advance." [15].

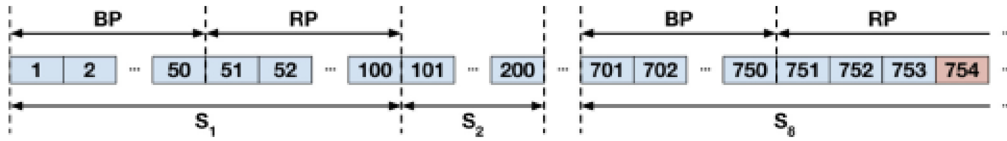


Fig. 6. Each slot consists of two phases; bid (BP), and reveal (RP) — see Section 3.2.4. In the normal case, each phase consists of a fixed number of blocks. Depicted above, a ledger with 754 blocks. If each phase runs for 50 blocks, this means that blocks 751-800 contain all the reveal activity for the 8th slot.

see Fig. 6. Under the conditions described above, the blockchain ledger height acts as a logical clock for each node.

However, this is a best-effort, non-foolproof approximation of a solution; the fundamental problem here is that we are trying to effect synchrony and run a time-sensitive protocol (the LEM) on a medium that is distributed and asynchronous. Consider the case where a network partition prevents the ordering service from reaching a quorum; during that time, no blocks can be cut[22] even if the agents in a partition post transactions to the ordering service within `batchTimeout`.

When network connectivity is eventually restored, and the ordering service reaches a quorum, the current block height (our logical clock) will be offset from the actual, current slot (the wall clock that moves forward regardless of block height progress). Any attempt to resynchronize the two clocks is inevitably non-deterministic. For completeness, we describe one such corrective process in Appendix.

3.2.5. *Data slicing*

When a client posts an offer or a key, they update the state maintained by the smart contract. At first pass, configuring the keys that a smart contract writes to may seem like a nit; however – as we will see in Section 4.2.1 – it can have a significant effect in the performance of the blockchain layer.

Transaction flow. In Fabric, recall that state is maintained in a versioned key–value store (KVS), and the state created by a chaincode is scoped exclusively to that chaincode. At a high level,[23] a successful state update requires the following sequence of steps: a client constructs a *transaction proposal* — a request to invoke a chaincode method with specific parameter values —, then sends it to as many peers as needed in order to satisfy the chaincode's endorsement policy.

Each targeted peer simulates the proposal by executing it against its local state; it does *not* commit the results of this proposal. Instead it returns to the client a *proposal response* that contains the *readset* and *writeset* for that proposal, i.e., the set of keys that were read during the simulation of the proposal along with the version of those keys, and the set of keys that were updated during the simulation along with their new values.

When the client collects enough endorsement responses to satisfy the chaincode's endorsement policy, it assembles them into a *transaction*, and sends that to the ordering service. Eventually every (committing) peer receives this transaction from the ordering service and runs it through the *validation phase*; it checks if it satisfies the endorsement policy of the associated chaincode, and also *checks if the versions in the readset match its current state*. If both checks pass, the transaction is marked as valid, and its writeset is used to update the peer's local ledger.

This transaction flow is depicted in Fig. 7.

The key per slot approach. Given the above, consider the case where we design the LEM's smart contract so that all offers for slot s_i get posted under the same chaincode key k_i. Let us assume the best case scenario where, at the start of s_{i-2},[24] all endorsing peers in the network are up-to-date; each peer carries the exact same keys and values in their ledger, at the exact same versions.

Then, when two or more agents post offers, their simulation proposals will have *the same readset* – since k_i is at the same version across all peers –, and a different writeset — as each participant aims to update k_i with their own offer. Each agent will follow the flow described earlier in this section, by assembling a transaction and submitting it to the ordering service. We now have a race condition; inevitably one of the pending transactions will be processed *first* by all peers; this transaction will pass the validation checks, and will have its writeset adopted as the new state for k_i. All other transactions will *fail* the validation because their readset is no longer current — we call this an *MVCC*[25] *conflict*. This is depicted in Fig. 8a.

The submitters of the failed transactions need to repeat the process from scratch, and hope that they *both* pick up an endorsement from a peer whose state is current (so that the included readset is current), *and* that their own transaction goes through the validation phase *before* any other transaction that writes to the same key.

The key per transaction approach. In our model, we prevent these conflicts by having each smart contract operation – be it an offer, or the posting of a private key – write to a unique key in the contract's key–value store. Specifically, we make it so that each transaction gets posted under a key following the naming pattern `slotNumber-transactionID`, where `transactionID` is a universally unique identification number (UUID) that the submitting client generates. Since all transactions reference *unique* keys in their readsets and writesets, race conditions are avoided, and the read-write conflict check at the committing peers passes. This is depicted in Fig. 8b.

3.2.6. *Bid encryption alternatives*

Participants in our model are responsible for encrypting and decrypting their own bids (see Section 3.2.3). Besides being the most straightforward implementation for a closed order book auction, this is also the option that maximizes user agency.

On the other end of this axis, we would find the *single (fixed or rotating) auctioneer* mode. In it, the market assigns the decryption of bids to a single ESCO, or a set of rotating ESCOs. Then, for the auction in slot s_i, all market participants would post in slot s_{i-2} orders encrypted with the public key pk_i. This corresponds to the private key vk_i of the ESCO that is tasked with running the auction for that slot. At the end of s_{i-2}, the ESCO posts vk_i on the blockchain via a newly-added `markEnd(privateKey)`[26] smart contract method.

Each participant is now able to decrypt all bids, and calculate the market-clearing price for s_i locally. The advantage of such an approach is that we no longer need to split the slot into bidding and key-revelation phases (Section 3.2.3) — the slot in its entirety now can be used for posting bids. Likewise, the number of transactions needed for N agents to communicate (i.e., commit and later on, reveal) their bids decreases from $N \cdot 2$ to $N + 1$.

[22] Remember that in accordance to Section 3.1.4, all energy needs during this timeframe will be satisfied by the grid.

[23] For a detailed explanation, see [30, Sec. 3].

[24] This is the slot during which participants bid for slot s_i.

[25] Multiversion concurrency control.

[26] It *marks* the *end* of a slot.

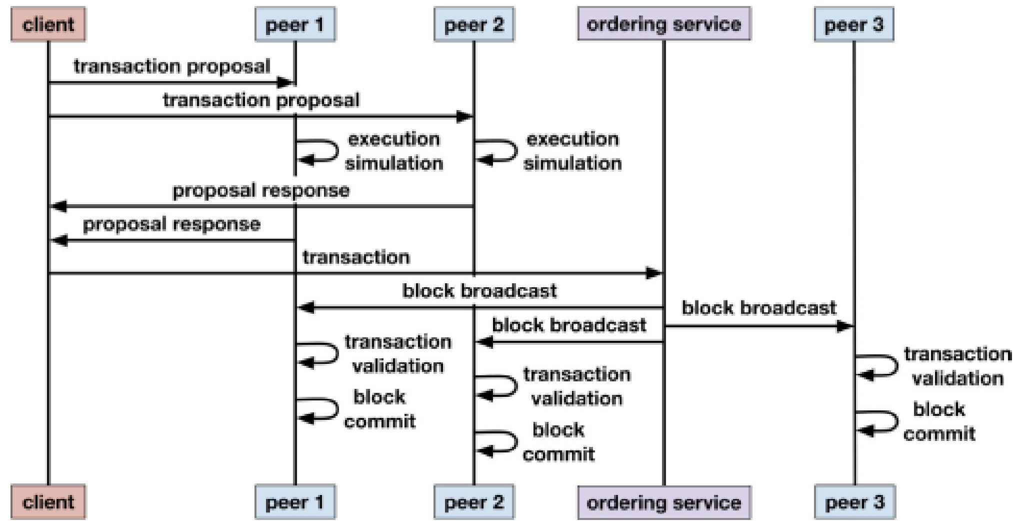
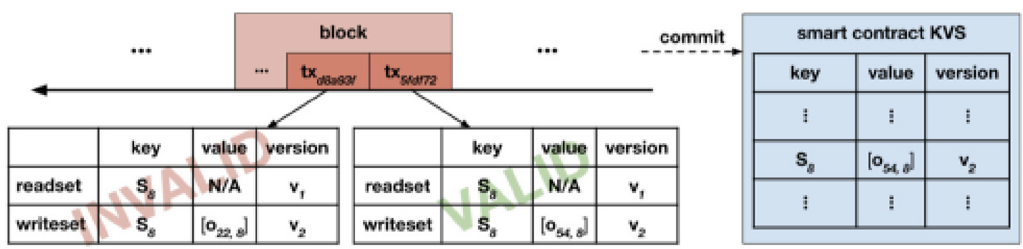
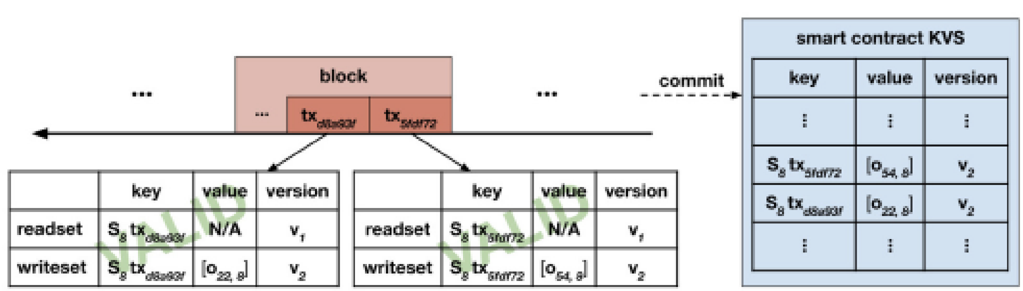


Fig. 7. High-level overview of the typical sequence followed by a transaction in Fabric. In this example the endorsement policy of the chaincode requires endorsements from peers 1 and 2.



(a) The *key-per-slot* approach: all activity for a given slot is recorded in the same key in the smart contract's KVS. In this example, the block contains two transactions that reference the same readset but different writesets, because each contains a different bid. The transaction that gets ordered first (`5fdf72`) passes the validation check on the committing peers, and has its writeset applied to their ledger. When the other transaction (`d8a93f`) gets processed by the comitting peers it fails the readset check due to version mismatch, and gets rejected.

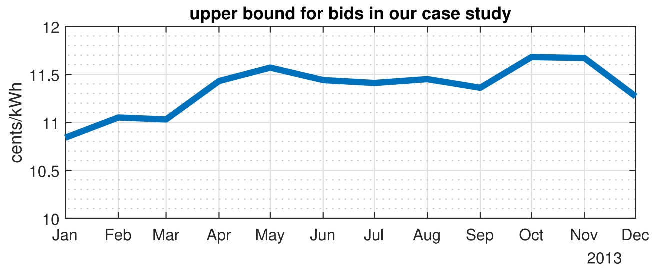


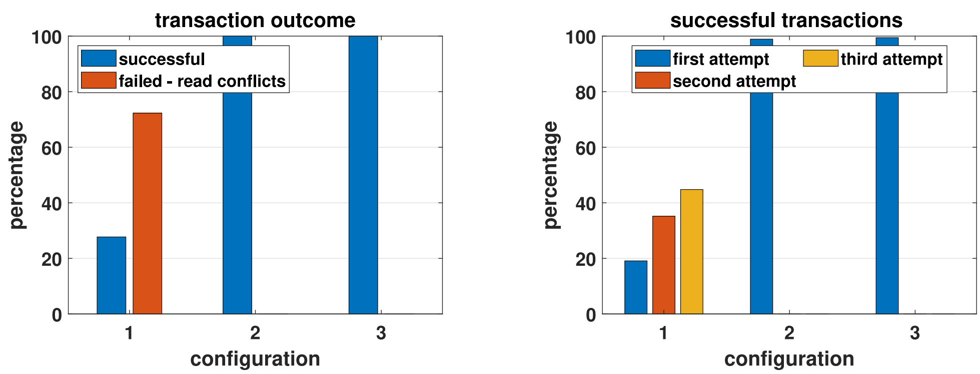
(b) The *key-per-transaction* approach: all activity for a given transaction is recorded in its own key in the smart contract's KVS. In this example, both transactions pass the validation check on the committing peers, as their readsets are current during commit time.

Fig. 8. Two possible ways of storing the data in the key–value store (KVS) of the smart contract that backs the local energy market auction — see Section 3.2.5.

3.2.7. Privacy and security considerations

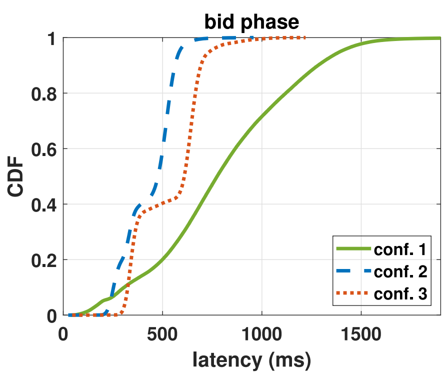
Our work does not consider additional privacy-related measures, other than the standard practice of using a hierarchical deterministic wallet [74] and unique key-pairs for each transaction. This approach may not be tamper-proof, and a malicious actor with enough resources may eventually be able to link all transactions back to the same identity. We deem it as good enough for the privacy model adopted in the work presented here. Our thesis on the privacy front is that energy storage and reputation-based markets will eventually render the privacy concerns irrelevant. As energy storage gets cheaper, the consumers'


upper bound for bids in our case study
cents/kWh
12
11.5
11
10.5
10
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
2013

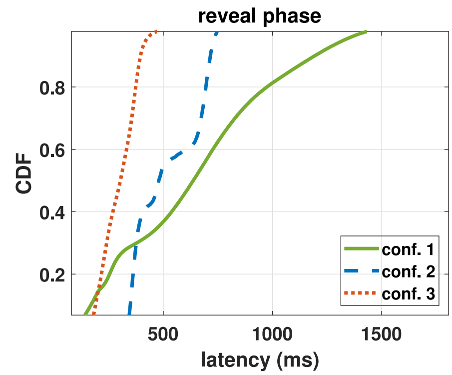


(a) Percentage of transactions that got posted successfully. The success percentage for Configuration 1 is 27.72%.

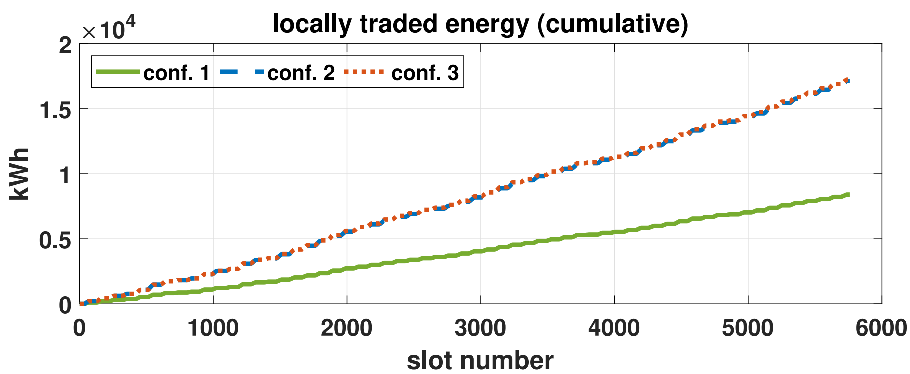
(b) Breakdown of successful transactions by the attempt number that got the transaction through. The numbers for Configuration 1 are 19.09, 35.18, and 44.79% for the first, second, and third attempt respectively.

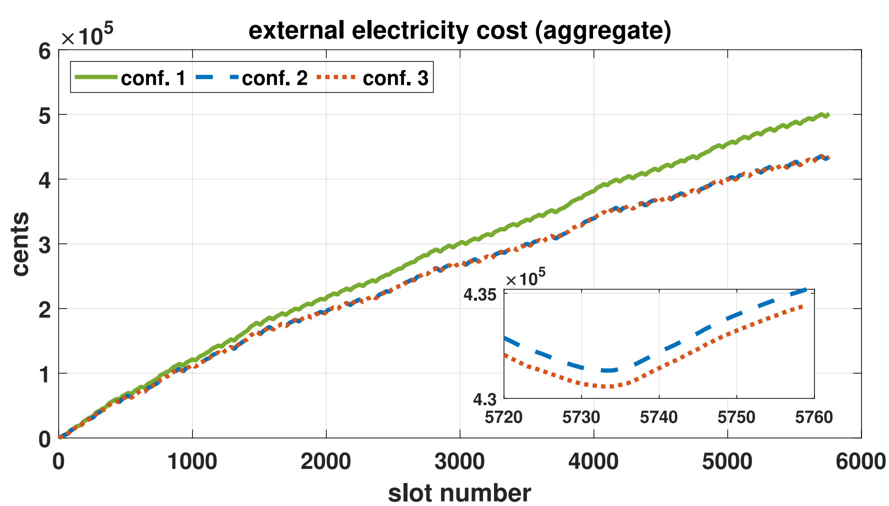


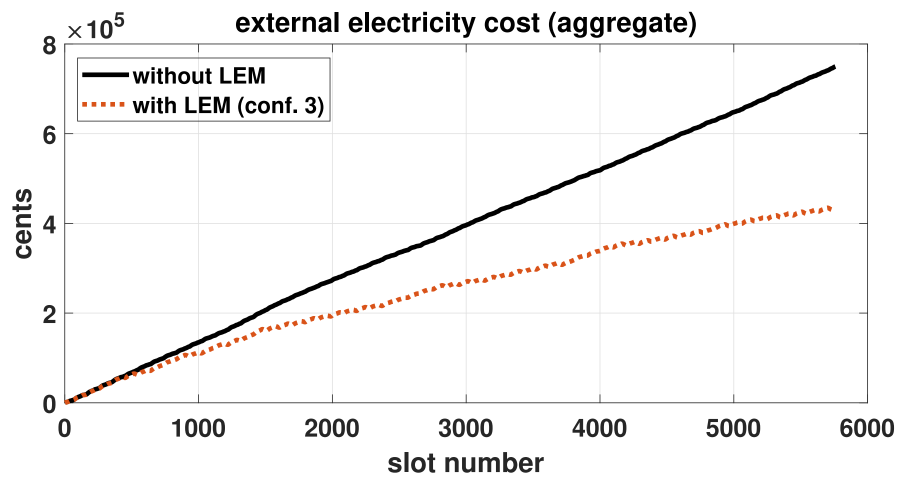
(a) Cumulative distribution function for the latency of successfully posted transactions across all considered configurations during the bid phase. Median (95th percentile) values are 774.4 (1395), 477 (585), 609 (737) milliseconds for Configurations 1, 2, and 3 respectively.

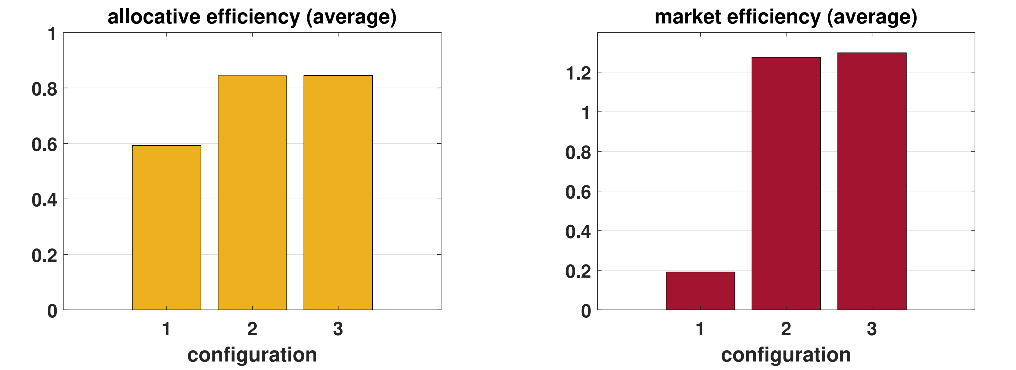


(b) Cumulative distribution function for the latency of successfully posted transactions across all considered blockchain configurations during the reveal phase. Median (95th percentile) values are 637 (1328), 479.5 (731.5), 303.5 (420) milliseconds for Configurations 1, 2, and 3 respectively.






external electricity cost (aggregate)
×10^5
without LEM
with LEM (conf. 3)
cents
slot number
0
2
4
6
8
0
1000
2000
3000
4000
5000
6000



(a) Market efficiency. Averages are 0.1915, 1.275, and 1.297 for Configurations 1, 2, and 3 respectively.

(b) Allocative efficiency. Averages are 0.5926, 0.8440, and 0.8451 for Configurations 1, 2, and 3 respectively.