Abstract

CHRISTIDIS, KONSTANTINOS. Blockchain-Based Local Energy Markets. (Under the direction of Michael Devetsikiotis and Srdjan Lukic.)

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. Blockchains — systems that have been popularized recently by technologies such as Bitcoin and Ethereum — allow us to establish transparent marketplaces without the need for a central authority. This thesis investigates the feasibility of local energy markets (LEMs) running on blockchains, and also introduces a canonical framework for designing and evaluating blockchain-based LEMs — a first in this space. We begin by examining whether existing blockchain implementations are capable of supporting such markets. We dissect blockchains into their core components, perform an analytical survey on the space, and introduce a taxonomy for blockchain systems. Our findings suggest an impedance mismatch for our use case; we identify a number of integration issues for IoT applications, and offer workarounds where possible. Shifting back into the original goal of designing a realistic blockchain-based LEM, a common theme we find across all relevant literature is the treatment of the blockchain component as a black box. Armed with a proper understanding of the blockchain space from our earlier analysis, we make the case that this approach is flawed because the choices in this layer affect the market's performance significantly. First, 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. We then create a model, configurable blockchain-based LEM: we design and present in detail the underlying smart contract architecture, the operational parties, and their roles. We also design and implement the market mechanism that sits atop the blockchain layer; to the best of our knowledge, our work is the first to explicitly identify how a closed-order book double auction can be implemented on a blockchain-based LEM. A numerical evaluation demonstrates the

applicability of our model to a real-world case, and its ability to provide insights on parameter choices during both market infrastructure planning and day-to-day LEM operation. Our case study, based on residential electricity usage data, shows how a change in the blockchain data model can decrease the market efficiency by approximately 90%, or — as another example — how a change in the way bids are encrypted can result in further market improvements, but 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.

Blockchain-Based Local Energy Markets

by
Konstantinos Christidis

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Computer Engineering

Raleigh, North Carolina

2020

APPROVED BY:

Subhashish Bhattacharya	Joseph DeCarolis
Michael Devetsikiotis Co-chair of Advisory Committee	Srdjan Lukic Co-chair of Advisory Committee

Dedication

To my family.

Biography

Konstantinos Christidis received the Dipl. Ing. in Electrical and Computer Engineering from the Aristotle University in Thessaloniki, Greece in 2011, and the M.Sc. in Computer Engineering from North Carolina State University in 2013. He is currently a PhD candidate in the school's Computer Engineering department, focusing on blockchain-based local energy markets. He is the recipient of an IBM PhD Fellowship. He is the co-creator of Fabric, an open-source blockchain project hosted by The Linux Foundation. Fabric is used by more than half of the companies in the "Forbes Blockchain 50" [50] list. He works full-time as a Senior Software Engineer at Netflix. Prior to that, he was the Technical Lead for IBM Blockchain. His research interests lie in the area of distributed systems, particularly transaction processing and consensus protocols.

Acknowledgements

To my advisor, Prof. Michael Devetsikiotis — thank you for encouraging me to explore a research problem that would also end up kickstarting my professional career, and for guiding me to get this line of work to the finish line. I am incredibly grateful for your mentorship and constant support.

To my committee; Prof. Lukic, Prof. Bhattacharya, Prof. DeCarolis — thank you for graciously donating your time whenever I would ask for it, for providing me with valuable feedback, and for steering my work to the right direction.

To my former employer, IBM, and esp. Andy Rindos and Jerry Cuomo; thank you giving me a chance to work on this area, and for supporting my work with the IBM PhD Fellowship award. To Meeta Vouk, and John Cohn — the first time I heard about blockchains was through you; I am grateful to you both for helping me out when I was starting in this area. Patrick Morrison — thank you for encouraging me to push through this, or for offering to brainstorm with me, esp. while I was trying to juggle finishing this work and working full-time.

To my colleagues from IBM and IBM Research with whom I collaborated for the work on Fabric; esp. Chet Murthy, Marko Vukolić, and Jeff Garratt — I had a blast working with you, and learned a ton from you all; osmosis works! I hope our paths meet again.

To the numerous friends I made at EB2 — thank you for making this a ride to remember.

To my family, and esp. to my wife, Yun — thank you for being there unconditionally.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

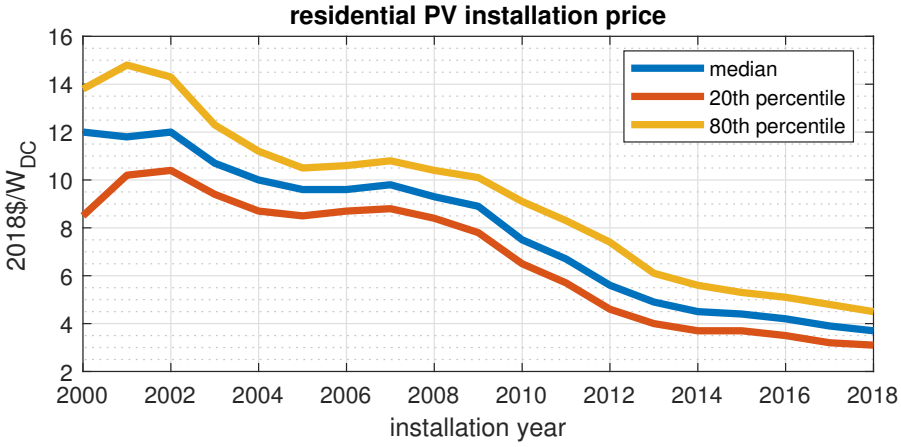
1.1 Context

1.1.1 Solar PV and Battery Storage Markets Keep Growing

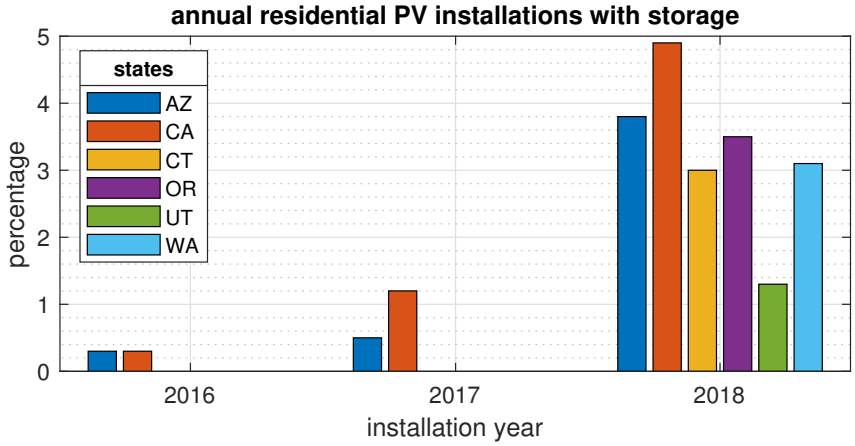
For the past year through Q3, solar photovoltaics (PV) accounted for 39% of all new electricity-generating capacity installed in the US [174]. In Q3, 2.6 GW_{dc} of capacity was added — up 45% from the quarter one year-ago —, and the residential sector had its largest quarter ever with 700 MW_{dc} — 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 [22] (Fig. 1.1a).

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 [222]. Moreover, the residential sector had a record-breaking quarter, with 40 MW_{dc} of newly installed capacity [207]. 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) [222]. Price-wise, costs for lithium-ion battery packs have fallen 85% from 2010 to 2019, to 176 USD/kWh [100], driven mostly by mass production of batteries for electric vehicles.

PV panels and battery storage make a powerful combination; a battery can store excess



(a) Residential PV installation price from 2000 to 2018 [22].



(b) Percentage of annual residential solar-plus-storage installation relative to annual residential plain solar installation [22].

Figure 1.1 Trends in the residential solar-plus-storage sector.

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% [136]. Fig. 1.1b tracks the increasing percentage of households that opt for both solar and storage across 6 US states.

1.1.2 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 [35], 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 [208]. The reference case (AEO2020) from the most recent Annual Energy Outlook released by the EIA, projects an annual growth in electricity demand that averages just about 1% throughout the projection period (2019-2050) [13].

These trends call for a redesign of the electricity business model. Under the existing regulation, the revenue that utilites make depends directly on the volume of retail sales[1]—reduced power consumption translates to less profits and the risk of *stranded investment costs* [188]. This leaves the utilities at a dead-end; if they do nothing, their revenues decay; if they penalize the solar customers — by means of lowering net metering payments or imposing additional fixed charges [123] —, they may delay revenue loss temporarily, but ultimately will only accelerate the solar-plus-storage trend. Changes in net metering and time-of-use rates *increase* the value of solar energy stored in batteries and discharged later in the day, a dynamic that is already driving battery adoption in Hawaii and Arizona for instance [207].

[1]Remember that the retail rate charged by utilities is a *bundle* of two components: one that covers the utility's investment costs, and one that covers the fuel costs. The latter is revenue neutral, i.e., the utilities do not make money on *the actual energy* they sell [137], but their fixed costs are recovered *through* charges based on how much electricity their customers use [168].

1.1.3 Transactive Energy and Performance-Based Regulation

Proposed business models heavily revolve around two concepts; that of *transactive energy* (TE) [109], and *performance-based regulation* (PBR). In TE, we take advantage of the deployment of two-way communications capabilities and intelligent, communicating sensors and devices and use market-based constructs to dynamically balance supply and demand, while considering grid reliability constraints [204, 218].

With PBR, profits are based on performance goals set by the local Public Utilities Commission (PUC), emphasizing results for customers and system efficiency [221], rather than capital expenditures [146]. Example goals include: reliability (System Average Frequency and Interruption Duration Indexes[2]), peak reduction, power balancing, environmental impact (CO_2/kWh), total cost per customer [114], and customer satisfaction[49].

Smart grid technologies then, turn the utilities into distribution wires companies that become platform service providers. They facilitate the transaction of independent, distributed agents in the electric network, enable experimentation at the grid edge, and lower transaction costs [137].

This is not merely a theoretical proposition. On the regulatory front, this is the direction that the New York Public Service Commission (PSC) is setting with their "Reforming the Energy Vision" (REV) [49] program. There, utilities operate a retail electricity market where third parties — commonly refererred to as third-party grid service providers (GSPs) or energy service companies (ESCOs) — can compete to provide products and services on the retail side, such as energy storage, demand response, and distributed generation [187]. The utilities themselves are rewarded in a PBR fashion[3].

[2]Referred to as SAIFI and SAIDI in the utility industry respectively [3]. SAIFI measures the average number of interruptions per customer per year, while SAIDI measures the average length of each interruption.

[3]Within the REV context this is captured by the Earnings Impact Mechanisms (EIMs) [48] concept.

1.1.4 The Case for Blockchain-Based Energy Markets

One direction we seem to be headed then, is that of energy markets at the distribution level, that operate on a varying degrees of decentralization. The question that arises is: *how does the underlying transaction management platform*[4]*(TMP) look like?* As we will see in Chapter 2 (and our work in [56]), blockchains allow us to build *transparent* digital marketplaces with *verifiable processes*.

Transparency is important because third-party vendors need access to a detailed market load profile and grid utilization data [144]; this allows them to optimize the placement of their resources (DERs), and make an informed choice on whether or not they should be participating in a certain market to begin with. SolarCity, a GSP that manages and deploys DERs, has expressed this exact request [2]. Community-choice aggregators in the San Francisco Bay Area shared anonymized customer data with the vendors for that same reason, when soliciting 30 MW of storage-plus-solar capacity last year [206].

Ultimately, transparency lowers the barrier to entry for GSPs and allows them to build more competitive businesses. Verifying the integrity of market processes assuages fears of maladministration by the market operators. As an example[5], if we are dealing with auctions, we can verify that no late bids were included when they are all published to a blockchain, even if they are encrypted [39].

[4]Recall that this is the layer that handles all market-clearing functions in a way that balances supply and demand in the local market [144].

[5]As another example, consider the case in 2016, where a datacenter operator based in Austin, TX, filed a complaint [156] against the local utility, Austin Energy, accusing them of trying to recover the production costs of its wholesale generating business through retail base rates [156]. According to the arguments presented in the complaint, Austin Energy attempted to recover the variable production costs associated with its wholesale endeavors twice: once from base rates, and then again from wholesale revenue. The utility would refuse to provide exact revenue figures for its generating business, or a sufficient justification for including its costs in the rate base. As the complaint noted — "AE has the burden of proof, and it failed to carry that burden."

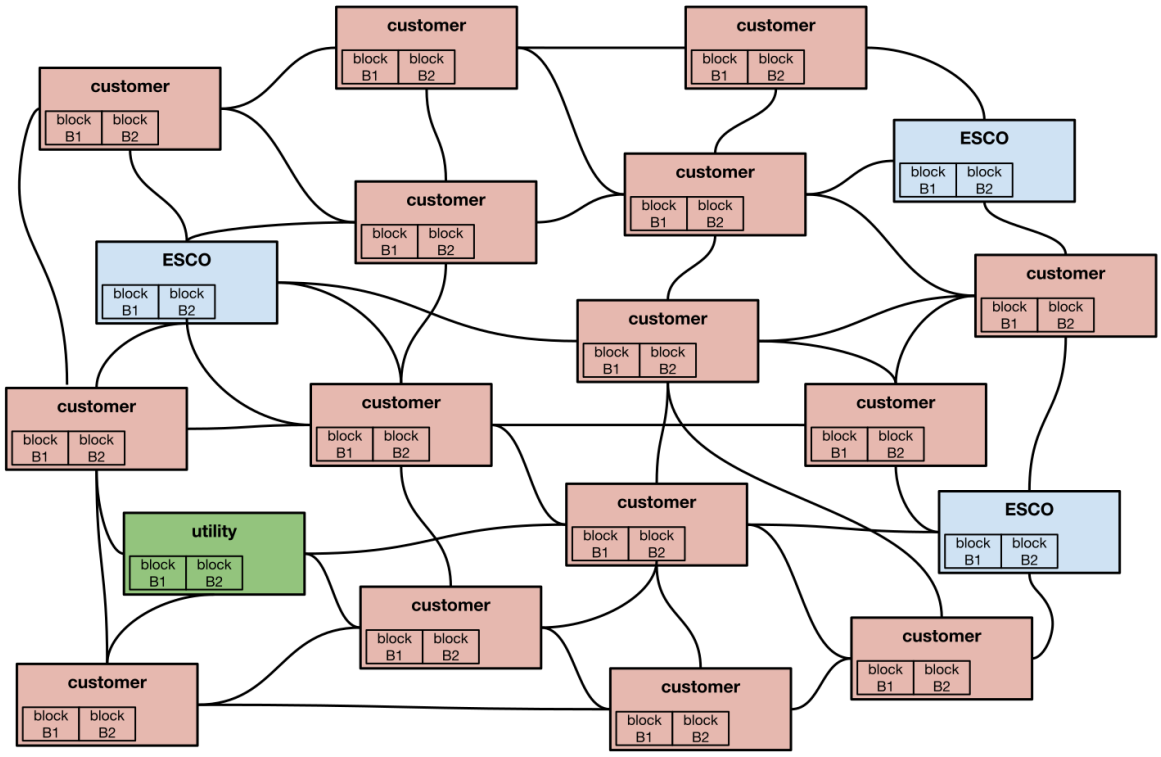


Figure 1.2 Conceptual diagram for a blockchain-based local energy market that spans the area served by a distribution substation. Third-party energy service companies (ESCOs) sell services such as distributed generation, storage, or voltage regulation to consumers. [204] contains an extensive list of transaction-based energy services that can be exchanged on the distribution level, along with technical requirements to fulfil 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.

1.2 Thesis Structure and Summary of Contributions

The questions that ultimately drive the rest of this work, then, are —

How does a realistic blockchain-based local energy market (LEM) look like?

What should we factor in when designing and implementing such a market?

How do we assess whether one design is better than the other?

In general, we find that a canonical framework for designing and evaluating blockchain-based LEMs is missing. The work presented here is an attempt to fill this gap.

We begin by examining whether existing blockchain implementations are capable of supporting such markets, in Chapter 2. We dissect blockchains into their core components, and perform an analytical survey on the space; we also introduce a taxonomy for blockchain systems (Section 2.2). We then study the impendance mismatch that comes up when considering a blockchain-based IoT application (Section 2.3); we explicitly identify integration issues, and offer workarounds where possible (Section 2.4). The work backing this chapter has been cited more than 2,000 times as of the time of this writing.

We then move into building a blockchain system that can be more readily adapted for energy applications. In the work that we have contributed to in Appendix A, we advance the state of the art (Section A.2) by designing a system that is both fast — in terms of throughput —, and flexible — in terms of the trust models it can accomodate, since it decouples the application trust model from the consensus trust model (Section A.3.5). Existing blockchains traditionally follow a *order-execute* model, which dictates that all nodes need to execute every transaction, and all transactions need to be deterministic — otherwise the ledger copies on each node may diverge ("fork"). With the design in Appendix A however, we switch to a *execute-order-validate* model (Section A.3), where it suffices to execute the transaction in a subset of nodes, then communicate *the result* of this execution to the rest of the network. Our

work is centered on the ordering part (Section A.4.2). This new blockchain platform achieves a throughput of 3,000 TPS compared to 7 TPS for Bitcoin, or 12 TPS for Ethereum (Section A.5). The work backing this chapter has been cited more than 1,000 times as of the time of this writing.

In Chapter 3, we identify a common theme with the state of the art in the blockchain-based local energy market space; all pieces of existing work in this area focus mostly on the local energy market part, and treat the blockchain component as, more or less, a *black box* (Section 3.2). As we note in Section 3.1.1, when setting up a blockchain-based system, a number of basic and important questions have to be considered and answered. Depending on the blockchain stack that is used, not all of these questions may apply directly, but the general observation stands: none of the answers to these questions are given, and it is the answers to these questions that dictate whether the proposal makes sense as a blockchain-based application, how that application performs in terms of throughput, and how this affects the market that is provisioned on top of it. Put differently, blockchains reify protocols, and the configuration options offered draw a pretty wide design space. Questions along the lines stated above *must* be answered, and the answers should also be considered for second-order effects. In Chapter 3 then, we design a blockchain-based local energy market by carefully considering the questions above, and reasoning over their implications (Section 3.3). Our work sets precedence on the blockchain design space definition (Section 3.3.2), and explicitly analyzes its impact on the performance, governance, and decentralization of the LEM. We utilize the open-source blockchain platform developed in Appendix A for the blockchain layer of our market (Section 3.3.2), and design and present in detail the underlying *smart contract* architecture (Section 3.3.2), the operational parties, and their roles (Section 3.3.2). We also design and implement the market mechanism that sits atop the blockchain layer; to the best of our knowledge, our work is the first to explicitly identify *how* a closed-order book double auction can be implemented on a blockchain-based LEM (Section 3.3.2). Finally, a numerical evaluation (Section 3.4.1) demonstrates the applicability of our model to a real-world case,

and its ability to provide insights on parameter choices during both market infrastructure planning and day-to-day LEM operation. Our case study, based on residential electricity usage data, showed (Section 3.4.2) how a change in the blockchain data model can decrease the market efficiency by approximately 90% (switch from Configuration 2 to Configuration 1), or — as another example — how a change in the way bids are encrypted can result in further market improvements, but at the risk of subverting the proper operation and resilience of the market (switch from Configuration 2 to Configuration 3). The simulation framework that we developed for the numerical evalution has been open-sourced and is available in [60].

Before wrapping up this work, we consider yet another application for smart contracts in the energy domain (Chapter 4). Compared to our work from the previous chapter, our focus here shifts away from the blockchain component and towards the algorithmic side of the problem. Specifically, we look into a cluster of buildings within proximity that share a large-capacity battery for peak-shaving purposes, and draw power from the grid at a premium once they reach a certain threshold (Section 4.3). We then identify a resource allocation policy that can be implemented via a smart contract, which minimizes the amount of energy the cluster draws at a premium, while also ensuring fair access to all of its members (Section 4.4). This adaptive policy allows for maximum energy savings when the network load is low, and ensures fairness when the aggregate power level is high (Section 4.5); this makes it suitable for a network where equal weight is given to both cluster-wide cost minimization and fairness among all customers.

We conclude this thesis by identifying a few possible directions for future work in Chapter 5.

Chapter 2

Blockchains and Smart Contracts for IoT Markets

An edited version of this chapter was originally published in [56].

2.1 Introduction

Blockchains have recently attracted the interest of stakeholders across a range of industries: from finance [132] and healthcare [130, 210], to the energy sector [147], real estate [164, 171], and the government sector [102].

The reason for this surge of interest is that with a blockchain in place, certain applications that could previously run only through a trusted intermediary, can now work the same way without them — this was simply not possible before. The removal of trusted middlemen means decreased operating costs and faster reconciliation for transactions. Practically every interaction in a blockchain network requires the use of cryptography — this brings authentication and integrity to the table, and it is what allows non-trusting parties to transact with each other. Trust has a financial cost, and blockchains allow us to decrease it. Finally, smart contracts —self-executing scripts that reside on the blockchain— integrate these concepts and allow us

to automate entire processes, or sequences of transactions.

That said, blockchains are not a silver bullet; for most business applications a blockchain either does not make sense to begin with, or it is unable to fulfill existing business requirements. The goal of this work is to allow readers to identify *when* such an integration makes sense, and *how* exactly they would go about doing this integration — both areas that, at the time of writing this, have remain unexplored in the literature.

The chapter is structured as follows. In Section 2.2 we review the underlying mechanisms of blockchains, and identify how blockchain networks and smart contracts work; we conlude the section with an emerging taxonomy for blockchains. In Section 2.3, we examine the applicability of blockchains in the IoT sector, and — based on our own experience in the field — we identify potential issues and workarounds for such applications in Section 2.4. We conclude with our thoughts on the integration of IoT applications in the blockchain domain in Section 2.5.

2.2 Preliminaries

2.2.1 How Blockchains Work

Technically, a blockchain is a data structure. It is a linked list, where each node:

1. Contains a batch (a 'block') of transactions, and
2. Establishes a link to the *previous* node by referring to the hash of that node's contents, i.e. by using a hash pointer [113].

As we see in Fig. 2.1, this gives us a *chain* of *blocks* — hence the blockchain name.

References to this concept date back to more than 2 decades ago [167]. The interest on blockchains today however is primarily due to the popularity of Bitcoin [166]. The peer-to-peer electronic cash system relies on the blockchain construct, along with the proof-of-work (PoW) mechanism and the 'heaviest chain' rule (see Section 2.2.2) to solve the double-spending

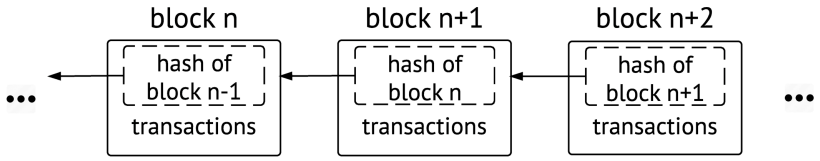


Figure 2.1 Each block in the chain carries a list of a transactions and a hash to the previous block. The exception to this is the first block of the chain (not pictured); it is known as the *genesis block*, and has no parent.

problem [14], i.e. the attempt to spend the same token on more than one transaction. As a result of how the nodes on the Bitcoin network append validated, mutually agreed-upon transactions to the blockchain, the network's blockchain is the authoritative ledger of transactions which establishes who owns what. Any node with access to this ordered, back-linked list of blocks [14] can go over it and eventually reconstruct the "state of the world". Thus, *practically*, a blockchain is both a data structure and an associated protocol.

We get a better understanding of how a blockchain works, when we examine how a *blockchain network* runs. A network is comprised of nodes, all holding a copy of the same blockchain. A node acts as the entry point for blockchain users into the network; this is usually a one-to-many relationship, but for simplicity we can assume that each user has their own node. These nodes then form a peer-to-peer network where they perform the following sequence ad infinitum:

1. Users interact with the blockchain via a pair of private/public keys [78]. They use their private key to sign their own transactions, and they are addressable on the network via their public key[1]. The use of asymmetric cryptography brings authentication, integrity, and non-repudiation into the network. Every signed transaction is broadcasted by the ingress node to its one-hop peers.

2. The neighboring peers make sure this incoming transaction is valid before relaying it further; invalid transactions are discarded. Eventually this transaction is propagated across the entire network.

[1]Depending on the implementation, the address can be the public key itself or (usually) a hash of it.

3. The transactions that have been collected and validated by the network using the process above during an agreed-upon time interval, are ordered and packaged into a timestamped candidate block by a node that *leads* that round[2]. The leader node broadcasts the candidate block to the network.

4. The nodes verify that the suggested block carries valid transactions and the proper hash pointer, i.e. it references the current tip (most recent block) of the blockchain. If that is the case, they add the block to their local copy of the blockchain, apply the included transactions, and update their world view. If that is not the case, the proposed block is discarded. This marks the end of a round.

With this sequence, the blockchain copies at each node are kept in sync. When we refer to *the* blockchain in a network then, we refer to this logical construct that every node essentially replicates.

When each node in the network follows the steps listed above, the underlying blockchain contains an authenticated and timestamped activity log for the network. Since every transaction is cryptographically signed, the nodes do not blindly trust what they see; instead they verify that the transactions are properly authenticated. This is why we often talk about reducing trust when we describe blockchains; trust is not granted but achieved as an emergent property from the interactions of different participants in the system [14].

Perhaps the most useful model for a blockchain network is that of a database with shared *write-access* [106]. In a blockchain network, this database is the blockchain, and the writers — who do not trust each other — modify its rows[3]. Each row of the database is mapped to a public key (or address) that dictates its editor.

[2]The choice of the leader node, as well as the contents of the block, depend on the consensus mechanism used by the blockchain network — see Section 2.2.2.

[3]Technically, they can only *append* new rows and invalidate existing ones by referencing them in the new rows.

2.2.2 Consensus Mechanisms

Nodes must agree on the blocks that get appended to the blockchain, otherwise we get forks, i.e. conflicting blockchains, and the utility of the network breaks down; nodes cannot progress. This means of agreement is encapsulated in what is referred to as a *consensus mechanism*. It consists of rules that are programmed into the protocol and used by each node on the network to decide on:

1. The validity of a proposed transaction[4]
2. The validity of a proposed block
3. The order with which valid proposed blocks get appended to the blockchain

What makes a proposed transaction valid?

In the shared write-access database model (Section 2.2.1), a transaction is deemed valid if it comes with a signature which matches the public key of the row it wishes to modify. The protocol may require that the content of the transaction satisfies additional, application-dependent predicates based on the row-to-be-modified. Nodes will propagate/gossip an incoming transaction that is deemed valid, and drop invalid ones. A node can only assess whether an incoming transaction is valid based on its *current* state, as described by its local copy of the blockchain; a transaction x attempting to modify row r may well be deemed valid by node n at time t and propagated into the network, but it may become invalid at time t' ($t' > t$) if a block is appended to the blockchain that does not include x but modifies r.

What makes a proposed block valid?

A proposed block is deemed valid if its hash pointer references the current tip of the blockchain, i.e. its most recent block at the time of the proposal, and it includes valid transactions. Nodes drop proposed blocks that are deemed invalid.

[4]We touched on this in Step 2 of Section 2.2.1 as well.

How do nodes agree on the order of valid proposed blocks?

First, it is useful to see *why* such an agreement is needed. Based on the rules we have established so far, we may well end up with a situation where blocks b_1 and b_1' are both valid (see Section 2.2.2) and get proposed by the network at the same time; then, a subset of the network may append b_1 to its blockchain, while the rest of the network will go with b_1' — we then have a fork, and the network breaks down.

A blockchain network avoids[5]forks by deciding [201]:

- Who gets to vote, and
- How these votes are tallied

These decisions depend on the network's performance goals and security characteristics (Section 2.2.5).

For instance, if the membership of the network is controlled, i.e. the nodes are known and whitelisted, every node can be given the right to vote. Then, the group can reach consensus even if certain nodes act maliciously (Byzantine nodes) if we pick a vote-tallying mechanism from the domain of Byzantine fault-tolerant (BFT) state-machine replication (SMR) protocols [225]. Practical Byzantine Fault Tolerance (PBFT) [52] is the first and perhaps most popular algorithm of that family. It provides a solution to the Byzantine Generals Problem [148] for asynchronous environments like the Internet, assuming that less than one third of the nodes in the network are malicious[6]. PBFT is a leader-based protocol. The leader ("primary") proposes blocks to the network, which get appended to the network's log, if they gather enough votes by the rest of the network ("replicas"). If the leader crashes or behaves maliciously, the protocol provides a mechanism ("view-change") that allows the replicas to elect a new leader.

[5]Or *resolves* temporary forks, as is the case with Bitcoin for example.

[6]Let N/f be the number of all/faulty nodes in the network. The PBFT algorithm works only if $N \geq 3f + 1$. When $N = 3f + 1$ the quorum/vote thresholds listed in [52] are correct. When $N > 3f + 1$ however, those thresholds may lead to forks and those in [42] should be used instead. This is the case for our PBFT implementation in v0.6 of Hyperledger Fabric [41].

Sieve [43] builds on the PBFT algorithm and filters out non-deterministic client requests. Such requests are not allowed in blockchain networks because they may lead to forks: if transaction t modifies node n_1's state in a different way than it does for node n_2, these two nodes may not able to agree on the appending of subsequent blocks to their ledger; their states have diverged. Sieve filters out those transactions by adding speculative execution and verification phases to the protocol, as in the execute-verify architecture of [128]. It therefore enables the network to not only consent on the input order of the transactions, but also on their outputs.

In an open network where unvetted nodes can come and go as they please and Sybil attacks [73] are a valid threat, a single actor may join with multiple identities/nodes, cast multiple votes, and thus steer the network in a direction that serves their own interests. In other words, a minority could seize control of the network. In such networks, classical BFT approaches will not work.

Bitcoin works around the Sybil attack problem by making this vote casting computationally expensive — a concept first proposed in [76] as a means to combat spam. Then, even if one wishes to impersonate multiple entities, they are bound by the computing resources available to them. Specifically, a Bitcoin node can propose a block if they solve a cryptographic puzzle[7]— the solution computation for that puzzle is expensive, but the solution verification is cheap. The solution to this puzzle is the node's proof-of-work [166] and grants it the right to propose this *mined block* to the network for appending to the blockchain.

Proof-of-work alone does not suffice to solve the original problem, i.e. agreeing on the order of valid proposed blocks, because we may still have more than one valid blocks mined/proposed by competing nodes at the same time. The Bitcoin protocol dictates that nodes should adopt the chain with the heighest cumulative difficulty ('heaviest'[8]chain) as the

[7] A node (i) picks any valid, non-committed transactions it wishes to append to the candidate block, then (ii) tries to calculate a random number (nonce) that will make the block header's hash ($SHA-256\hat{2}$) have the *right* amount of leading zeroes. The right amount is decided by the network every 2,016 blocks. The number of leading zeroes adjusts the puzzle's difficulty: the more leading zeroes required, the more difficult the solution to the puzzle is. The network then adjusts the amount of zeroes so as to account for changes in the network's collective CPU hash power – it wishes to have 1 block generated approximately every 10 minutes.

source of truth. The way this plays out in practice is: two nodes n_A and n_B propose a valid block of the same difficulty around the same time, and the network is temporarily forked with one subset of it siding with the n_A block, and the other with the n_B one. This fork is resolved automatically by the next block via the heaviest chain rule. One of the two forks will almost certainly mine the next block first, so its chain becomes the heavier one, forcing the other fork to switch over.

Proof-of-stake (PoS) [40] is an alternative to proof-of-work. As is the case with PoW, it dictates who gets to vote, or propose a block. Unlike PoW, proposing a block does not require solving cryptographic puzzles. A node simply has to "lock" a certain amount of tokens (issued in the blockchain's base cryptocurrency — see Section 2.2.3) into a deposit before they are able to propose blocks to the network. This process is called *staking*, and we say that the staking node is now eligible to become a *validator*. The higher the stake, the higher the probability with which this validator proposes blocks to the network, and collects rewards. If a validator behaves maliciously however, their deposit is forfeited by the network — staking then is a means to incentivize honest behavior.

The two most common ways [179] for a PoS-based network to reach agreement on the order of proposed blocks are:

- Chain-based, where the algorithm pseudo-randomly selects a validator during each time slot, e.g. based on the hash of the most recently added block and how this maps to the current validator set. That validator is then tasked with proposing a valid block.
- BFT-style, where the validators are randomly assigned the right to propose blocks, but a multi-round voting process by all validators ensues before the network agrees to append the proposed block to the blockchain.

Proponents of PoS-based consensus mechanisms point to its reduced energy consumption

[8] Also referred to in the literature and in the original Bitcoin paper [166] as 'longest' chain. We find that this term can be misleading though; the number of blocks in a chain does not matter; the difficulty of each block does. A shorter — in terms of number of blocks — chain will be preferred over a longer one, if it is heavier.

compared to PoW-based approaches, among other benefits [40]. The very fact that block proposal is not computationally costly however, exposes PoS to so-called Long Range attacks [67], wherein new nodes or nodes that have been offline for a significant amount of time may be tricked into adopting a chain different to the main one, as the source of truth.

Our own experience in the field is aligned with the conclusions of [44]; consensus protocol development should not be undertaken in an ad-hoc manner, and protocols that do not come with a formal proof of correctness should be dismissed. Consider the case of Tangaroa [63] a consensus protocol that is in use even though it violates basic consensus properties [44].

[45] compares the consensus protocols of permissioned blockchain platforms, with respect to their fault models and resilience against attacks. [225] compares PoW-based and BFT SMR-based blockchains in terms of scalability. [220] evaluates several *proof-of-X* mechanisms.

2.2.3 Tokens and Digital Assets

In Section 2.2.1, we modeled the blockchain as a database with shared write-access. Every row in that database carries a value and is mapped to the public key of its owner. That value can range from gibberish, to a string that has special meaning in the "real" word according to a legal contract that has been signed by the blockchain network participants before their onboarding, to a set of tokens issued in the blockchain's native cryptocurrency, or even to a script that reacts to whatever it sent to it, in the case of blockchains supporting smart contracts (see 2.2.4).

In either case, the value is simply a digital piece of data; what makes it *valuable* is the fact that it is *unique*. Bob may own a row with value `foo`, and Alice may own a row with value `foo` as well, but `foo` in both cases is unique; we are dealing with two separate pieces of data here, that can also be tracked separately; `foo` that originated from Bob, and `foo` that originated from Alice.

As a consequence of the fact that all of these values are created and then manipulated on the chain, we have a mechanism in place that allows us to create scarcity of digital things [93]

— and that is intrinsically valuable [153]. We therefore use blockchains to issue digital assets and we can even split those assets into tokens and track them all separately.

Then, since a blockchain network allows appropriately authorized clients to modify existing rows or create new ones, we also have an asset/token trading mechanism in place. So blockchains enable the tokenization of digital assets, and the creation of deeper markets around them, thereby increasing the liquidity of those assets [158].

2.2.4 Smart Contracts

So far we described how blockchains can be used as distributed ledgers, tracking the provenance of digital assets (Section 2.2.3). Blockchains however, can be extended to perform general-purpose computations. They do so when they support *smart contracts*. These are scripts that clients deploy *on* the blockchain. As is the case with a standard blockchain transaction that moves assets on the blockchain:

1. The contents of these deployment transactions are public; any client wishing to interact with a smart contract then, can inspect its source code in advance.

2. Any tampering of the transaction's contents (i.e. the source code of the smart contract) will be evident, since the signature of the tampered source code will not match the signature the client provided upon deployment.

A successful deployment transaction results in the smart contract getting installed in all (or, as is the case with our work in [11], a select subset) of the nodes in the network. Going back to the shared database model (Section 2.2.1), every contract is a row in that database; as such, it has its own address and can maintain its own balance of assets. The difference with a regular account (row), is that a contract reacts to transactions clients sent to its address, based on the logic encoded in it and the contents of the incoming transactions.

The code of a smart contract is executed on nodes across the blockchain network, and affects what each node records on their local copy of the blockchain. It is imperative then

that the contract's behavior is deterministic; for the same input, it should generate the same output. Otherwise, nodes may end up updating ther local ledgers differently, the network forks, and its functionality is crippled. Blockchains traditionally achieve this determinism by only supporting domain-specific programming languages for smart contracts that lack non-deterministic constructs; for example, there is no random generator function in Ethereum's Solidity language. Our work in [11] takes a different approach. It supports general-purpose programming languages, complete with their non-deterministic functions, but filters out non-determinism by separating the ordering phase from the smart contract execution phase, and reversing their order; the nodes order smart contract execution *results*.

A smart contract is generally also constrained to work with *public state*, i.e. data that resides on the blockchain; that can be data already appended to the ledger, or what is included in the invocation transaction as an input argument. This is needed both for auditability and fork-prevention.

Consider, for example, a smart contract that accepts any transaction attempting to transfer tokens to it, only if the temperature in Raleigh, NC is above 40°Fahrenheit, according to `weather.com`, for the point in time that the transaction references. The external variable here is the temperature lookup. The temperature on 1/29/2015 at 15:00 EST was indeed above 40°F. A transaction referencing that date attempts to send 1 token to the contract. After executing this transaction, every node on the network should update their local copy of the ledger by incrementing the balance of the smart contract by 1 token. If a certain node however is subject to a BJP hijacking attack during transaction execution and the corresponding temperature lookup with the `weather.com` API, they may get a temperature reading that is lower than 40°F. That incorrect reading will result in that node's ledger being forked; clients interacting with that node will be operating on incorrect data. Similar concerns apply to an auditor, or a node that wishes to catch up with the rest of the network. In both of these cases, they *should* be able to verify that the copy of the ledger they are given is legitimate, simply by replaying all the transactions in it. If a transaction were to reference an external variable, the value of that

variable could be tampered during replay time, and deceive the auditor or the onboarding node.

A properly written smart contract can be thought of as a replacement of a trusted third party (TTP), computationally constrained with public state [157]. Counterparties to a transaction can inspect its source code, be certain that it will perform its function honestly and accurately, and use its services to solve coordination problems in a predictable, certain manner. As an example of that, consider a smart contract that holds tokens in escrow and redistributes them according to a set of terms agreed upon by a group of participants. These terms are encoded in the contract by the contract's author, and can be verified without any calls to external variables. If the terms are not met, a function in the contract allows the original depositors to withdraw their tokens. The presence this smart contract solves the complex coordination problem that its users would have otherwise. The users do not have to deal with multi-party atomic exchanges, or blindly trust a TTP to help with the redistribution of tokens. Smart contracts ensure that logic is accurately executed across untrusted entries [150], and give us processes of verifiable integrity.

2.2.5 Taxonomy

We identify three main axes for classifying blockchain networks.

1. *Who can participate in the network?* A network that allows only whitelisted participants is called permissioned or private. In contrast, a network that anybody can join is called permissionless or public. There are distinct characteristics caveats associated with each of these types.

    - A public network is susceptible to Sybil attacks, and as such, it looks to proof-of-work (or stake) for reaching consensus — see Section 2.2.2 for more detail. Since such mechanisms are computationally expensive, the networks incentivize participation by introducing the concept of native cryptocurrency. The nodes that

enforce consensus are eventually benefited by a reward in the form of native cryptocurrency tokens. Bitcoin and Ethereum are the largest public blockchain networks to date. In permissioned networks, the Sybil attack vector is not there, because we fall back to the slower, "analog", real-word process of vetting and whitelisting participants. This replacement of one costly process (of mining blocks) for another (vetting new applicants), allows us to employ less costly BFT protocols to reach consensus, and do without cryptocurrency-backed incentives, presumably because the value derived out of the permissionless network surpasses the cost of vetting new applicants. As we will see in Section 2.4, attention should be paid to the governance mechanisms of permissioned networks; we need to ensure that the vetting of new applicants is not centralized, and that there is a protocol for enforcing eviction of malicious participants.

- When a user or application pushes data to a public blockchain network, they do not control where this data is replicated; any node across the world has access to it. This can be a deal-breaker for applications that are required by regulation to limit where the data processing takes place [65].
- The consensus mechanisms used in public networks affect the consistency of the data that users operate on. As we described in Section 2.2.2, we may get temporary forks because two or more nodes propose a valid block at around the same time. A network partition [21] can also result in a fork; the nodes in the two disconnected sets will extend their blockchains differently. At the root of this problem is the fact that an unknown number of participants can come and go, thus the precise number of participants is not known. This presents public blockchain-based applications with two problems: one, they do not know *when* a fork is in progress, and consequently when they may be operating on "unsafe" data. Two, when the fork eventually gets resolved and the network converges to a single "truth", we end up with a group of users — those exposed on the "wrong" side of the fork — operating on data that

is retroactively labeled as "inacurrate". The workaround for this is to ignore the latest X blocks,where X is a function of the probability of observing a deviation[9], as these are the blocks most susceptible to a fork. This gives, for instance, rise to the oft-used "wait for six blocks to confirm a Bitcoin transaction" advice. The problem with this is that *all* operations of an application interfacing with a public blockchain incur a non-trivial delay by default; the application designer has no control over which operations are allowed to be inconsistent, and by how much. In permissioned blockchains on the other hand, we have the option of working protocols that deliver strong consistency; applications can be built on top of private blockchains with the guarantee that the data they operate on now will not be retroactively labeled as forked by the network.

2. *Among the participants, who can do what?* On the one hand, we see blockchain implementations that do not support fine-grained permission controls at the base level — there is *one* tier of participants, and they are allowed to read ledger, write to the ledger, deploy smart contracts and interact with them, or participate in the consensus process. Permission controls in these cases are implemented at the application level. Public blockchain implementations usually follow this model. On the other had, we see protocols ([11, 104]), with built-in support for different roles and permissions. In our work in [11], we distinguish between nodes that participate in the consensus process ("orderers"), nodes on which smart contracts are deployed and executed ("peers"), and clients who interact with the orderers and peers in the network. Clients can be restricted to simply reading from the network, or writing to it via the enforcement of signature-based policies that are supported at the base level. This model of permissions management is usually supported by private blockchain implementations. There is no right or wrong approach on this. Ultimately, we can have an access control layer (ACL) or governance structure whether the blockchain supports it as a built-in feature or not, because in all blockchain

[9] Itself a function of the network's aggregate hashrate.

implementations, every action is cryptographically signed. As we noted earlier, this is the case for the first category, where ACL can be implemented via the whitelisting of addresses, or the usage of tokens — i.e. you can interact with a smart contract, as long as you have access to the tokens that this contract generated and distributed at an earlier stage.

3. *Are smart contracts supported?* Support for smart contracts does not come in all blockchain implementations. One *might* argue that Bitcoin supports smart contracts because it does use a scripting system for transactions, but that system is intentionally not Turing-complete, and does not support loops [195]. By any practical definition then, the most well-known blockchain network lacks smart contract support. If such functionality is desired (see Section 2.2.4), applications should consider other blockchain implementations. The criticism against smart contracts focuses on transaction throughput. Each block contains a list of transactions. The faster the nodes can process these transactions, the higher the transaction throughput. The simplest way of speeding this process up is by executing transactions concurrently on each node. This is possible on blockchains that lack smart contract support, because they follow a model where every new transaction explicitly identifies the transactions it depends on; going back to the shared database model introduced in Section 2.2.1, transactions create new rows by invalidating existing rows (transactions) in the database, and this dependency relationship is explicitly identified in the incoming transactions. Nodes can then process transactions that do not depend on the same rows concurrently, and only serialize the execution of transactions that touch on the same rows. This model is called Unspect Transaction Output, or UTXO — it was introduced by Bitcoin. With a basic smart contract implementation however, as is the case with Ethereum, the dependencies of a transaction that invokes a smart contract are *not* clear, unless the smart contract function in question is executed. As noted in [105], Bitcoin transactions form a directed acyclic graph, which is only partially ordered, whereas Ethereum transactions *must* be strictly totally ordered. This means that

all nodes in the network process smart contract invocations serially, and this translates to the network's transaction throughput taking a hit. Our work in [11] addresses this problem with a three-step approach:

(a) It allows smart contracts to be installed on *subsets* of nodes. Put differently, this means that every node does not have to have process every smart contract transaction. This is effectively a sharding technique.

(b) As noted previously (see Section 2.2.4), it orders smart contract execution *results*, in contrast to smart contract invocations.

(c) It uses a versioned key-value model for its state, and requires that each transaction identifies the keys that it modifies, and the versions at which these keys were read.

This allows us to process and validate transactions within a block concurrently, and achieve a throughput of more than 4,000 transactions per second (TPS) [11].

2.3 Blockchains and IoT

The following cases can be made for blockchains in the IoT sector:

1. Long-term sustainability/serviceability of IoT devices. This is an argument originally raised in [34]. Consider an IoT device for which a firmware update has been issued by the manufacturer. The manufacturer is obligated to keep this update available in its servers for years to come. As the number of IoT devices being put into the market increases, this distribution of software updates incurs a non-trivial cost to the manufacturer. Additionally, consider the case where the manufacturer shuts down; any IoT device that has not been updated to the latest firmware has now forever lost its opportunity to do so. Such issues can be solved with an architecture that operates transparently via a peer-to-peer model and distributes data securely. Consider the following setup, as an example of how this could work. All IoT devices of a manufacturer communicate with

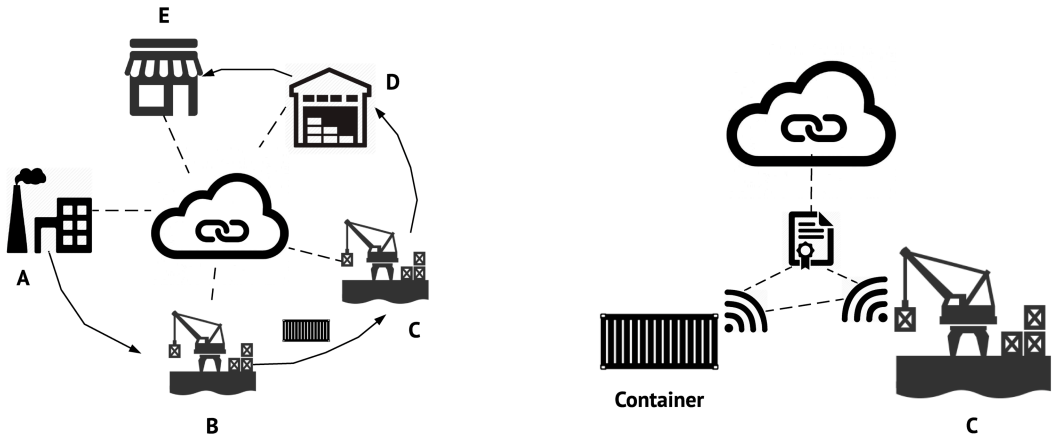


Figure 2.2 An asset tracking example using smart contracts and IoT. On the left, a container leaves the manufacturing plant (A), gets transported to the closest port (B), is then shipped to the destination port (C), where it gets transported to the distributor's facilities (D), until it finally reaches the retailer's site (E). On the right, we focus on the B-C stage. The container carrier performs a digital "handshake" with the dock at the destination port (C) to confirm that the container is delivered to the expected location. Once that handshake is completed, it posts to a smart contract to sign the delivery. The destination port follows along to confirm reception. If the node at C does not post to the contract within an acceptable timeframe, the shipping carrier will know and can initiate an investigation right away.

each other via the Internet and form a blockchain network. The manufacturer deploys a smart contract — in it, it stores the hash of the latest available network. The devices are shipped with that contract's address baked into their blockchain client, or find out about it via a discovery service mechanism (see Section 2.4). They can then query the contract, find out about the new firmware, and request it by its hash via a distributed peer-to-peer filesystem such as IPFS [25]. The first requests for this file will be served by the manufacturer's own node (also taking part in the network). However, after the binary has propagated to enough nodes [24], the manufacter's node can stop serving it. Assuming the devices are configured so as to share the binary they got[10], a device that joins the network long after the manufacturer has stopped participating in it, can still retrieve the sought-after firmware update and be cryptographically assured that it is the right file. All of this coordination happens automatically, without user intervention. In the classic centralized scenario, if the manufacturer shuts down or simply stops serving

the update, the device will get a `404` error code [91] when requesting the firmware update.

2. A backend for a marketplace of services between devices. This is the case for cryptocurrency-driven blockchain networks. Going back to the previous example, devices that store a copy of the binary may charge for serving it, in order to either sustain their infrastructure costs, or profit. Other examples along the same lines include Filecoin [180] which allows devices on the Internet to "rent their disk space", or `21.co` [103, 154] which monetizes API calls; the caller has to provide a micropayment in bitcoin before issuing an API request. Blockchains provide a common, easily-accessible *billing layer*. They allow devices to expose their resources and get compensated for them via microtransactions. This marketplace can extended to transactions between devices and users. As an example, `slock.it`'s [69] electronic locks can be unlocked with a device carrying the "right" token. These tokens are bought on the Ethereum blockchain. The owner of a smart lock can rent their house or car by setting a price for timed access to that lock. An interested party can use a mobile app to identify the lock, pay the requested amount in Ethers, then communicate with the lock via a properly signed message[11] to unlock it. In the energy sector, the integration of IoT with blockchains allows for peer-to-peer marketplaces where machines buy and sell energy with each other automatically, according to user-defined criteria. This is exactly what LO3 Energy is experimenting with in a neighborhood in Brooklyn, NY; solar panels record their excess output on the blockchain, and sell it to neighboring parties via smart contracts [190].

3. Improved tracking and visibility in supply chains. A typical supply chain flow looks somewhat like Fig. 2.2: a pack of containers leaves the manufacturering plant (point A), gets transported to the closest port (point B), is then shipped to the destination port (point C), where it gets transported to the distributor's facilities (point D), until it

[10] For example, via a tool such as `ipfs-cluster` [23].
[11] Using a peer-to-peer communication protocol for distributed apps, like Whisper [227].

finally reaches the retailer's site (point E). This process involves several stakeholders and checks. Each stakeholder usually maintains their own database to track the asset, and updates that database based on inputs from the other parties along the supply chain. In a blockchain-based setup for the same scenario, there is one shared database to monitor across all stakeholders, ledger updates are cryptographically verifiable, get propagated along the network automatically, and build up an auditable trail of activity. Then, when the shipping carrier reaches the destination port for example, they can send a signed message to a predetermined and agreed-upon smart contract to allow everyone on the chain to know that the container is now at point C (Fig. 2.2). Since the transaction is signed, it constitutes a cryptographically verifiable claim from the shipping company that the container has reached the destination port. The receiver at the port posts to same smart contract to confirm that it is in posession of the container. The same process can be implemented via atomic token exchanges [107] on blockchains that lack support for smart contracts and follow the Bitcoin transactional model (Section 2.2.5). We propose the following flow, though alternatives can also work. For every container or set of container that flow through the supply chain, the participants collectively approve of the issuance of two types of token:

(a) The container-carrier token.

(b) The container-receiver token.

The first is issued to the manufacturer's public key. Tokens of the second kind are issued to the remaining stakeholders. When the manufacturer hands off the container to the freight transportation company that will move it from point A to point B, it creates a transaction that does the following:

(a) Assigns the container-carrier token to the transportation company's public key, effectively passing ownership to it.

(b) Assigns the transportation company's container-receiver token to its own public key

Going back to the shared database "row" model of Section 2.2.1, this amounts to deleting two rows (the one where the container-carrier token belongs to the manufacturer, and the one where the container-receiver token belongs to the transportation company), and creating two new ones (where the tokens are swapped). This transaction needs two signatures in order to be valid; one from the manufacturer for token transfer #3a, and one from the transportation company for token transfer #3b. The manufacturer signs their part, then sends this incomplete transaction to the transporter who signs their own part, then pushes it to the blockchain. When this transaction is added to blockchain, the manufacturer has received the "I have received the container" token from the transportation company, and the transportation company now holds the "I have the container token." A similar atomic exchange will take place at point B between the transporter and the shipping company, until the retailer finally receives the container-carrier at point E. At this point there is a complete, cryptographically verifiable, timestamped trail that tracks the asset. The process, as described above, constitutes an upgrade over existing practices. The integration of IoT improves it further as follows. Assume that every stakeholder owns a smart tracker with:

(a) A Bluetooth Low Energy (BLE) radio for connectivity with neighboring devices

(b) A GSM or LTE radio for Internet connectivity

(c) A blockchain client

Containers are also equipped with identical trackers. When two stakeholders interact at any of the meeting points A to E, *and* the containers are also present, the stakeholder trackers can send signed transactions to the blockchain automatically without any user input. The process automatically moves to the next stage as soon as the required tokens have been exchanged. In our setup the BLE radio is needed so that the devices can

identify when they are in proximity of each other, and when that happens they can transact on the blockchain via the Internet. This is just one possible configuration out of many. Filament [79], for instance, manufactures sensors with long-range radios called "Taps". Taps form mesh networks via the telehash protocol [125], and interact with each other using smart contracts on a shared blockchain. The sensors themselves do not connect to the Internet to cut down on costs, but can connect to gateway nodes that provide Internet connectivity.

2.4 Deployment Considerations

Our own experience from building proofs-of-concept for the past 2 years has allowed us to compile a list of issues that designers of blockchain-based IoT applications should keep in mind. Where applicable, we note possible workarounds, or highlight the ongoing work being done to address these issues.

1. Performance. Compared to a properly configured centralized database, a blockchain will be slower, both in terms of transaction processing throughput and latency. This performance difference is particularly pronounced in (i) public networks using proof-of-work for consensus (see Section 2.2.2), and (ii) blockchain stacks that support smart contracts because of the concurrency issues noted in Section 2.2.5. Proposals, such as the Bitcoin-NG protocol [83], allow for better performance. In general however, a performance hit compared to good centralized solutions is to be expected; this is the "penalty" paid for reduced trust and increased resiliency.

2. Privacy, for the identity of the participants and/or the contents of the transactions. In a blockchain network, each participant/device is identified by their public key[12], and participants only need to know the public keys of transacting counterparties. All transactions however are posted for all the participants to see; a shared log of the network's activity is at the backbone of the blockchain concept. By analyzing this data,

attackers can identify patterns between addresses, and make informed inferences about the real-world identities of participants [159, 189]. The issue can be mitigated by:

(a) Having all participating devices use new, unique keys per transactions. Schemes, such as the one proposed in [230] on hierarchical deterministic wallets for Bitcoin, allow the derivation of an infinite number of public keys in a manageable and safe manner. The problem with this approach is that new keys need to be communicated to the counterparties of the transaction — a potentially cumbersome and time-consuming process.

(b) Creating separate networks per application instance. Going back to the asset tracking example in Section 2.3, that would mean a network that is used *only* for a specific set of containers and their transportation from point A to point E; beyond that point the network can be archived and/or disbanded. This technique applies to private blockchains, where a participant of the consortium may get a competitive advantage by tracking another device's activity.

[220] provides an additional look into methods of blockchain analysis, and into ways to prevent them from succeeding. For transactional privacy, i.e. obfuscation of a transaction's contents, the safest approach is to communicate the actual contents of a transaction only to the involved counterparties via a side-channel, and publicize only the hash of that transaction to the wider network. This is the concept behind private data collections and SideDB [177], which was added as a feature to version 1.2 of our work in [11]. Zero-knowledge proofs [140], a cryptographic primitive that allows one party to prove to another the validity of a statement without revealing its content, are a promising solution with limited applicability at the moment; zero-knowledge techniques are resource-intensive and on an exploratory stage, while IoT devices are resource-constrained.

3. Censorship. Cryptographic signatures attached to each message on a blockchain network

[12] Or its hash.

ensure that no node can fake another node's transaction. However, a node that is tasked with "cutting" a new block (see Section 2.2.2), *can* choose to ignore another node's submitted transaction on purpose. Time sensitivities aside, in the long run this should be OK, because this form of censorship is only temporary; assuming a properly configured protocol that periodically rotates the leader[13], the censored transaction will eventually be included in a block by a non-malicious node. However, every BFT consensus protocol has a certain threshold for malicious actors; PBFT, for instance, can guarantee correctness when less than one third of the participants are acting in a Byzantine manner concurrently. For Bitcoin, that threshold increases to 49% (i.e. we just need the majority to be honest). Under the presence of a properly-coordinated mining pool ("selfish mining pool"), that threshold can drop down to less than one third for Bitcoin as well [82]. When that happens the risk of permanent censorship increases. When building a blockchain-based IoT application, ensure that the set of nodes enforcing consensus are chosen so that the chances of collusion among them are minimized. In private networks in particular, legal contracts should be put in place so that detected collusions are penalized appropriately.

4. Legal enforceability of smart contracts. This applies to smart contracts that dictate that if their state is modified according to a certain manner (for example, sensor at location B sends a signed transaction to the contract attesting to the reception of a container), there are "real-world" consequences (the receiving entity at location B is now liable for anything that happens to this container). Using smart contracts to track and manage processes in the physical world is a private network use case; participants should be whitelisted and their real-world identities should be known. The problem occurs when an affected entity arbitrarily disputes the outcome of a smart contract, despite its verifiability. The goal then is to establish a mapping between a smart contract and the corresponding legal contract in the real world. With dual integration [74], one deploys a smart contract, records its address, and includes that address in the corresponding, real-world legal

[13]Do note that this is not the default in PBFT[52].

contract. They then hash the legal contract, record its hash digest, store the digitized version of the contract in a safe space, and then send the legal contract's hash to the smart contract. Another way to mitigate this problem is to use smart contracts derived from templates that are built specifically for this purpose. The Accord Project [5] is an industry-led specification for smart legal contracts; Cicero [62] is an open-source implementation of that specification. It allows for the creation of templates for executable natural-language legal contracts and clauses.

5. Expected value of tokenized assets. Blockchains are used to trade these tokens because they are associated with some value. If that value is expected to be redeemable in the physical world, participants need to examine beforehand who stands behind the exchanged assets, and assess the assurances they have about their value. This is tangential to the problem of legal enforceability for smart contracts (see item 2 in this list). For tokens that are generated and managed by a smart contract then, the solution is identical; dual integration, or a smart contract that is generated by a template that follows an industry standard. For blockchains that do not support smart contracts however, the closest equivalent is hashing a real-world legal contract, and embedding this hash as metadata on the traded token. This is essentially notarization by hash [178].

6. Bugs in smart contracts. Smart contracts are programs, and as their complexity grows, so does the probability of introducing bugs into their logic. A good example of that is the hack of "The DAO" [75]. The DAO, whose name stands for decentralized autonomous organization, was an attempt to create a crowdfunded venture capital fund using a smart contract on the Ethereum blockchain. Participants could buy into the fund by buying tokens that the smart contract had generated, and could participate in deciding which Ethereum-based projects The DAO would subsequently fund, using the governance mechanisms that the contract had introduced in its code. In June 2016 however, an attacker exploited a bug in the contract's code and moved out of it 3.6M Ethers, valued

at $50M USD at the time of the attack. Besides the obvious recommendation of vetting the contract's code carefully, we propose the following:

(a) Keeping the smart contract logic as simple as possible. In general only the parts of the business logic that need to be captured in a blockchain belong to the smart contract; everything else should be computed off-chain.

(b) Introducing fail-safe mechanisms such as time-locked vaults for token transfers and administrative functions. A smart contract can be written so that its behavior changes based on user input; this gives us the ability to create administrative functions that effectively undo bad actions (e.g. a malicious withdrawal of tokens), by applying the opposite action ("cancel-withdrawal"). These can be invoked by a priviledged set of users, using rules that are clearly outlined in the contract's logic and therefore known in advance to all of its participants. For example, in the case of a smart contract that acts as a deposit box for tokens, we would need to ensure that every attempt to withdraw, places the tokens temporarily in a joint escrow account before they can be taken by the withdrawer. When the funds are in the escrow account, M out N whitelisted keys can sign a "cancel-withdraw" invocation, and cancel a malicious attempt to move tokens out of the contract illegally, in case the original "withdraw" function had a flaw in its implementation.

7. Secure off-band communication and file exchange. As we noted earlier, transactions in the blockchain can be read by every network participant — this is by design. Whenever a private communication channel is needed, protocols such as telehash [125] or Whisper [227] should be used instead. The network's file-sharing needs may be addressed by a content-addressed peer-to-peer file system such as IPFS [25].

2.5 Conclusions

Our findings suggest that:

- There are use cases (Section 2.3) in which a blockchain fits in the IoT application stack. These all revolve around the technology's ability to create digital scarcity for assets in a provable way.
- The blockchain layer is not one-size-fits-all. Its design (Section 2.4) dictates the performance, privacy guarantees, and resilience against smart contract bugs that an IoT application built on top of it can offer to its users.

The technology is nascent. Based on the progress and interest in the space we have witnessed during the past 24 months, we expect substantial improvements in both the protocol and the tooling level; these should accelerate the emergence of more blockchain-based IoT applications.

Chapter 3

A Framework for Designing and Evaluating Realistic Blockchain-Based Local Energy Markets

An extended version of this chapter has been submitted to Applied Energy [17]. Dimitrios Sikeridis, from the Department of Electrical & Computer Engineering, University of New Mexico, and Yun Wang, from the Department of Electrical & Computer Engineering, North Carolina State University, have contributed to this work. The work in this chapter has been partially supported by the US National Science Foundation under the New Mexico SMART Grid Center - EPSCoR cooperative agreement Grant OIA-1757207.

3.1 Introduction

3.1.1 Motivation

In Section 1.1.4, we made the case for blockchain-based markets at the distribution edge. We are not the first ones to suggest that such markets (LEMs) (Fig. 1.2) *may* work; in Section 3.2 we analyze all relevant work in the space. The problem however with all pieces of existing

work in the blockchain-based local energy market space is that they focus mostly on the local energy market part, treating the blockchain component as, more or less, a *black box*.

When setting up a blockchain-based system, a number of questions have to be considered and answered. To name a few:

1. Who runs blockchain nodes, and how are they deployed?
2. Who has read/write-access to the blockchain?
3. What consensus mechanism makes sense? Do all nodes run it?
4. What is the frequency with which blocks are cut? Is the block-cutting frequency configurable? If so, what is its value?
5. Who deploys the smart contracts that expose market primitives, such as "buy" and "sell" orders, and who is allowed to invoke a given smart contract?
6. How is the smart contract designed?
7. What, if any, are the conditions under which the proposed design is subject to tampering by byzantine actors in the market?
8. In case of auctions, what dictates the cut-off time for bids?
9. In case of closed-order book auctions, how are the encrypted bids processed for the market-clearing price to be calculated? How is this price communicated to the market participants?

Depending on the blockchain stack that is used, not all of these questions may apply directly, but the general observation stands: none of the answers to these questions are given, and it is the answers to these questions that dictate whether the proposal makes sense as a blockchain-based application, how that application performs in terms of throughput, and how this affects the market that is provisioned on top of it.

Put differently, blockchains reify protocols, and the configuration options offered draw a pretty wide design space. Questions along the lines stated above *must* be answered, and the answers should also be considered for second-order effects; we identify this as a gap in all relevant work (see Section 3.2.1).

3.1.2 Contribution

In this chapter, we design a blockchain-based local energy market by carefully considering the questions above, and reasoning over their implications. Specifically:

- We build this market on top of the open-source blockchain platform that we developed and presented in [11].
- We discuss the design of the smart contact that backs the market operations; we identify who deploys the contract, who can invoke it, what its endorsement policy is, what is the considered data model and how it affects performance.
- Given the above, we consider a double auction with closed-order book, and analyze the options available for implementing this mechanism on a blockchain — something that is noticeably absent in the relevant literature.
- We explicitly demonstrate *how* these design choices matter, by creating three distinct configurations for our local-energy market. Each configuration occupies a different point in the design space: we explain the implications in terms of performance, governance, and degree of decentralization.
- We validate this by performing a case study on the considered configurations, and perform a *joint analysis* on the performance of both the blockchain and the market layers.

Note that the work in this chapter is not meant to be the be-all, end-all work in the space. In addition, we do not perform a sensitivity analysis across *all* parameters exposed to us by

the underlying framework, nor are we presenting the preferred configuration (Configuration 2 covered in Section 3.3) as the golden standard for blockchain-based LEMs.

Our hope is that the work presented here serves as a guiding framework for designing and evaluating realistic blockchain-based local energy markets. Thus, we have explicitly chosen to focus on how specific blockchain implementation options impact the performance, governance, and degree of decentralization of a next-generation LEM. To that effect, all of the code we have developed and used for this work, including the simulation framework and the local energy market configurations, has been open-sourced [57–61].

Structure This chapter is structured as follows: We cover related work in Section 3.2. In Section 3.3 we design the blockchain-based LEM. We define three distinct configurations for the proposed local energy market, run a case study on them, and and compare and analyze the results in Section 3.4. In Section 3.5 we note our concluding observations.

3.2 Related Work

3.2.1 Literature Review

The authors in [161] present a preliminary economic evaluation of a blockchain-based local energy market. They consider 100 residences with solar panels. The market participants can sell energy to each other at random prices between the feed-in tariff and the grid electricity price. Excess supply and demand can be balanced by the grid. The agents adopt a zero-intelligence bidding strategy, and trade once every 15 minutes. The market mechanism is a closed double auction, settled every 15 minutes, at a uniform clearing price. The simulation covers one year. It is performed on a private Ethereum blockchain network. As the authors themselves note, this implementation runs a PoW-based consensus mechanism, which is unnecessary for this application.

In addition, there is no technical evaluation of the blockchain as an ICT (information and

communications technology) platform for local energy markets. In [160], the same authors describe the simulations that were conducted during the project planning phase of LAMP (Landau Microgrid Project), a real-world deployment of a blockchain-based local energy market in Landau, Germany. The market runs on a time-discrete double auction mechanism, and consists of 20 households that produce energy via solar panels. Participants can place an order once every 15 minutes, and use a smartphone app to set maximum (minimum) prices for their bids (asks).

For the simulations, the authors assume perfect forecast for next slot's generated energy, and model the participants as agents that learn from their past trades and adjust their bidding strategies accordingly. They find that the marketplace increases self-consumption by 16%. However, the details of the blockchain implementation behind this experiment were not shared, and thus the influence of the blockchain setup on the simulation results cannot be assessed.

In [141] a blockchain-based solar energy production and distribution network is presented. The service area the authors focus on is the local area power distribution system. Smart meters measure the energy prosumers inject into the system and reward them with tokens. Prosumers can then transfer these tokens to each other in exchange for energy. However, there seem to be unnecessary centralization points such as the proposed *single* middleware controller instance for all agents involved in the system, and the single smart contract owned by the controller (Section V).

In addition, the coins remain at the contract owner's address and are considered spent, when talking about transfers (Section V), which contradicts the semantics of a transfer. Finally, there is a single controller node in this network, on which the contract is to be deployed; that node is the only one that can transfer coins from one user to another. 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. If we are guaranteed that all the nodes will interact properly with the controller — as the paper effectively suggests —, we should be using a centralized

database instead.

The work in [144] presents "PETra", a blockchain-based transaction management platform for transactive energy systems (TES). The stated goals of this platform are to protect the privacy of its participants, and the safety of the system by ensuring it operates within stable bounds. However the proposed solution lacks in two fronts:

1. On the privacy front, the authors merely propose the standard and widely-adopted practice in blockchain platforms that has participants using a new private/public key-pair for every transaction.

2. On the system safety front, the underlying concern is valid only in the absence of a proper market mechanism, i.e. one that would collateralize the sale, for the low supply case. For the oversupply case, slack buses are there to absord to extra load.

The author in [127] design a blockchain-based system that allows electric vehicles (EVs) to exchange energy with each other. They utilize a consortium blockchain, where the validating nodes are the local aggregators that run a double auction mechanism for clearing the market. 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 [56] for more on this. In addition, the use of an auctioneer defies the claim that this trading happens "without reliance on a trusted third party" — since the auctioneer is exactly that —, while the same functionality can be easily facilitated by a smart contract. Finally, similar to other works there are no specifics regarding the blockchain platform's configuration. Therefore, it is unclear how the blockchain's configuration affects the experiment and the results; this seems like yet another application for a centralized database.

The work in [139] presents a blockchain-based protocol that allows electric vehicle owners to choose a tariff dynamically via Pedersen commitments [172] 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. That commitment is ultimately redeemed when the EV reaches the station for charging. 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 this work. In addition, under no reasonable market assumption, will *multiple* charging station owners *reserve exclusively* a charging slot for a customer that may ultimately not show up. Thus, the applicability of the presented solution is questionable. Section 4.2 in their paper is an admission of this impracticality.

A local energy marketplace designed as part of "NOBEL", a European project, is presented and simulated in [122]. 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. Unlike what we saw in [161], there is no single clearing price per slot. For the evaluation scenario, the authors use "zero intelligence" agents because their behavior can help reach a high level of order matching, therefore the equilibria of the market can be found, and we can try to understand what the outcome of the market model is. This was also the case in [161].

The market is comprised of 50 producers and 50 consumers. The agents choose order prices uniformly from a common, arbitrarily chosen price range, and predict their production/consumption with perfect accuracy. Time is divided into fixed 15-min time slots because this is the typical smart metering measurement frequency. The bidding for each slot closes 2 hours before that slot is about to become active; bidding is open for 2 days.

In [94], the authors test out four variations of a smart contract, all implementing a uniform-price, double-auction mechanism for a decentralized energy market. Their setup consists of a private Ethereum network made up of 306 nodes, one for each of the 240 houses, 60 solar panels, 3 wind turbines, and 3 miners. Their simulations run for 4 days (February 1st to 3rd, 2018 in Seattle, WA) in real-time and use the IEEE-13 bus test feeder ("a short and relavitely highly loaded feeder that contains a substation voltage regulator, overhead and underground lines, with variety of phasing, shunt capacitor banks, in-line transformer, and unbalanced spot"). The market clears every 15 minutes.

Each smart contract is testing a different way to calculate the market-clearing price (covering

the spectrum from off-chain to on-chain calculations) and 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. Given that this is a private Ethereum network it is unclear why so much gravity is given to gas costs; there is even a breakdown on monthly gas cost for bidders. This is an externality that ultimately should *not* matter in private networks — all of the interactions happen with one well-vetted contract, and consumers should be able to "refill" their gas balance as needed.

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 [94] would presumably cover), and we also track how the effectiveness of the *market* (i.e., its allocative efficiency [122]) is affected by the underlying blockchain implementation.

Another work by the same authors, that is closer to our line of work, is [95]. When it comes to experiments, the *main* (but not only) difference compared to [94] is in the sensitivity analysis that is performed during the simulations; the number of market participants (540 to 1,000 residential houses), the auction period (5 to 15 minutes), and the block generation time (1 to 5 to 15 minutes) 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 [94].

Specifically, the authors compare the clearing price and quantity of each market clearing over the time horizon of one day (i.e. 96 market clearings for 15-minute auction periods, and 288 market clearings for 5-minute auction periods) with those same numbers for the centralized scenario, by using the scaled L^2 (L2 or Euclidean) and L^∞ (L-infinity) norm. They visualize the difference in clearing price and quantity in order to track the effectiveness of the distributed systems they have implemented, compared to the corresponding centralized

system. They find that the more stressed the decentralized system is, the worse its market effectiveness.

The same applies for tight clock synchronization — 5-minute auction periods relying on a blockchain with 5-minute block generation periods give bad results. Their natural conclusion is 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 [31] the authors run simulations of a day-ahead and real-time local energy market on a private Ethereum blockchain network with a focus on gas usage. Gas is effectively the cost that users need to pay for a transaction or smart contract operation; the more complex the operation, the higher the gas cost. The underlying thinking in [31] then, is that measuring gas usage gives us a sense of the scalability of the system because any given block has a cap on gas costs, which translates to an upper bound for transactions per block.

However, in a purpose-built private network with vetted participants and a select number of smart contracts, gas is an artifical constraint, and while the authors acknowledge that the gas limit can be adjusted, their concern that "by setting the gas limit too high, the blockchain could suffer from technical disadvantages" is ill-placed. The sensitivity analysis revolves around changing the number of participants (150, 300, 600) and the trading frequency (1, 5, 10, 15 minutes per slot). All the results follow a linear relationship. 600 participants means we need 10 transactions per second in the 5-minute slot scenario, or 5 times that amount for 1-minute slots.

The authors in [36] deploy an agent-based simulation framework to simulate blockchain-backed energy marketplaces. The agents in their framework are economically rational (their bids are bound between the utility's wholesale and retail price), and follow a zero-knowledge strategy (their bids are always randomly set within those bounds). The energy traces come from 200 households in Lille, France, sampled at 1-minute intervals. On the blockchain side

of things, a private blockchain network that runs on Ethermint [53] (i.e., an Ethereum VM and the Tendermint consensus engine [37]) and cuts blocks every 5 seconds is deployed on a 20-GB, 6-vCPU Openstack VM. Every household is represented by a node on that network, and the marketplace runs a double auction.

There are two distinct sensitivity analyses going on during the experiments. In the first one, the authors vary the period under inspection (summer, winter), and the proportion of prosumers in the grid (from 0 to 100% in 25% increments), and track the gains of the market participants and the efficiency of the market, in terms of the local overproduction sold in the market. They find that the marketplace is more efficient, i.e. a higher percentage of locally-produced excess energy is sold, when the offer and demand volumes are bigger. In the second analysis, they vary the number of validating nodes, bids sent per second, and the period between successive bids.

Their findings show that the higher the number of validators used, the longer it takes for a single transaction to get validated; the average validation time per transaction grows from less than 20 seconds on a 3-validator network, to approximately 2 minutes for a network with 20 validators. We note that the two analyses are, unfortunately, decoupled; we cannot see, for instance, how the blockchain parameters affect the market's efficiency in this integrated system.

We also note that the validator-related findings offer no new insight; we know that increasing the number of validators decreases the throughput of the system becauses Tendermint is a rebranded PBFT protocol, and the PBFT algorithm [51] has quadratic communication complexity ($\mathcal{O}(n^2)$ where n is the number of validators in the network). On a final note, it is unclear why setups with, say, 20 validating nodes are examined — in a PBFT network, n is picked such that $n = 3f + 1$ where f is the number of faulty nodes we wish to tolerate. A 20-validator network has *the same* fault tolerance as a 19-validator network (the value of n for $f = 6$), but worse performance because the additional validator needlessly participates in the consensus process.

The work in [169] is a prime example of what motivates our work in this chapter. The authors introduce a game-theoric model for demand-side management, and conduct a case study with residential loads in a microgrid. In it, they introduce a blockchain, but treat is as a black box, without any reference to its configuration, and how this may affect the flow of bids into the system. For instance, they use the blockchain implementation that we have built in [11], but there is no reference to the block periods, or who is running the nodes.

Our work does not consider additional privacy-related measures, other than the standard practice of using a hierarchical deterministic wallet [230] 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' load profile will be decoupled from their presence on premise, so a malicious actor will be unable to detect when the customers are home[1]. Furthermore, as some blockchain-based networks inevitably move to reputation-based markets, pariticipants in the market will actually *want* to have their bids coupled to their identity in return for lower rates or higher quality of service.

For a privacy-friendly approach to local energy trading, we direct the reader to the work of [4]; it proposes a protocol that guarantees the confidentiality of bids and the privacy of the auction winner by using secure, multiparty computations. The work does *not* consider a blockchain implementation, but integration with blockchain-based systems should be feasible. The protocol can handle 2,500 bids in less than five minutes in the on-line phase.

[1] [7] writes that "consumers worry that intelligent monitoring devices, which transmit power-usage information to the utility as frequently as every 15 minutes, would make them vulnerable to thieves, annoying marketers, and police investigations." If we fast forward a few years ahead, when solar panels and battery storage are prevalanet, and where smart electric devices can have their operation time-shifted, what does the fact that a customer draws energy in the middle of night can tell you about their whereabouts? Nothing.

3.2.2 Blockchain Performance Evaluation

[71] introduces a benchmarking framework called "BlockBench", and a set of workloads for evaluating private blockchain systems. Their benchmarks are derived from database workloads and are essentially smart contracts that exercise a certain component of the blockchain system, or the whole stack. They use two macro-benchmarks to assess the performance of the application layer, and micro-benchmarks to evaluate the performance of the three layers found in each blockchain system: consensus, data, and execution.

They compared Ethereum with Parity and Hyperledger Fabric (v0.6). Their evaluation metrics generally are: throughput (successful transactions per second), latency (response time per transaction), scalability (changes in throughput and latency when increasing number of nodes/workloads), and fault tolerance (throughput and latency changes during node failures — crashes, network delays, and random responses). Specifically, they measure:

1. Peak throughput and latency (y-axis) for their two macro-benchmarks (x-axis) and a testbed consisting of 8 servers, 8 concurrent clients and trial lasting for 300 seconds.

2. Average throughput and latency (y-axis) for their two macro-benchmarks when the request rate changes from 8 to 1024 tps.

3. Average queue length (y-axis: number of requests) over time (x-axis) for two request rates (8 and 512 tps).

4. Average troughput and latency (y-axis) by varying the number of clients and servers (x-axis) from 8 to 32 (i.e. 8 on the x-axis means 8 clients and 8 servers), or only the number of servers from 1 to 32.

5. The average throughput (y-axis) when they kill 4 servers 250 seconds in the trial, and they range the number of servers from 12 to 16 (x-axis). The number of clients is fixed to 8. This allows us to assess the performance of the system in the presence of faults.

6. Execution time and peak memory usage (y-axis) by varying the size of the array from 1 MB to 100 MB in the micro benchmark on execution. (The smart contract has to quicksort an array.)

7. Average write throughput, average read throughput, and disk usage (y-axis) when the number of keys/values that are read/written varies from 0.8 to 12.8M. We are dealing with fixed-size keys and values (20 and 100 bytes respectively).

8. Throughput (y-axis) when the "do-nothing" smart contract is executed. This allows us to infer the overhead incurred by the rest of stack sitting on top of consensus.

In our previous work [11], we evalute the performance of our open-source blockchain implementation, Hyperledger Fabric, using a simple cryptocurrency that follows the Bitcoin UTXO model. We begin our experiments by deploying 3 ordering service nodes (the nodes that package transactions into blocks and disseminate them to the network), and peer nodes (the nodes that execute chaincode according to client requests, and maintain the state of the blockchain) in VMs in one datacenter.

The VMs are interconnected with network links rated at 1 Gbps. For all experiments, we use clients that issue concurrent requests to the peers; we increase the number of these clients until the system end-to-end (E2E) throughput is saturated, and focus on the throughput and E2E latency (as observed by the peer nodes) *just below* saturation. Our sensitivity analysis unfolds as follows:

1. Block size: we vary the block size from 0.5 to 4 MB, tracking average throughput and E2E latency, as measured at the peers. We find that the throughput does not significantly improve beyond a block size of 2 MB, but latency gets worse. We therefore adopt a 2 MB block for all subsequent experiments; this gives us an E2E latency of 500ms.

2. Number of vCPUs per node: we vary the number of vCPUs from 4 to 32. The peak observed throughput is approximately 3,500 transactions per second for 32 vCPUs.

However, we observe that scaling beyond 16 vCPUs gives us diminishing returns; non-parallelizable stages in the block validation process (read-write check, ledger access) dominate the latency.

3. Stable storage medium: we switch from SSD to RAM disks for stable storager at all peer VMs, and notice limited benefits — a sustained peak throughput that is 9% improved over the SSD case at approximately 3,900 transactions per seconds (32 vCPUs).

4. Scalability over a local area network: we fix the number of endorsing peer nodes to 10, and increase the number of non-endorsing peers from 10 to 90; all of the peers are in the same datacenter, and all of them connect directly to the 3 ordering service nodes (the nodes that cut the blocks). We observe a negligible drop in the throughput. This goes against our intuition, as we would expect the bandwidth of the 3 ordering service nodes to be consumed by the peer connections. It turns out that while the nominal rate for the network links is 1 Gbps, the actual rate (as measured by `netperf` between pairs of nodes) is 5 - 6x that number.

5. Scalability over two datacenters and impact of gossip: we place the 3 ordering service nodes, 10 endorsing peer nodes, and all clients in the IBM Cloud Tokyo datacenter. We place 20 to 80 non-endorsing peers in the Hong Kong datacenter; all non-endorsing peers maintain direct connecting with the ordering service nodes and the endorsing peers. The throughput between the two datacenters (as measured by `netperf`) is 240 Mbps on average. Our findings show that the throughput (Fabric transactions per seconds) remains the same as in the single-datacenter case for up to 30 peer nodes total. Beyond that, the network connections of the ordering service nodes in the Tokyo datacenter are saturated. For a total of 90 nodes (10/80 endorsing/non-endorsing peers), the throughput falls by about one third compared to the single-datacenter case, to approximately 2200 transactions/second. If we enable gossip mode, and group the 80 peers in the Hong Kong datacenters in groups of 10 each, with only the leader of each group connecting to

the ordering service, the throughput rises to approximately 2800 transactions/second.

6. Scalability over multiple datacenters: we repeat the previous experiment, and keep the setup of the Tokyo datacenter the same, but now spread 100 non-endorsing peers across 5 datacenters (Tokyo, Hong Kong, Melbourne, Sydney, and Oslo), with each datacenter carrying 20 nodes.

In [211] the authors focus on the consensus stage in networks running Hyperledger Fabric with PBFT, and use Stochastic Reward Nets to model it. With the resulting model in hand, they:

1. Compute the mean time to reach consensus for networks of up to 100 peer nodes, and
2. Perform sensitivity analyses that allow them to quantify the effect of message transmission delays, or delays at a certain stage in the consensus process, on the overall time to reach consensus.

3.2.3 Real-world Deployments

In September 2017, the Department of Energy awarded $2.5M to the Pacific Northwest National Lab [86, 155] along with project partners Guardtime, Washington State University, TVA, Siemens, and DoD Homeland Defense and Security Information Analysis Center (HDIAC), so as to "develop block-chain (sic) cybersecurity technology for distributed energy resources at the grid's edge, such as transactive energy exchanges that can be expected to create new markets".

The work in [162] discusses the Brooklyn Microgrid (BMG) project. It is a blockchain-based local energy market, covering a 10-by-10 housing blocks area, and spanning three distribution grid networks. Participating households can buy/sell energy from one another. The market mechanism is a closed-order book with a time-discrete double auction in 15-min time slots. The network can operate in island mode when power outages occur; criticial facilities such as hospitals receive energy at fixed rates, while residences and business have to bid on

the microgrid's remaining power. The system is a private blockchain network that runs on Tendermint.

Grid+ [108, 163] is a utility provider that uses the Ethereum blockchain and a protocol/system built on top of it to sell energy to its customers, or let its customers sell energy to each other. Its customers will own a piece of hardware called "Smart Agent" (now renamed to "Lattice1" [143]). This device can store cryptocurrencies, buy/sell electricity on behalf of the user in real-time, and manage smart loads. Customers buy custom tokens (BOLT tokens) using US dollars, and have them automatically transferred to their Smart Agent. The BOLT token is issued and backed by Grid+ and it is stable; i.e., it will always be reedemable for 1 USD. A stable token is needed to avoid speculation and liquidity slips. Transactions take place on payments channels to avoid the network congestion and comparatively high transaction fees of the main Ethereum network (mainnet). Customers pay for their electricity either in the day-ahead market (DAM), or in real-time (every 15 minutes or 1 hour, depending on the local Independent System Operator, or ISO).

From the description provided[2] it seems that the customer pays *after* consumption in the real-time market (RTM). The company's mission stems from its belief that Ethereum can enable lower cost electricity via disintermediation and user agency. Their revenue will be generated from markup on wholesale energy prices, transaction fees, interest on capital backing BOLT tokens, and sales of Smart Agents. Grid+ expects to charge 20% above the wholesale plus distribution cost. For the Texas market this would provide a customer savings of 38%. The mandatory use of deposits results in the complete removal of bad debt expenses, which creates a competitive advantage over incumbent retailers. The Smart Agent will predict usage for the customer and try and find the most efficient split between the day-ahead and real-time markets. The company is considering adding a firm fixed option.

Swedish utility Vattenfall launched "Powerpeers", a blockchain-based energy trading platform for prosumers. It charges a subscription fees to traders, and plans to expand the

[2]See [163, Fig. 8].

platform to other countries in the region [118]. Their executives see this platform as enabling "the exchange of goods versus money in real time" [119].

Belgium-based energy technology company Ferranti developed a blockchain-based DSO (Distribution System Operator) application, allowing trades on next-day energy positions. They shifted from Ethereum to Fabric, citing the fact that "the platform doesn't depend 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 [30].

A utility in Fukushima, Japan debuted a blockchain-enabled microgrid last year [191] to experiment with energy exchanges between the 1,000 participating households. The participants were also given smart power strips by the utility 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 [147]. Their target is developing countries [117]; 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 directly with energy producers. 700 households have used it to bid on energy. They use Ethereum because they want a permisionless blockchain, but the long transaction times are a paint point ("20 to 30 seconds to tell a customer whether they can buy energy or not" [116]).

Finally, we point the reader to the survey work done in [10]; 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.

3.3 System Model

3.3.1 The Energy Market

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.

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 [144, 161]. A uniform market-clearing price (MCP) is determined for each time slot[3]; the lowest bid price that can still be served given the aggregated supply determines the market clearing price [161]. Fig. 3.1 shows an example of how the MCP is calculated.

Smart Meters

Bids are placed and received by the households via their TE-enabled smart meters, inline with the setup adopted in [144]; 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. 3.2).

For simplicity, we assume that the forecast module has perfect accuracy for the next slot,

[3]We refer the reader to [61], for an open-source library we developed for MCP calculation.

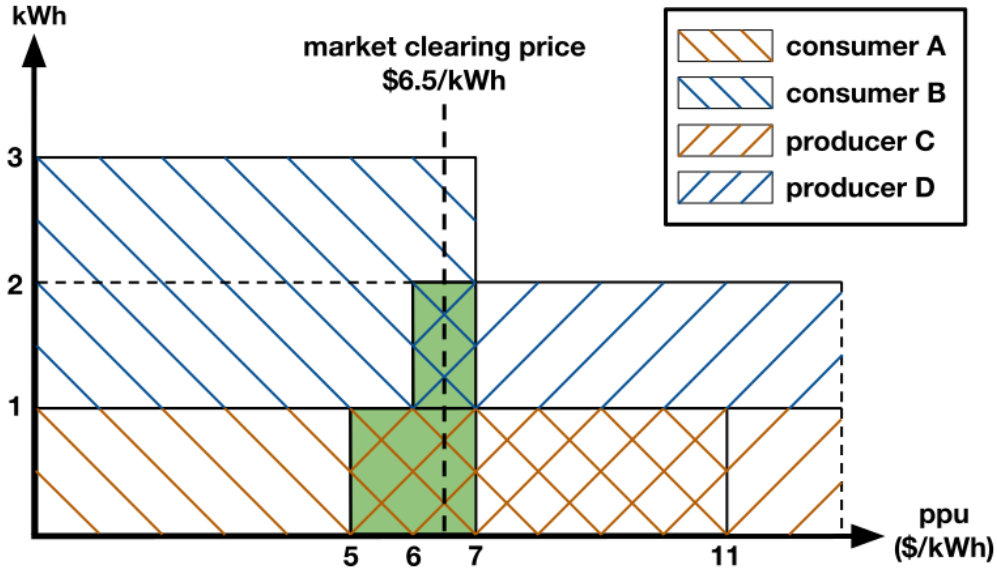


Figure 3.1 An example market clearing price (MCP) calculation in a double auction. Consumers A and B have placed bids for 1 and 2 kWhs with reservation prices of \$11/kWh and \$7/kWh respectively. Producers C and D have placed offers for 1 kWh each, with reservation prices of \$5/kWh and \$6/kWh respectively. We sort the demand and supply according to their reservation prices — highest (lowest) reservation price for bids (offers) goes first. We choose as MCP the price that maximizes the amount of tradeable energy. When more than one price satisfies this condition, 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.

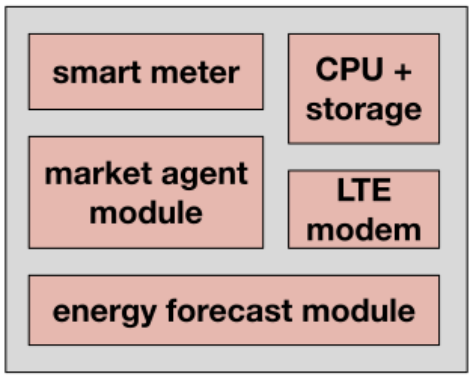


Figure 3.2 Main modules in the transactive energy enabled (TE-enabled) smart meters that all prosumers use in our system model (Section 3.3.1).

as in [160]. 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.

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[4].

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 [88]), 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 [88], 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 [88], 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.

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 [4], i.e., bidding for slot s_i

[4]The maximum (minimum) price at which the buyer (seller) is willing to trade [144].

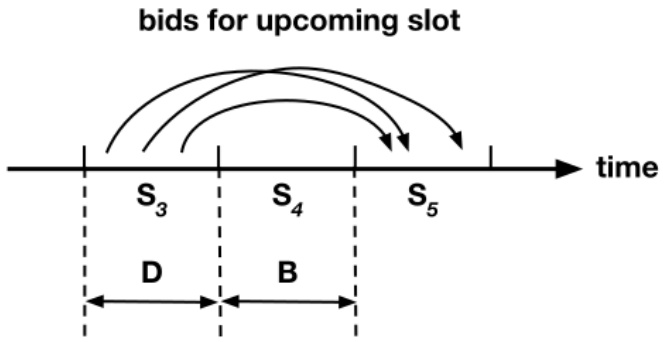


Figure 3.3 In the example shown here, bidding for slot s_5 takes place during slot s_3. In general, bidding for slot s_i takes place during slot s_{i-2} (Section 3.3.1).

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. 3.3.

3.3.2 The Blockchain Layer

Blockchain Stack

The blockchain network in our model is *permissioned*[5]. This happens for two reasons. One, 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 3.1). Two, we want to minimize exposure of the blockchain nodes to the public Internet for security reasons; the power grid can [203] and has been hacked [149], but electricity is a public good — its uninterrupted delivery to customers should not be taken lightly.

A permissioned network configuration means we do not need Proof-of-Work (PoW), as it solves a problem (Sybil attack [73]) we do not have. Consequently, we have no need to incentivize the miners financially, which translates to *no need for cryptocurrency support*. Instead,

[5]As opposed to *public* or *permissionless*; see our taxonomy of blockchains in [56].

we resort to classic Byzantine fault tolerant (BFT) solutions, and can use a mechanism such as BFT-SMART [27] as our consensus engine[6].

We choose Hyperledger Fabric, the system we developed in [11], as the blockchain implementation for the LEM's TMP. Besides supporting the choices above (BFT using the ordering service developed in [28]), it allows us to write smart contracts in a standard, general-purpose programming language, and supports a fine-grained trust model[7]. We leverage both of these features in the text that follows.

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[8] 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

[6]Refer to our previous work in [56] for an exposition into the terms used here.
[7]Via endorsement policies at the application (smart contract) level. This allows us to dictate who can invoke what.
[8]In blockchain terms: *blocks*.

(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 [9][9]. 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 3.4.1, energy transactions rarely happen on scales shorter than five minutes[10], and physical limitations may not allow this length to get any shorter. Such timeframes — which map to auction periods in our LEM — are, for now[11], long enough for the ordering service to rotate through at least a third of its members, thus eliminating the possibility of censorship.

Peer nodes are maintained by all entities in the system (utility, ESCOs, prosumers). When computing resources are scarce, peer nodes are maintained *at a minimum* across the same subset as the ordering service, and ESCOs are created with the sole purpose of acting as the ingress point for the prosumers' client nodes into the blockchain network. Hereinafter we refer to peer nodes maintained by the utility and the ESCOs as *organizational*[12] *peers*, to distinguish them from the *individual* peers that may be run by prosumers.

Table 3.1 shows which Fabric roles a LEM entity (depicted in Fig. 1.2) can assume.

Smart Contract Design

Purpose Market participants[13] use the smart contract to *post orders for the auction*. A closed order book means that *these orders need to be encrypted*. Furthermore, the auction having discrete market closing times means we need a way to communicate these time barriers to

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

[10] "Timescale for energy market services transactions: auctions or agreements for energy exchanges are typically made minutes to day ahead in advance." [204]

[11] The need for ancillary services (regulation, flexibility, ramping) is expected to increase with increased penetration of renewables [204] and as these trends develop, we will be moving towards shorter periods. The Transactive Energy Framework [109] notes that interfaces with existing market structures should be considered within the seconds-to-minutes time band.

[12] Not to be confused with the concept of an organization in Fabric.

[13] In the text that follows, we use the terms "market participants", "agents", or "nodes" interchangeably.

Table 3.1 Association between Fabric roles and market roles in the system model in Section 3.3.2. *can* means that the LEM entity *can* assume that role but is *not required* to; *must* means that this Fabric role *must* be assumed by the entity if it wants to participate in the blockchain-based LEM. Concretely here: the utility can participate in the network with orderer or peer nodes, but cannot operate as client since it is not expected to post orders. ESCOs *will* be assigned with running ordering service nodes, and peers. Optionally they may act as clients if the energy service they provide requires them to bid into the market. Finally, prosumers cannot deploy their own ordering service nodes, can deploy peer nodes if they want to, and must act as Fabric clients since that allows them to bid into the market.

market role	Fabric role		
	orderer	peer	client
utility	can	can	cannot
ESCO	must	must	can
prosumer	cannot	can	must

the participants, effectively coming up with a rough synchronization primitive. All of these considerations need to be taken into account for the smart contract(s) that make the LEM work. We proceed with the following design.

Methods For the auction in slot s_i, each market participant n_j posts in slot s_{i-2} (see Section 3.3.1) an order encrypted with the public key $pk_{j,i}$ that corresponds to private key $vk_{j,i}$. Then the participant n_j follows up by posting their private key $vk_{j,i}$ to the blockchain. They do this, so that their order can be decrypted and a market-clearing price can be calculated for slot s_i. This suggests that every slot is actually broken into two phases (sub-slots): *bid*, followed by *reveal*. These are denoted with subscripts BP and RP respectively. We describe how the agents discern the active slot and active phase within the slot in Section 3.3.2.

On a high-level then, the smart contract[14] exposes the following methods:

1. `buy(encryptedOrder)`[15]. During the bidding phase of a slot, the invoker creates an `order` object defining the amount of energy they are willing to buy, as well as their maximum reservation price per unit (ppu). The invoker then encrypts this order locally using a key pair generated strictly for that particular order, then passes the resulting

[14] In Fabric terms: chaincode.

[15] This is pseudocode, and not the actual method signature; see [60] for our open-source implementation of this.

byte array (the `encryptedOrder`) to the contract's `buy` method. If the order is posted successfully to the blockchain, it returns a transaction ID (`transactionId`).

2. `sell(encryptedOrder)`: Works identically to the `buy` case. Used for selling locally produced energy.

3. `postKey(transactionId, privateKey)`: Invoked during the reveal phase of a slot, so as to reveal the private key (`privateKey`) that decrypts the `encryptedOrder` that corresponds to `transactionId`.

If a smart contract invocation fails — for instance, because of a network partition, or an MVCC conflict (Section 3.3.2) — agents retry a finite number of times. The wait time between retries follows an exponential backoff algorithm, as opposed to retrying immediately, for better flow control.

The only way for participants to know if their order — or more generally, their invocation of a smart contract method —has been submitted successfully: if it has, it will eventually show up in the ledger. When a participant is able to read their write, they know that their transaction has been recorded. This applies to every blockchain network, regardless of underlying implementation.

Endorsement policy The smart contract is deployed on a single channel by the regulated entity that overlooks the network, the utility. The simplest, most reasonable *endorsement policy* for this contract requires the majority of `peers` on the network to sign off on a transaction proposal before it can be considered valid — see "Transaction Flow" in Section 3.3.2.

Time-slotting

As explained in Section 3.3.2, the closed book requirement translates into each slot consisting of two phases; bid (BP), and reveal (RP). The problem we need to solve then is that of *communicating* to the agents the *current* slot and phase.

In Fabric, the ordering service nodes batch incoming transactions and eventually cut them into blocks. Specifically, a block is cut either (a) after a certain amount of time[16] has elapsed after the first transaction of a new batch comes in, or (b) when at least a certain amount of data[17] have been accumulated in the new batch — whichever comes first.

Under perfect network conditions then, we could ensure that a new block is cut *precisely* every `batchTimeout` seconds by issuing a transaction with that exact same frequency ($1/batchTimeout$). Given that the network is unreliable however [21], we are better off issuing transactions at a higher frequency, and this is exactly the approach we adopt. Specifically, we introduce a new `clock` method to the smart contract that is meant to be as lightweight as possible, so as to minimize transmission latency, processing requirements, and disk space effects on the blockchain — it is practically a "no-op" transaction that primes the creation of new blocks.

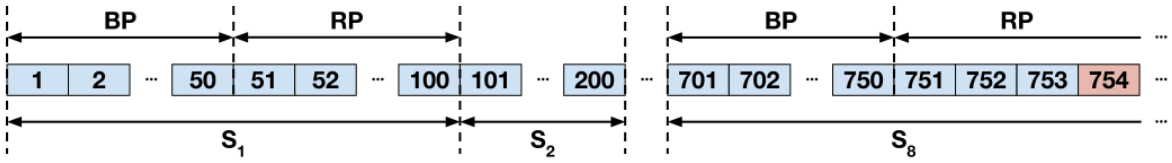


Figure 3.4 Each slot consists of two phases; bid (BP), and reveal (RP) — see Section 3.3.2. In the normal case, each phase consists of a fixed number of blocks. Depicted above, a ledger with 754 blocks. If each phase runs for 50 blocks, this means that blocks 751-800 contain all the reveal activity for the 8th slot.

Any agent can invoke this method at whichever frequency they choose; practically we expect the utility and ESCO nodes to undertake this task, and opt for a frequency that is a single-digit multiple of $1/batchTimeout$. Given a steady, deterministic rate of incoming blocks, every node in the system can then inspect the blockchain ledger *height* to "tell time". For example, if a slot corresponds to 100 blocks, and a phase (subslot) corresponds to 50 blocks, a

[16] In Fabric terms: `batchTimeout`.
[17] In Fabric terms: `batchSize.preferredMaxBytes`.

blockchain that is 754 blocks long means we are now at the reveal phase (RP) of the 8th slot — see Fig. 3.4. 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[18] 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 below.

Resynchronizing blockchain height with market slot The two clocks could be re-synced via a `reset(slotNumber)` method that links the beginning of `slotNumber` with the block that encapsulates this call. This method would be exposed via a separate smart contract on the same channel. We would use a separate smart contract because we would require a different endorsement policy compared to the methods of the existing smart contract. Specifically, the new endorsement policy would require signatures from the majority of the organizational peers present on the network, at a minimum. An application-level (i.e. non-blockchain) process would monitor the wall-clock and the current block height; in absence of new blocks coming in after `batchTimeout + slackThreshold` seconds, it would invoke the `reset(slotNumber)` method on the second smart contract, with the new, appropriate `slotNumber` value according to the what the invoking node's wall-clock reads. The invoking node would then issue a

[18]Remember that in accordance to Section 3.3.1, all energy needs during this timeframe will be satisfied by the grid.

transaction proposal to the organizational peers, and each peer would endorse after checking that the proposed `slotNumber` corresponds to the local wall-clock value, plus or minus a certain slack.

This corrective process is yet again best-effort[19], because we do not know if the local clock of the invoking node is right. Receiving peers may not endorse the proposal because their own clock skews make them think that the proposal is incorrect, or because the proposal does not reach them in time. This is Segal's law[20], and we attempt to get to the truth by applying a stricter endorsement policy, i.e. by polling more peers.

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 3.4.2 — 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[21], 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

[19] Technically this `reset` proposal is also non-deterministic, which makes this an anti-pattern. We rely on the fact that Fabric can handle non-deterministic proposals without issues because it orders their *result* – in this case whether the slot number is reset or not.

[20] "A man with a watch knows what time it is. A man with two watches is never sure."

[21] For a detailed explanation, see [11, Sec. 3].

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.

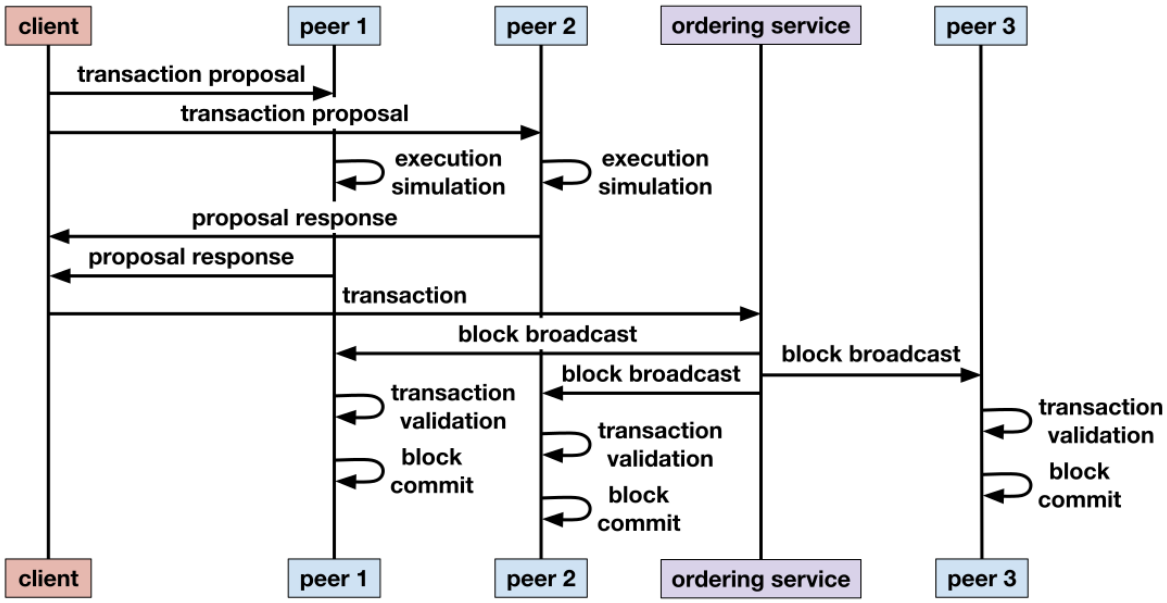
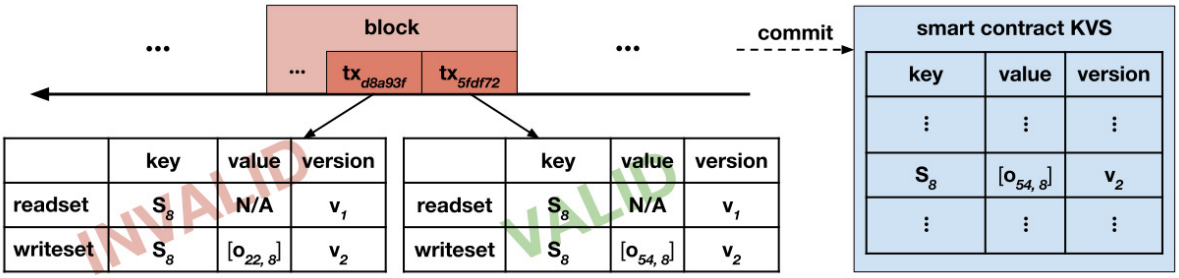
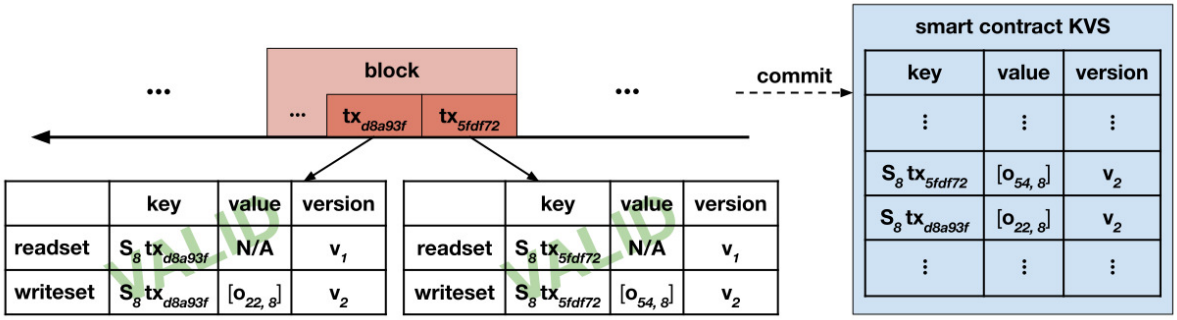


Figure 3.5 High-level overview of the typical sequence followed by a transaction in Fabric. In this example the endorsement policy of the chaincode requires endorsements from peers 1 and 2.

This transaction flow is depicted in Fig. 3.5.



(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.

Figure 3.6 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 3.3.2.

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}[22], 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*[23] *conflict*. This is depicted in Fig. 3.6a.

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. 3.6b.

[22]This is the slot during which participants bid for slot s_i.
[23]Multiversion concurrency control.

Bid Encryption Alternatives

Participants in our model are responsible for encrypting and decrypting their own bids (see Section 3.3.2). 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)`[24] 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 3.3.2) — 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 $N + 1$.

3.4 Case Study

3.4.1 Setup

Market Layer

We use a set of 63 single-family houses with solar panels in Austin, TX. We focus on their electricity generation and consumption during the first two months of 2013. The data is sourced from [66], a repository of residential electricity usage datasets. Our methodology for extracting, transforming, and loading (ETL) the traces in [66] is captured in our open-sourced Jupyter Notebooks [58]. We also open-sourced the utility that allowed us to identify overlapping traces

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

across all candidate houses in [59].

All households in our set participate in the LEM described in Section 3.3 via a zero-intelligence agent [99, 122, 161]. This provides a lower bound on market efficiency [161].

Each slot in the double auction is 15-minutes long [95], and bidding closes 15 minutes before that slot is about to become active, i.e., $D = B = 15m$ (Section 3.3.1).

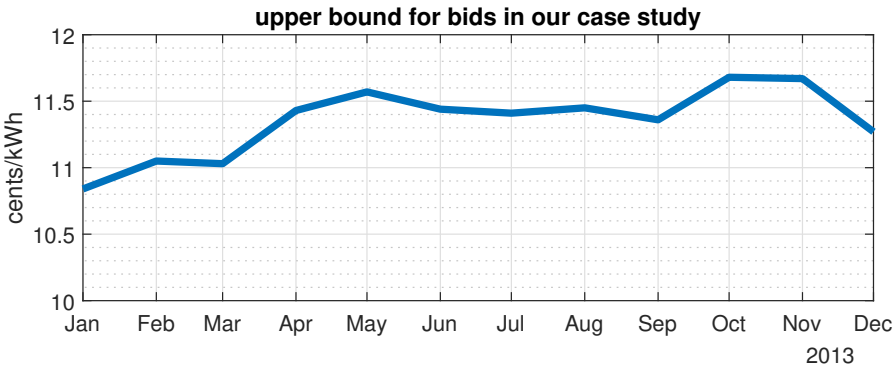


Figure 3.7 Average retail price of electricity (¢/kWh) for residential customers in Austin, TX throughout 2013 [77]. This defines the upper bound for all bids placed in the local energy market presented in Section 3.4.

The *upper bound for bids* is equal to the grid electricity tariff, as is the case with all related works in the literature (Section 3.2.1). We track this via the average retail electricity price for residential customers in Texas [77], shown in Fig. 3.7.

For the *lower bound*, we use the wholesale real-time energy market price from the South zone [1] of ERCOT[25]. We do so because, since they converted to a nodal market for energy in 2010, ERCOT became the singular massive pool through which all energy sales flow [115]. The feed-in tariff [88] — the standard choice for a lower bound — cannot work in Texas. The closest thing to such a rate in the state — the VOST rate [26]— is actually *higher* than the average residential rate (our upper bound). In 2012 for instance, the residential rate for 1 kWh in Texas was approximately 11¢, while the VOST rate was 12.8¢.

[25]There are four competitive load zones [80] in Texas, and Austin belongs in the South zone.

[26]There are no net metering policies in effect in the state of Texas [89]. The closest thing to such a policy, is the

Table 3.2 Market layer parameters for the case study in Section 3.4.

parameter	value
location	Austin, TX
number of households	63
time span	Jan 1st – Mar 1st, 2013
market agent strategy	zero intelligence
slot duration	15 minutes
lower bound of bids	wholesale real-time rate
upper bound of bids	retail rate

All the parameters chosen for the market are summarized in Table 3.2.

Blockchain Layer

Table 3.3 The three blockchain layer variants under consideration in our case study (Section 3.4.1). Under "key per transaction" column, every `buy/sell/postKey` transaction gets posted in a unique key in the smart contract's KVS. In the "key per slot" column, all transactions for a given slot get posted in the same smart contract key. The "common key" row carries configurations where each each bidder encrypts their bids using their own public key; in "individual keys" , each bidder encrypts their bids using a rotating common entity's public key. Configuration 2 constitutes the sensible default. Configurations 1 and 3 are considered for comparison purposes. Only the data slicing and bid encryption columns are shown here, as these contain the decision variables. All other dimensions are set as described in Section 3.3.2.

data slicing	bid encryption	
	individual keys	common key
key per slot	configuration 1	
key per transaction	configuration 2	configuration 3

We consider the following configurations (Table 3.3):

1. In *Configuration 1*, the agents encrypt and decrypt their own bids (Section 3.3.2) but post

Value of Solar Tariff (VOST) program adopted by Austin Energy in 2012. Residental PV customers under VOST are given credit for each kWh they produce, according to the utility's calculated value of solar. This rate may be adjusted annually. It was set to 12.8¢/kWh in 2012 [214]; in 2017, it was set to 10.6¢/kWh [12].

to the same smart contract key for a given slot (Section 3.3.2, "key per slot" option).

2. *Configuration 2* is the one presented in Section 3.3.2 for our system model. It differs from *Configuration 1* in that the agents create a unique smart contract key per slot (Section 3.3.2).

3. In *Configuration 3*, the agents post to unique smart contract keys as in *Configuration 2*, but their bids are encrypted using the public key of a common, rotating auctioneer (Section 3.3.2, "single (fixed or rotating) auctioneer" option).

Simulation

As part of this work, we developed and open-sourced `island` [60], a simulation framework specifically geared for blockchain-based local energy markets.

Table 3.4 Parameters used when running `island` for the case study in Section 3.4.1. N.B. The exponential backoff (see parameter `Alpha`) applies only to Configuration 1. It is used in order to minimize contention between the bidders, since they all attempt to post to the same key in the smart contract.

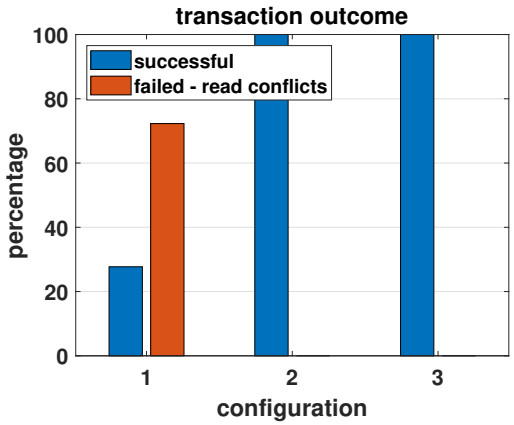
parameter	value(s)	description
`ExpNum`	1–3	blockchain configuration (Section 3.4.1)
`RetryCount`	2	maximum number of retries an agent may repeat a failed smart contract invocation
`Alpha`	10	exponential backoff factor; agents wait for $[0, \alpha \cdot 2^{n-1})$ blocks before attempt n
`TraceLength`	5760	process the first `TraceLength` entries in the trace; each entry corresponds to a 15-minute slot
`BlocksPerSlot`	140	how many blocks comprise a slot
`BlockOffset`	70	the reveal phase starts this many blocks into a slot (Section 3.3.2)
`ClockPeriod`	100 ms	dictates the frequence with which agents invoke the `clock` method (Section 3.3.2)
`BatchTimeout`	200 ms	dictates the frequency with which the ordering service cuts blocks if there is activity on the network, and the file size limits have not been exceeded

We used `island` with the parameter values listed in Table 3.4, against version 1.4.4 of Fabric. We executed the simulations in virtual machines leased at DigitalOcean [70], using their Standard plan (8 GBs of RAM, 4 vCPUs). In order to speed up the execution time, we set `BatchTimeout` to `200 ms` effectively moving us from 15-minute slots (D — see Section 3.4.1) to 28-second slots.

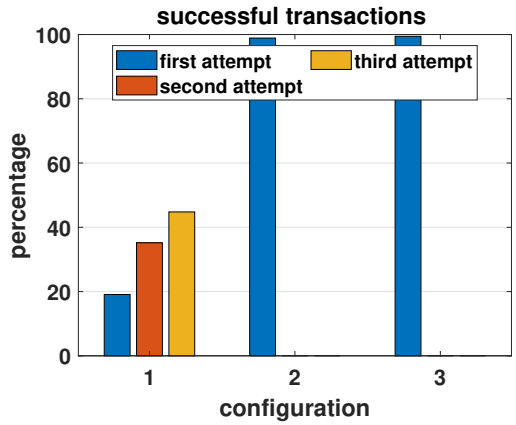
3.4.2 Results

Blockchain Layer Performance Analysis

The first part of our analysis focuses on the performance of the blockchain layer, as a function of the chosen blockchain configuration (Section 3.4.1). The questions we seek to answer are *how many transactions does this TMP let go through*, and *how fast does it do so*, the latter being a metric also explored in [144]. N.B. that we talk about *transactions* that get posted successfully; these are not necessarily successful *bids*, i.e. bids that got matched. Our focus here is on what happens on the transport layer with regards to effective throughput.



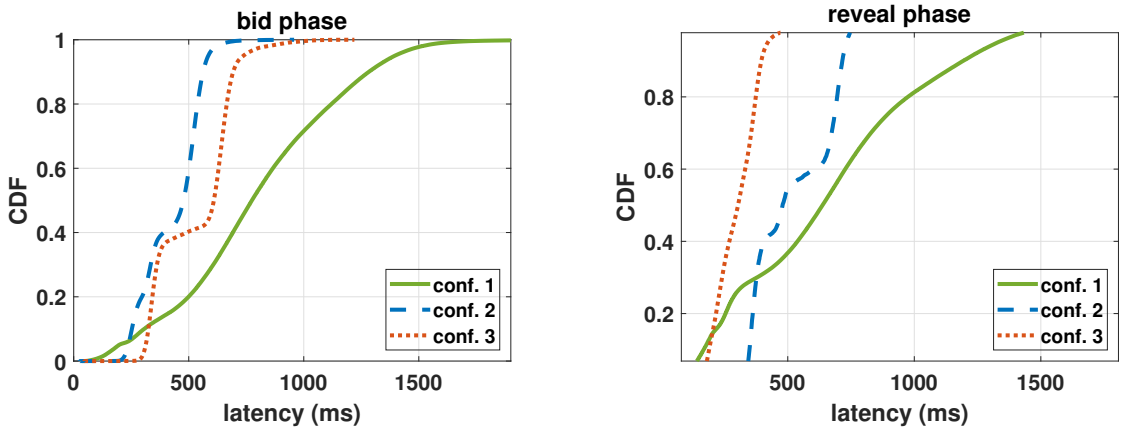
(a) Percentage of transactions that got posted successfully. The success percentage for Configuration 1 is 27.72%.



(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.

Figure 3.8 Effect of blockchain configuration on transaction success rate.

Fig. 3.8a shows that when running under Configuration 1, 72.28% of the time, agents are unable to post a bid (or the decryption key for that bid) to the network in time, and ultimately fail. All failures are due to MVCC conflicts (Section 3.3.2); the readset that these transactions carry is no longer current. In fact, as we see in Fig. 3.8b, out of the successful set, almost half of these successes (43%) register on the very last attempt — recall from Table 3.4 that each agent can retry a transaction up to 2 times, for a total of 3 attempts. Contrast this with Configurations 2 and 3 where all transactions clear (Fig. 3.8a), and do so at the first attempt (Fig. 3.8b).



(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.

Figure 3.9 Latency distribution for successfully posted transactions across all considered configurations in our simulations. Higher and to the left is better. The phases are described in Section 3.3.2; the blockchain configurations in (Section 3.4.1).

In Fig. 3.9, we break down the latency distribution of successfully posted transactions, grouping them by phase (Section 3.3.2). Half of the bids get posted in under 477 ms in Configuration 2, or 609 ms for Configuration 3. When switching to Configuration 1, this number spikes to 775 ms; that is 62% up compared to Configuration 2. The degradation in

quality of service incurred by Configuration 1 becomes more pronounced when looking at the 95th percentiles — its latencies practically doubled compared to Configurations 2 and 3 (Fig. 3.9a).

In the reveal phase, we confirm that Configuration 3 outperforms Configuration 2 — the 95th percentiles are 420 and 731.5 ms respectively. We expect this because the reveal phase consists of a single `markEnd` transaction (Section 3.3.2) in the Configuration 3 case, versus N `postKey` transactions in the Configuration 2 one.

Overall these results validate the problem with the key per slot approach (Section 3.3.2) that is codified in Configuration 1. Under this data model, assuming a common state across peers, we expect only *one* transaction proposal to go through successfully among all concurrent transaction proposals; every other proposal has to be repeated. This is a waste of computing and network resources, and increases the risk that the market participants will not get their offers in on time. Note that this contention occurs twice per slot; both during the bid and the reveal phase. Having the agents default to backing off exponentially even before their first attempt to post a bid mitigates this contention, but not substantially.

Energy Market Impact Analysis

Next, we evaluate the impact of each configuration on the energy market, with an eye towards maximizing the welfare of *the community*. As such, we focus on the comunity as a whole, instead of looking at differences between individual agents within the community.

Fig. 3.10 shows the amount of demand that is met internally, that is, the energy needs that were settled *within* the local market between prosumers. In Fig. 3.11 this is translated into the metric that ultimately matters most from a welfare-maximizing perspective; the aggregate financial obligations ("external cost") of the community towards the grid. As both plots show, Configuration 1 performs the worst; a move away from it results in the amount of energy that is traded internally to more than double (Fig. 3.10). This is accompanied by a 13% drop in external costs (Fig. 3.11).

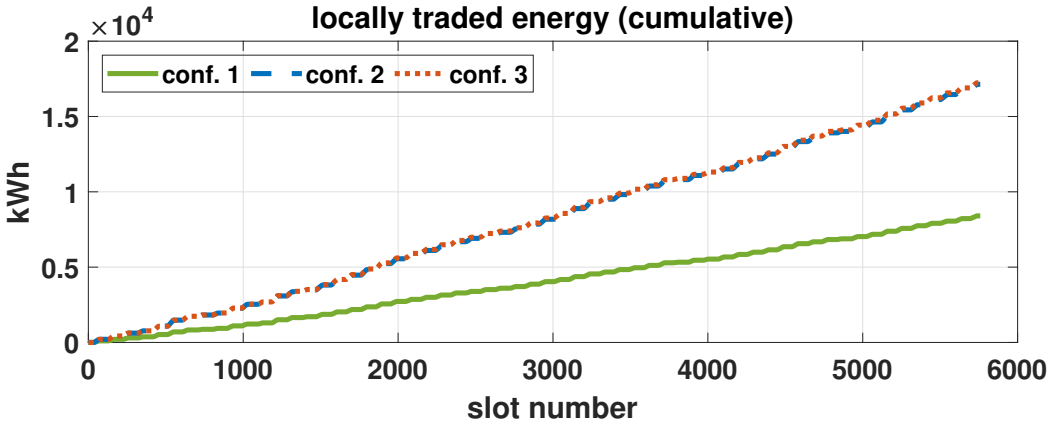


Figure 3.10 Cumulative energy demands met within the local energy market, plotted over time. Any excess demand is satisfied from the grice at a higher price (Section 3.3.1), so the higher the plotted the line, the better. At the end of the simulation run, the community had 8,402 MWh settled internally under Configuration 1, versus 17,140 and 17,240 MWh with Configurations 2 and 3 respectively.

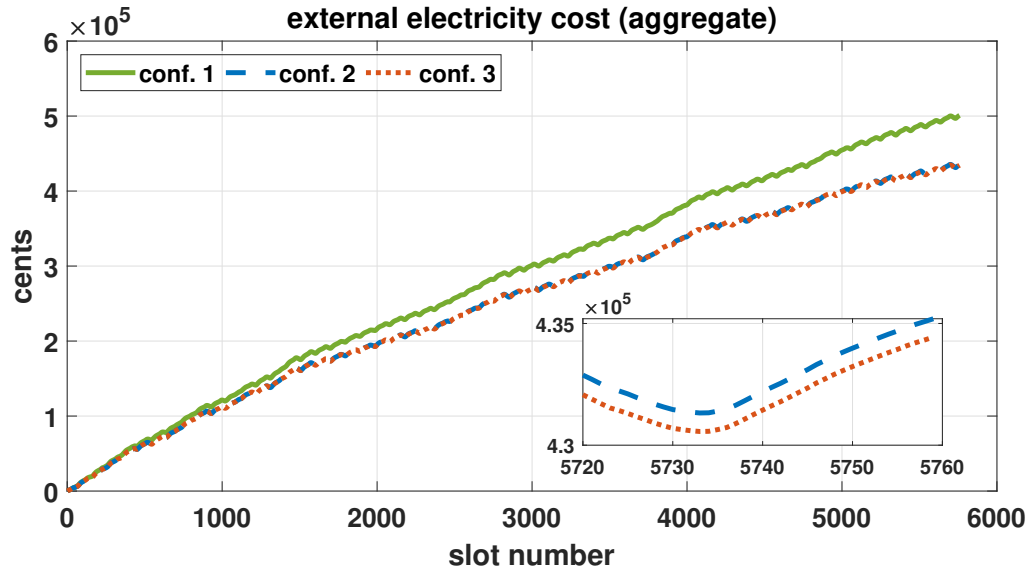


Figure 3.11 Aggregate financial obligation towards the grid service provider for the community. This quantity is the aggregate of the amount paid to the grid from the community at the retail rate for excess demand, minus the amount earned from the grid at the feed-in tariff for excess production. We do not factor in the internal demand since any transactions related to that amounts to money that stays within the community, and as such as it is welfare-maximizing. When running under Configuration 1, the community's obligations towards the grid at the end of the 2-month run amount to $5,009, versus $4,352 for Configuration 2, and $4,344 for Configuration 3. All prices in 2013-USD.

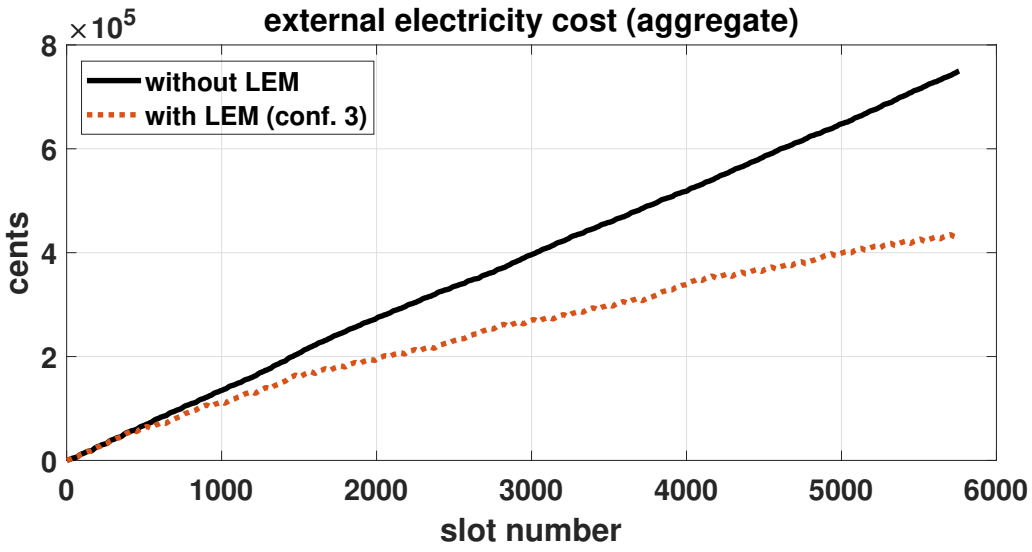


Figure 3.12 Cumulative external cost with and without a local energy market. Lower is better. The local energy market is running Configuration 3, the best-performing configuration from Fig. 3.11. Without a local market, the community pays $7,497 to the grid. With a local market, that number drops by 42%. All prices in 2013-USD.

Configurations 2 and 3 run comparably, with Configuration 3 performing better at the margin. For reference, we compare the external cost of the best-performing configuration (Configuration 3) against what we would get if there was no local energy market and the community's energy needs in their entirety were satisfied by the grid. According to Fig. 3.12, at the end of the 2-month run, the community's external cost drops by almost half (42% decrease). N.B. that this difference is *not* equal to cost savings, since we do not account for the money that is exchanged *within* the community whenever the local market clears. This difference, however, translates directly to aggregate welfare enhancement, since it corresponds to either money that is not spent — because prices in the internal market are lower than the grid's retail rate —, or money that changes hand inside the community, from one prosumer to another.

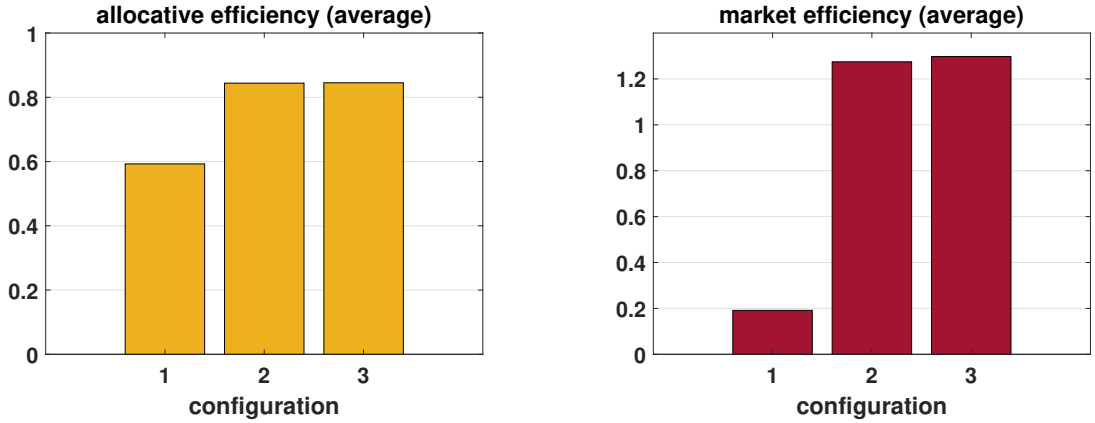
Finally, we examine the market and allocative efficiences under each configuration, as is the case in [144] and [122] respectively.

The *market efficiency* (Eq. 3.1) is defined as the ratio of energy demand met internally over the energy demand imported from the grid for a given slot. For slot s_i:

$$M_i = \frac{E_{trd,i}}{E_{imp,i}} \tag{3.1}$$

The *allocative efficiency* (Eq. 3.2) is defined as the energy that was traded internally ($E_{trd,i}$) divided by all the energy that could have been traded in that slot. For slot s_i, assuming $E_{buy,i}$ ($E_{sell,i}$) is the aggregate energy requested (offered) across all buy (sell) orders for that slot, we have:

$$A_i = \frac{E_{trd,i}}{min(E_{buy,i}, E_{sell,i})} \tag{3.2}$$



(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.

Figure 3.13 Indexes used to assess the health of the local energy market: market, and allocative efficiency. These are averaged over the course of the case study. Higher is better. Regardless of index we observe that the efficiency of the market is worse under Configuration 1, and comparable between Configurations 2 and 3.

We consider both the averages (Fig. 3.13) and the L2-norms of these metrics (Table 3.5). The L2-norm (Eq. 3.3) is defined as:

Table 3.5 L2-norms $\|x\|_2$ for the market and allocative efficiences (higher is better), the total energy cost for the community (lower is better), as well as the energy trade (higher is better). "imported" energy stands for energy that is bought from the grid, "exported" is surplus energy that is sold to the grid, and "traded" stands for energy that is traded internally between prosumers. The market efficiency of Configuration 1 is approximately 10× smaller than the competition. Same goes for the energy that is traded internally, where the L2-norm is approximately half of that of Configurations 2 and 3. Configurations 2 and 3 perform comparably.

configuration	efficiency		energy cost ($\times 10^7$)	energy ($\times 10^3$)		
	market	allocative		imported	exported	traded
1	26.86	49.09	2.382	0.919	1.625	0.191
2	252.40	59.90	2.110	0.857	1.449	0.381
3	261.53	59.94	2.105	0.856	1.447	0.383

$$\|x\|_2 = \sqrt{\frac{1}{S}\sum_{j=1}^{S}(x_j)^2} \tag{3.3}$$

where S is the total number of 15-minute slots evaluated in this case study with $S = 5758$ for our 2-month timeframe. For completeness, Table 3.5 is expanded with the L2-norms of the total energy cost for all the households in the community, and the energy quantities that were imported (bought from the grid), exported (sold to the grid), or traded internally.

Across all metrics, the common theme is that of Configuration 1 having a substantially negative impact on the performance of the energy market, compared to the rest of the configuration. For instance, its market efficiency is approximately 7 times lower on average (Fig. 3.13), or 10 times worse when comparing L2-norms (Table 3.5). The L2-norm on energy traded internally is approximately half of that of the competition.

Intuitively, these results make sense and build upon the analysis of the previous section (Section 3.4.2). When the market runs on a TMP with decreased effective throughput, as is the case with Configuration 1, it observes less bids and offers per slot. This translates to the market clearing at a suboptimal point. We *know* this is a suboptimal operation point, because of Configurations 2 and 3 confirm there is room to grow; a LEM running under either of these configurations sees the demand that can be met internally increasing, and its external costs

decreasing.

When it comes to comparing Configurations 2 and 3, we find both to be comparable across the board; Configuration 3 is only performing marginally better.

However, this improvement comes with an asterisk; the design (Section 3.3.2) is excessively centralized and makes the key-carrying node — the single auctioneer that invokes the `markEnd` call — a single point of failure. Even if we set ideological objections to centralization aside, the downsides of such an approach are concrete and practical:

1. This node's private key vk_i for slot s_i can be stolen by an adversary, or shared privately with them in the case of bribery. The adversary then would have an assymetrical market advantage during the bidding for s_i, as all the encrypted bids would be known to them.

2. To a lesser degree, the market under Configuration 3 also runs the risk of the key-holding ESCO node deciding to "not show up", i.e., not invoke the `markEnd` call for s_i. When that happens, no trading takes place during s_i since all the bids remain sealed. Even though in practice, such a risk would be mitigated via collaterilization[27], it is still worth noting.

3.5 Conclusions

In this work, we *design* and analytically evaluate an auction-based local energy market on a permissioned blockchain. This is a departure from the current related literature that studies the intersection of blockchains and energy markets by treating the blockchain layer as a *black box*. As our work demonstrates, the design choices in the blockchain layer of a blockchain-based LEM should be explicitly called out and evaluated for possible trade-offs, because they affect both the welfare of the community, and the network's usage of computing resources.

Our work sets precedence on the blockchain design space definition, and explicitly analyzes its impact on the performance, governance, and decentralization of the LEM. We utilize the

[27]The ESCOs that assume the key-holding service would sign a deposit-backed contract, as is the case with the ordering service nodes in Section 3.3.2

open-source blockchain platform Hyperledger Fabric [11] for the blockchain layer of our market, and design and present in detail the underlying *smart contract* architecture, the operational parties, and their roles. We also design and implement the market mechanism that sits atop the blockchain layer; to the best of our knowledge, our work is the first to explicitly identify *how* a closed-order book double auction can be implemented on a blockchain-based LEM.

Finally, a numerical evaluation demonstrates the applicability of our model to a real-world case, and its ability to provide insights on parameter choices during both market infrastructure planning and day-to-day LEM operation. Our case study, based on residential electricity usage data, showed (Section 3.4.2) how a change in the blockchain data model can decrease the market efficiency by approximately 90% (switch from Configuration 2 to Configuration 1), or — as another example — how a change in the way bids are encrypted can result in further market improvements, but at the risk of subverting the proper operation and resilience of the market (switch from Configuration 2 to Configuration 3). The simulation framework that we developed for the numerical evalution has been open-sourced and is available in [60].

We consider that the presented work can be adopted as a canonical framework for designing and evaluating blockchain-based LEMs. Part of our future work includes extending our model to evaluate the use of intelligent market agents, and analyze the market gains that can be had out of this change.

Chapter 4

A Smart Contract Example in the Energy Sector: Adaptive Multi-Tiered Resource Allocation Policy for Microgrids

An edited version of this chapter was originally published in [55].

4.1 Introduction

Utility tariffs that employ time-of-use pricing schemes or demand charges are becoming the norm for high-demand customers, in an effort to keep their demand under control ([84, 181, 209, 216]). On the customer side, an often used solution is to deploy a large capacity battery, charge it during off-peak hours when the electricity is cheap, and have it absorb part of the building's needs during the on-peak period of the day. Most of the time however, it is not economical for a single customer to deploy such a solution on their own, due to the high battery cost that outweighs the incurred energy cost savings.

Therefore, several buildings within proximity of each other may partner and collectively buy a single battery to serve the needs of the whole cluster. An energy consumption controller deployed in the cluster, along with smart meters with bidirectional connectivity in each building are used to ensure that the battery is (i) used optimally for the cluster, and (ii) shared fairly among its members.

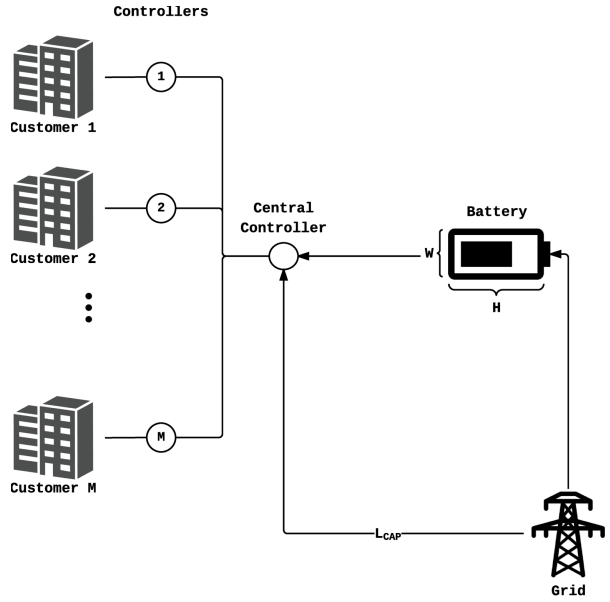


Figure 4.1 System diagram.

We consider a group of customers that share a battery for peak-shaving purposes, and also sign a group contract with the utility where they effectively get a lower energy unit price in exchange for guaranteeing their power consumption won't exceed a certain threshold. The lower the threshold (the tighter the constraint), the lower the contract price. If the load exceeds the contract rate, the cluster is paying a premium for every unit above the threshold. Our goal is to minimize the amount of energy drawn at a premium.

From the utility's point of view, the cluster is treated as a single, atomic entity for billing purposes. However, when it comes to the group, each participant is charged separately for

their own usage. This necessitates the need to monitor the intra-group demand on a per building/stakeholder basis, and motivates us to examine the issue of *fair resource sharing*. We group the cluster customers into high- and low-demand tiers based on a stochastic characterization of their demand profiles. Our second, parallel goal then is to ensure that the delta between the unit price that each customer is paying is as small as possible.

In order to achieve these goals, we examine two standard resource allocation policies with complementary benefits. We then introduce our own hybrid adaptive policy which allows us to find a sweet spot between the two previous policies. We compare all three and perform sensitivity analysis to identify trends.

We begin with an overview of related and motivating work in Section 4.2. In Sections 4.3 and 4.4 we present our model formulation, and resource-allocation policies respectively. We proceed with the performance evaluation of those policies in Section 4.5, and go over our conclusions in Section 4.6.

4.2 Related Work

Caron et al. [47] consider customers within a local distribution network that have access to the instantaneous total load on the grid, and introduce a distributed 'time/slackness' stochastic strategy that reduces the total cost for the customers and makes the overall load profile flatter. Koutsopoulos et al. [142] develop two online scheduling policies where demands are queued if the current load is higher than a threshold, and activated again either right before their deadline expires, or when there is enough residual energy. Using Jensen's inequality they also derive a lower bound on the average cost performance of all considered scheduling policies. Samadi et al. [192] consider a utility function for the customers and treat this as a welfare maximization problem, solved using a distributed pricing algorithm. Kim et al.[138] assume a time-varying price of electricity and a scheduler with statistical knowledge of future prices, and solve the resulting cost minimization problem by means of a Markov decision process. Alizadeh et al. [8] consider a neighborhood scheduler that queues the energy requests of

'deferrable' loads and optimizes the time at which they are served, so as to lower the overall cost and allow distributed energy resources to contribute. Fahrioglu et al. [87] look into how the utilities could design incentive-compatible contracts for effective demand management, using game theory principles.

Finally, we see parallels to this problem with the one examined by Goudarzi et al. [101] in multi-tier cloud computing systems, where the clients have Service Level Agreements (SLAs) and the system's profitability depends on the resource allocation that will allow these SLAs to be met.

4.3 Model Formulation

In [19], the behavior of a power consumer is modeled as a k-state continuous time Markov Chain, since it is the random superposition of different appliances, themselves considered as multi-level on-off sources, according to [185]. We adopt the modeling assumptions presented in [142] because (i) they capture the bursty nature of arriving requests, as well as (ii) the fact that the chances of having large durations decrease fast, and (iii) they allow for mathematical tractability. We do not introduce any new assumptions.

In our model, a cluster (or 'network') comprised of M buildings generates power requests (or jobs). During a peak consumption period T, each building $n \in [1, M]$ issues requests (or jobs) according to a homogeneous Poisson process with an arrival rate λ_n. The time duration of each demand $\tau_{n,j}$ (where $j \in \mathcal{N}^+$ is a request counter), is exponentially distributed with parameter μ_n.

The power requirement $p_{n,j}$ of each request is also considered exponentially distributed with parameter κ_n, and is measured in kW. Each customer request is then characterized by the tuple:

$$(\alpha_{n,j}, \tau_{n,j}, p_{n,j})$$

Table 4.1 Notation.

Parameter	Description
T	Duration of peak period. Measured in hours.
M	Number of buildings in cluster.
n	Building identifier. $n \in [O, M]$.
λ_n	Interarrival rate for power requests of building n. (Rate of a homogeneous Poisson process.)
μ_n	Service rate for power requests of building n. (Rate of a homogeneous Poisson process.)
κ_n	Rate that determines the power level of the requests of building n. (Rate of a homogeneous Poisson process.)
j	Request identifier. $j \in \mathcal{N}^+$.
$\alpha_{n,j}$	Arrival instant of request j of building n. Measured in hours. $\alpha_{n,j} \in [O, T]$.
$\tau_{n,j}$	Time duration of request j of building n. Measured in hours. $\tau_{n,j} \in [O, T]$.
$p_{n,j}$	Power level of request j of building n. Measured in kW. $p_{n,j} \in \mathcal{R}^+$.
W	Nominal power rating of battery. Measured in kW.
H	Discharge duration of battery at its nominal power rating W. Measured in hours.
η	Charging efficiency of battery. Unitless.
C	Battery capacity. Measured in kWh.
RAF_n	Resource allocation factor for building n. Unitless.
AUC_n	Average unit cost of energy for building n. Measured in $\$/kWh$.
$d_{n,j}(t)$	Demand profile of request (n, j). Measured in kW.
$L(t)$	Aggregate instantaneous load on the cluster. Measured in kW.
$J(t)$	Number of active requests on the network at time $t \in [O, T]$.
L_{CAP}	Power threshold set in contract for discounted electricity. Measured in kW.
$E(t)$	Quantity by which $L(t)$ exceeds L_{CAP}. Measured in kW.
$L_{GC}(t)$	Load served by the grid at the contract rate. Measured in kW.
$L_{BAT}(t)$	Load served by the battery at the contract rate. Measured in kW.
$L_{GP}(t)$	Load served by the grid at the premium rate. Measured in kW.
p_1	Unit cost for every kWh drawn from grid-contract. Measured in \$.
p_2	Unit cost for every kWh drawn from the battery. Measured in \$.
p_3	Unit cost for every kWh drawn from grid-premium. Measured in \$.
p_{BAT}	Factor by which p_2 is greater than p_1. Unitless.
p_{GP}	Factor by which p_3 is greater than p_1. Unitless.

where $\alpha_{n,j}$ is the interrarival time of requests in hours (hence exponentially distributed with parameter λ_n), $\tau_{n,j}$ is the duration of the request in hours, and $p_{n,j}$ is the instantaneous power consumption in kW, considered constant throughout $\tau_{n,j}$.

Adopting the notation introduced in [47], the demand profile of each customer request is defined as:

$$d_{n,j}(t) \coloneqq p_{n,j} \times \mathbb{1}_{\{\alpha_{n,j} \leq t \leq \alpha_{n,j}+\tau_{n,j}\}}$$

where $\mathbb{1}$ is the indicator function and $t \in [O, T]$.

Likewise, the aggregate instantaneous load on the network is defined as:

$$L(t) \coloneqq \sum_{n=1}^{M} \sum_{j} d_{n,j}(t)$$

The number of active requests on the network at any given time $t \in [0, T]$ is:

$$J(t) \coloneqq \sum_{n=1}^{M} \sum_{j} \mathbb{1}_{\{\alpha_{n,j} \leq t \leq \alpha_{n,j}+\tau_{n,j}\}}$$

The cluster signs a 'tight guarantee' contract with the utility for a threshold of L_{CAP} kW. The unit cost for every kWh consumed when the power demand is equal to or less than L_{CAP} kW, is p_1. The integral over time of every kW of demand beyond L_{CAP} kW is priced at a premium $p_2 = p_{BAT} \times p_1$ per kWh (i.e. p_{BAT} is a factor of the contract rate).

The grid is equipped with a battery rated at W kW, a discharge duration of H hours at its nominal power rating, and a charging efficiency η [98]. At the beginning of T it is fully charged with a capacity of: $C \coloneqq W \times H$ kWh.

Now, let $E(t)$ be the remainder of the cluster's aggregate instantaneous load beyond the contract cap of L_{CAP} kW, that is:

$$E(t) \coloneqq \max(0, L(t) - L_{CAP})$$

Further, let L_{BAT} be the load served by the batter, and L_{GC}, L_{GP} be the loads served by the grid at the contract and premium rate respectively. During peak hours, the cluster's power requests are served in the following order:

1. When $L(t) \leq L_{CAP}$ all the demands are satisfied by the grid at the low, contract rate of p_1 per *kWh*.

2. When $L_{CAP} < L(t) \leq L_{CAP} + W$:

$$L_{BAT} = E(t)$$

   This is the demand served by the battery at a rate of p_2 per *kWh*.

3. When $L(t) \geq L_{CAP} + W$, $E(t)$ is greater than the battery's nominal rate (W *kW*), there will be a remainder that is served by the grid at a premium $p_3 = p_{GP} \times p_1$ per *kWh*.

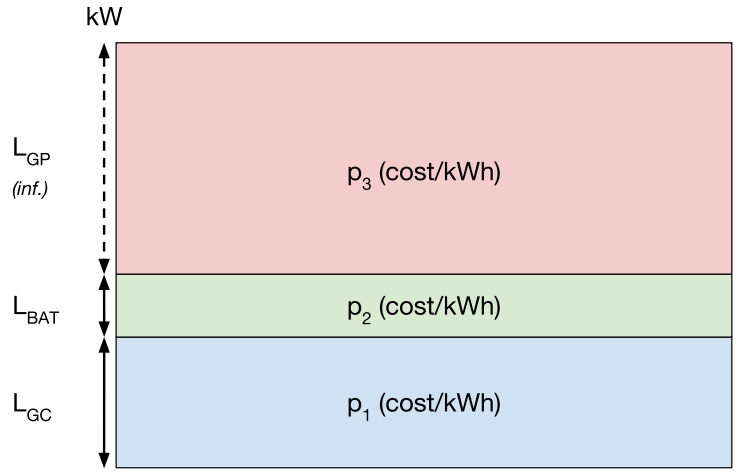


Figure 4.2 The three servers/bands in our system.

Note that, in the policy that we consider in this work, the charge p_2 of drawing from the battery is higher than the contract rate p_1, even though the battery is charged during off-peak

hours when the unit cost of electricity is cheaper. The difference is the self-imposed charge by the cluster in order to pay off the amortized procurement and investment battery costs. The cluster could also settle on a price-point p_2 that is less than p_1 (i.e. $p_2 < p_1$), depending on the discount rate, the amortization period, and the agreed upon payback period of the investment (book life). The former option constitutes a more aggresive policy, and is the one we focus on here. At any rate, the unit cost p_2 for drawing from the battery should be lower than that of drawing from the grid at a premium (p_3), otherwise one could just skip the battery in this setup altogether. We therefore have:

$$p_1 < p_2 < p_3$$

As shown in Fig. 4.2, we are effectively dealing with three *servers* that we access in order of increasing cost: grid at the contract rate (**grid-contract**), **battery**, and grid at a premium (**grid-premium**). Each energy request from the buildings is serviced *immediately*, i.e. there is no deferral. As noted in [142], when each request is served upon arrival, $J(t)$ can be thought of as the occupation process of an $M/M/\infty$ service system; it is therefore a continuous time Markov chain whose steady-state probabilities can be derived from equilibrium equations [26]. The adopted notation is summarized in Table 4.1.

4.4 Resource Allocation Policies

We consider the following scheduling policies, all of them put into effect by a network-wide controller that regulates the power flows for every member of the cluster; see Fig. 4.1 for a conceptual system diagram. If we assume that all of the members in the cluster participate in a private blockchain network, the function of this network-wide controller can be regulated by a smart contract that is audited by all members of the network.

In the 'strict bounds' policy (Fig. 4.4), we use the stochastic characteristics of the requests generated by each building, to define a factor proportional to its estimated power needs. We

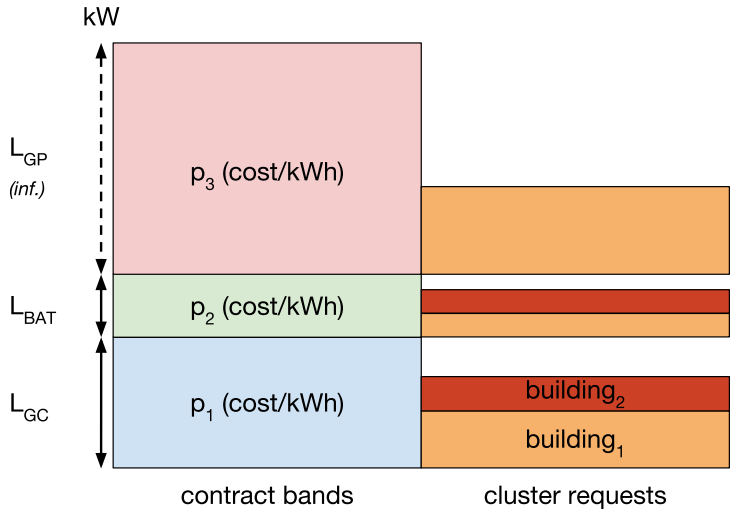


Figure 4.3 An example of how the 'strict' policy works for a given time point t. Buildings 1 and 2 are the only active customers in the cluster at t. Buildings 1 and 2 draw *power* from grid-contract at a rate defined by their RAF. They can draw *energy* from the battery server up to a quantity defined by their RAF; the rate at which the draw from the battery however is fixed at W/M. Any excess demand is served by grid-premium.

call this the **resource allocation factor** and it is defined as follows:

$$RAF_n := \frac{\frac{\lambda_n}{\mu_n \times \kappa_n}}{\sum_n \frac{\lambda_n}{\mu_n \times \kappa_n}} \tag{4.1}$$

We then use this quantity to establish bounds for each building when drawing from the grid or the battery. Specifically, each building can draw up to $RAF_n \times L_{CAP}$ kW from the grid, and up to $RAF_n \times C$ kWh of the battery's initial charge at a maximum rate of W/M (kW). Any demand above these thresholds is accommodated by the grid at a premium.

In the 'no bounds' policy (Fig. 4.4), all customer jobs are queued according to a first-come, first-served (FCFS) logic. The grid-contract server picks up jobs from the head of the queue until $L(t)$ reaches L_{CAP} kW, then the battery comes into play until it becomes full; finally, the grid-premium server picks up any outstanding requests.

The 'adaptive bounds' policy (Fig. 4.5) is a hybrid of the previous two; when the aggregate

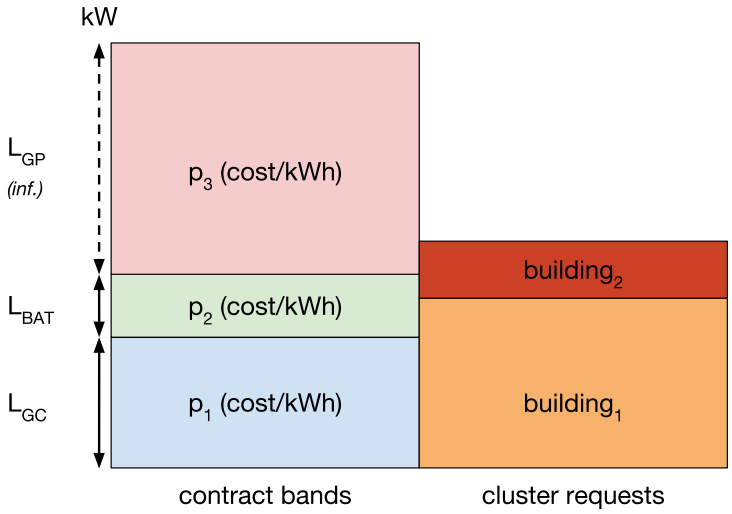


Figure 4.4 An example of how the 'no bounds' policy works for a given time point t. Buildings 1 and 2 are the only active customers in the cluster at t, and their requests as served in first-come, first-served basis. As a result of coming in first, Building 1's power needs are served by the lower-priced tiers, grid-contract and battery. Compare and contrast with Building 2, whose needs are served exclusively by the battery server, and grid-premium at this point in time.

instantaneous load on the network is less than the grid contract cap L_{CAP}, access to the grid-contract resource is not constrained. If the about-to-be-admitted job's power level is such that the total load will exceed the cap, the controller switches to a strict bounds regime. If the overall demand $L(t)$ drops under L_{CAP} again, the 'no bounds' policy is put into effect again, and any throttling at the grid-contract level is ceased.

Note that contrary to most of the work presented in Section 4.2, we do not defer load requests; we attempt to identify a resource allocation policy that improves the cluster's welfare, while serving its requests without delays.

To that effect, we track two output performance measures. One, the amount of energy that the cluster draws from the grid at a premium; this is a quantity that we wish to minimize so as to decrease the utility charges. Two, the average unit cost of energy per server for each customer. We expect this to be higher for a building that cannot tap into the grid-contract

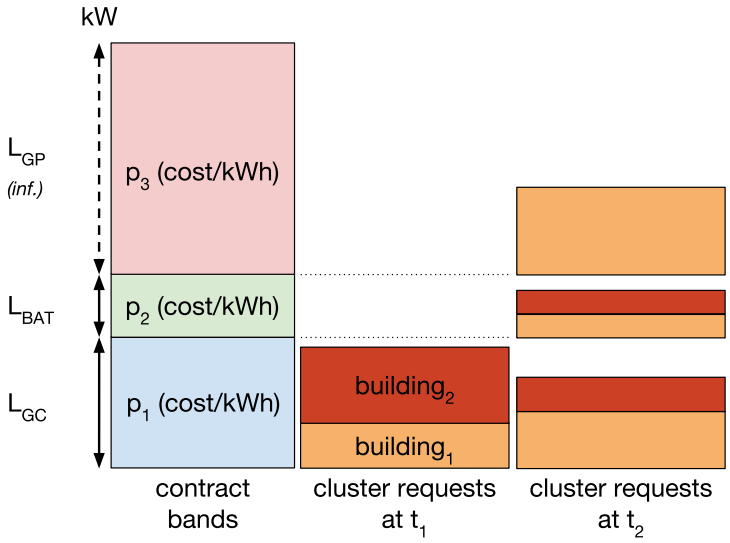


Figure 4.5 An example of how the 'adaptive' policy works for two given time point t_1 and t_2. Buildings 1 and 2 are the only active customers in the cluster at both time points. At t_1, the aggregate demand is below L_{CAP}, so the requests are accomodated according to the 'no bounds' policy. At time t_2, the overall demand exceeds L_{CAP}, so the controller switches to the 'strict' policy for serving requests.

server with the same frequency as another building. In a fair resource allocation regime, the relative difference (or standard deviation) of the average unit costs per server for all buildings should be minimal.

Our expectation is that the 'strict bounds' rule will be the fairest of all considered policies, since each customer has uncontended access to their own 'power' band in the grid-contract and battery servers[1]. This is conditional on our definition of the resource allocation factor, and its appropriateness as an index for each building's energy consumption throughout the peak period.

The 'no bounds' policy is expected to perform well in terms of peak shaving for the whole cluster, since it stacks the incoming jobs on top of each other, without leaving any unused resources in the lower-priced servers (grid-contract and battery).

[1]Note that there is no point in assigning bounds for the grid-premium server, since its resources are effectively infinite.

Finally, we expect our 'adaptive bounds' policy to combine the positive characteristics of both previous policies to some degree; the fairness of the 'strict bounds' policy, and the load compacting of the 'no bounds' one. The underlying algorithm behind our 'adaptive' policy is shown in Algorithm 1.

Algorithm 1 Adaptive algorithm.

$t \leftarrow 0$

for $n \in [0, M]$ **do**
    $UC_n \leftarrow 0$ (kWh) ▷ UC := usage counter

while $t \in (0, T]$ **do**
    if $L(t) < L_{CAP}$ **then**
        $P \leftarrow NB$ ▷ P := policy, NB := no bounds policy
        $Q_n \leftarrow L_{CAP}$ (kW) ▷ Q := quota
    else
        $P \leftarrow SB$ ▷ SB := strict bounds policy
        $Q_{n@gc} \leftarrow L_{CAP} \cdot RAF_n$ ▷ Units: kW

        while $UC_n \leq C \cdot RAF_n$ **do**
            $Q_{n@bat} \leftarrow W/M$ ▷ Units: kW
        $Q_{n@bat} \leftarrow 0$ ▷ Units: kW

4.5 Performance Evaluation

All the results below are the averages of 100 replications with 95% confidence intervals.

We consider 10 buildings in our cluster; all of them share the same stochastic characteristics when it comes to the power requests they generate. On average, jobs arrive once per minute, last 50 seconds, and have a power level that we gradually increase in each batch of replications from 20 to 120 kW (this is the x-axis of Fig. 4.6). For the battery we wish to pick a capacity that (a) prevents the cluster from reaching to the grid-premium server when we're at the lower range of the cluster's capacity, and (b) does push the cluster toward grid-premium otherwise,

so that we can investigate the interaction of the cluster with all three bands, as the aggregate power demands of the cluster increase. We therefore assume that a 1.5 MWh Lithium-ion battery with 90% round-trip efficiency [90] is shared by the cluster, i.e. W=500 kW, H=3 hrs, C=1500 kWh, η=0.9. This is the rounded maximum capacity at which the cluster does not draw power from the grid-premium server.

Fig. 4.6a and Fig. 4.6b quantify how well each policy can utilize the available resources, as the load on the network increases. The goal is to minimize the cluster's exposure to the costly grid-premium server, so we want the energy drawn from it to be as small as possible.

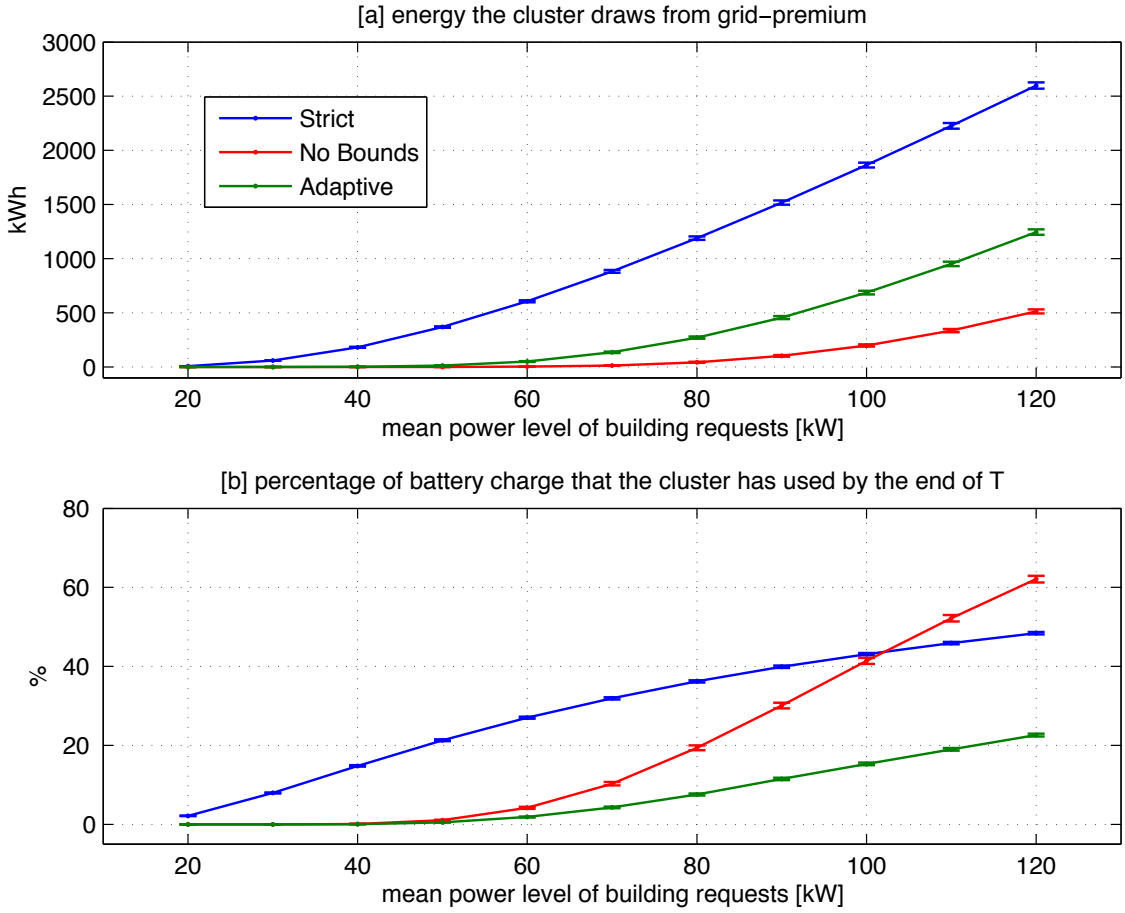


Figure 4.6 Utilization of premium-priced resources as the network load increases. (a) Top: Energy the cluster draws from grid-premium. (b) Bottom: Percentage of the initial battery charge C that the cluster has used by the end of T.

In that regard, as Fig. 4.6a shows, the 'strict bounds' policy performs the worst, and the 'no bounds' policy the best, due to its ability to "stack" the jobs on top of each other and fully use the resources of the grid-contract server, before moving to the more expensive battery server, and then to the most expensive grid-premium server. The 'adaptive bounds' policy lies in between those two. In the beginning, it stays close to the performance of the 'no bounds' policy; the instantaneous load on the network is such that the policy can accommodate within the grid-contract server. As the load increases, the number of times the adaptive policy switches over to the strict policy increases; this is when the distance between the 'no bounds' and the 'adaptive' curve begins to grow in Fig. 4.6a. Finally, the load becomes so large that the adaptive policy effectively degenerates to the strict one; this corresponds to the rightmost side of the figure.

The adaptive policy is overall closer to the performance of the 'no bounds' policy than the strict one, i.e. it does not just perform as the average of these two. Given that the 'no bounds' policy here is the optimal one, this result is highly desirable. Notice how all three strategies begin at the same starting point; when the demand is low enough that access to the grid-premium server is minimized, all policies perform the same when it comes to absorbing the cluster's demand via the grid-contract and battery servers.

Fig. 4.6b complements Fig. 4.6a by showing how the battery server is used. Beginning with the leftmost side of the figure, we see that the strict rule is the first one to access the battery server; as soon as any building hits the ceiling of their allocated band in grid-contract, it moves on the battery server, regardless of any unused resources in the rest of grid-contract. On the contrary, both the 'no bounds' and adaptive policies defer access to the battery server until grid-contract is full. The results on the rightmost end will also be interpreted in conjunction with our observations on Figure 3 below. Given that access to grid-premium is inevitable due to the high aggregate load, the fact that the 'no bounds' policy uses up most of the battery charge is a desirable quality; the more it draws from the battery, the longer it delays its transition to the premium band. The adaptive policy uses the battery less than the strict

rule, but this is because it utilizes grid-contract in a far more optimal manner.

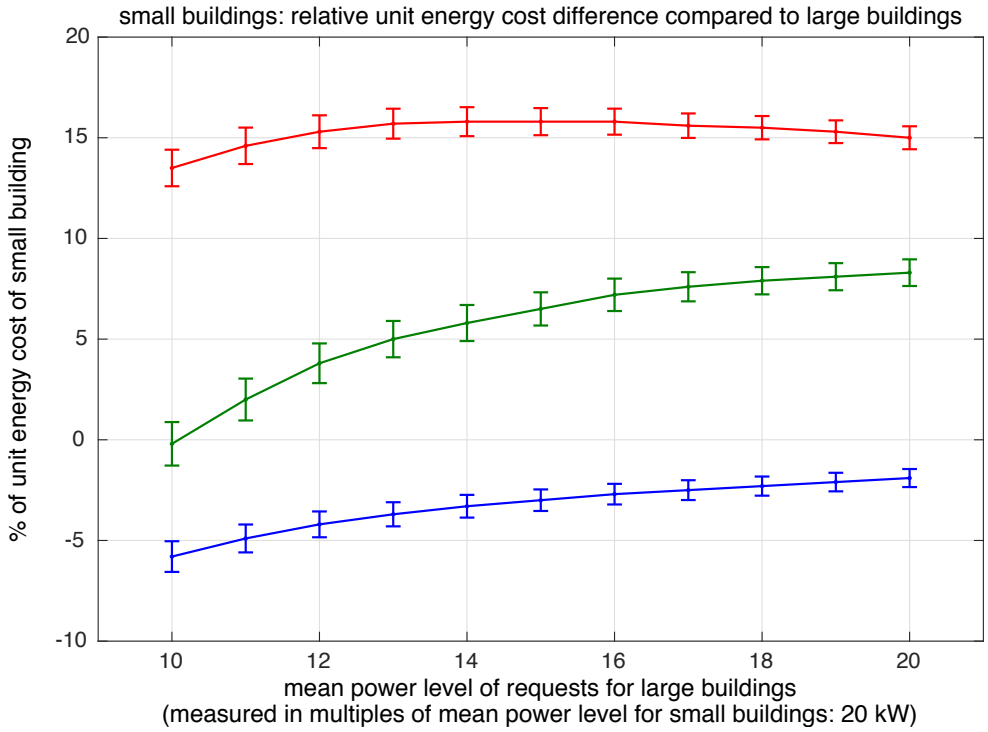


Figure 4.7 Small buildings: relative unit energy cost difference compared to large buildings. The units in the x-axis are measured in multiples of 20 kW, the mean power level of small buildings.

In Fig. 4.7 we are switching our focus to fairness. Let us consider 9 large buildings and 1 small one in our 10-building cluster. A building falls into the 'large' tier if its average power level is at least an order of magnitude larger than a building from the 'small' tier. In this example, the small building generates requests with an average power level of 20 kW. For the large buildings in the cluster, this number begins at 200 kW, or 10 times the level of a small building, and eventually grows to 400 kW (this is the x-axis of Fig. 4.7). The mean interarrival and service time rates remain fixed for all buildings at $\frac{1}{60 \text{ seconds}}$ and $\frac{1}{50 \text{ seconds}}$ as before. (Note that tiers could have also been drawn by changing how often the buildings generate requests, or how long their requests need to be serviced for, i.e. by modifying the λ or the μ parameter

in the (λ, μ, κ) tuple and keeping all other values fixed.)

Going back to the considered setup, this is a network where the requests coming from large buildings dominate the network when it comes to using its resources, and as we increase their power level, this phenomenon becomes even more prevalent. It is therefore a good stress test for the fairness performance of all policies. We evaluate fairness as follows: at the end of each run, we multiply the amount of energy each building drew from each of the three servers by their unit cost. We add up these products, and divide the result by the overall energy usage of the building:

$$\begin{aligned} AUC_n \coloneqq & \frac{p_1 \times \int_0^T L_{GC,n}(t)dt}{\sum_j p_{n,j} \times t_{n,j}} \\ & + \frac{p_2 \times \int_0^T L_{BAT,n}(t)dt}{\sum_j p_{n,j} \times t_{n,j}} \\ & + \frac{p_3 \times \int_0^T L_{GP,n}(t)dt}{\sum_j p_{n,j} \times t_{n,j}} \end{aligned}$$

This gives us the average unit cost of energy (notated as AUC in the formula above) for each building.

In a fair policy, the standard deviation between the unit costs of all buildings should be as small as possible. In our case, since we are dealing with 9 large buildings which have similar unit costs due to their identical stochastic characteristics, we focus on the relative difference between the two tiers as a less biased metric of fairness:

$$\frac{AUC_{\text{small}} - AUC_{\text{large}}}{AUC_{\text{small}}}$$

As with the standard deviation, we wish to keep that difference as small as possible.

Examining Fig. 4.7 we observe that, as the network becomes saturated with power requests, all policies eventually settle to a steady state, but the levels at which they settle are different. The

strict strategy performs the best, with the relative difference remaining within 5% throughout the entire range of scenarios considered. The adaptive policy is slightly worse, due to its operation at times as 'no bounds'; it settles at a relative difference in unit costs of around 7%. However, it constitutes a significant improvement over 'no bounds', improving its fairness by a factor of two.

For the adaptive policy, also observe that under heavy load, the unit cost for small buildings is slightly higher than that of large buildings. This happens because in the adaptive regime, when access to grid-contract happens on a first-come, first-serve basis (i.e. when it is effectively operating as 'no bounds') there will be instances that this server's resources are used entirely by the dominant traffic generator on the network, that is, the 'large building' tier. Such occurrences bring the tier's average unit cost down. This is not possible in the 'strict' regime since the large buildings are always limited to their own allocated bands and cannot capture grid-contract in its entirety; this is why the strict policy practically tends to zero.

Finally, we examine how sensitive the policies are to changes in the access costs of the premium-priced servers (battery and grid-premium), and whether they pass along this cost difference to the large and small customers in the cluster in *a proportional manner*. This is always tied directly to the aggregate load on the cluster; performing this sensitivity analysis on a different operating point will bring about different results. However, for a given operating point, this evaluation allows us to identify how each policy utilizes the premium-priced resources (battery and grid-premium) and whether it passes along the cost increases fairly (i.e. does the relative unit cost difference stop increasing as we move to the right in Fig. 4.8?).

In our considered setup, the 'large building' tier now generates requests with a fixed average power level of 300$ kW. All other building-related parameters remain the same as before. This is a heavily loaded network that keeps all servers busy.

In Fig. 4.8a, we are gradually increasing the cost parameter p_{GP} that applies when accessing grid-premium (this is the x-axis of Fig. 4.8a). This corresponds to the scenario where the cluster is willing to sign a contract with tighter guarantees, and thus a higher premium if

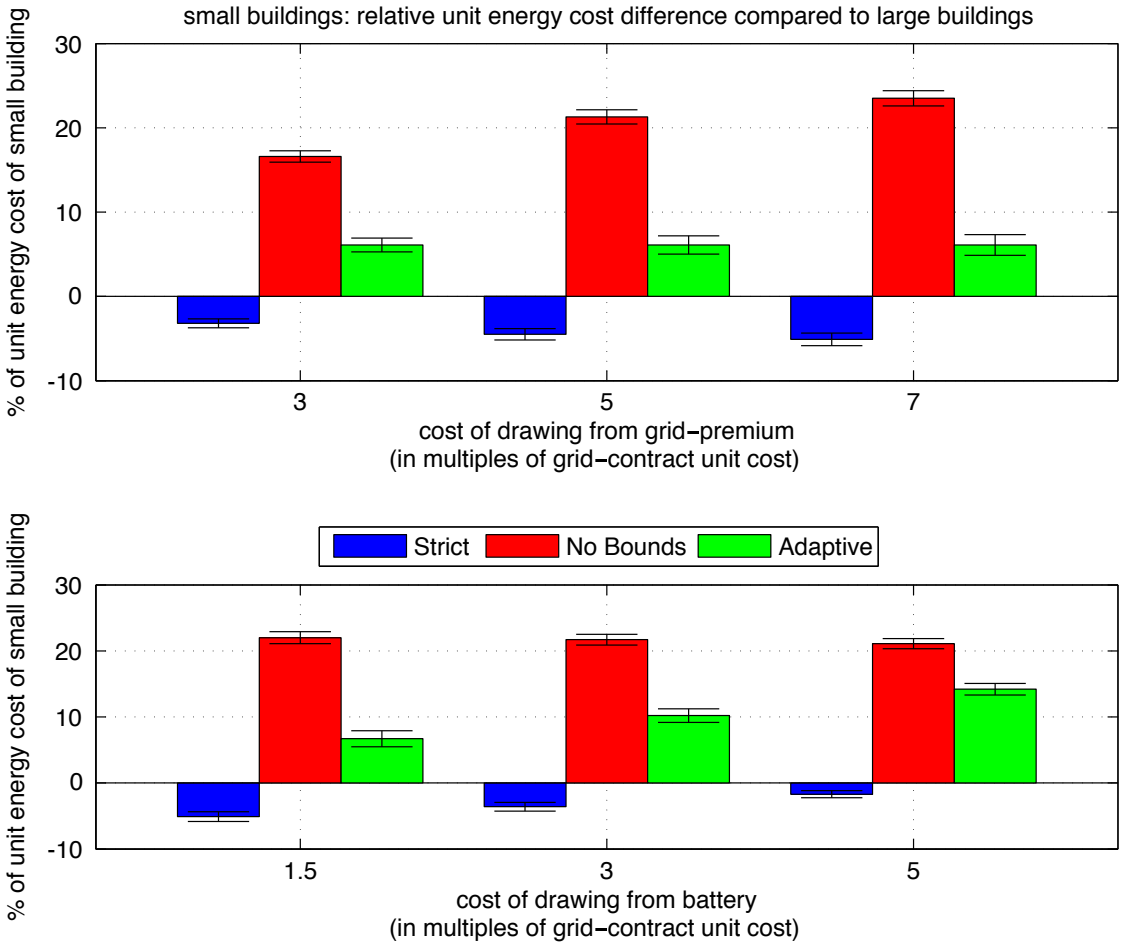


Figure 4.8 Small buildings: relative unit energy cost difference compared to large buildings. (a) Top: p_{GP} ranges from 3 to 7; p_{BAT} remains fixed at 1.5. The strict policy drew 910 *kWh* from the battery and 8,952 *kWh* from grid-premium. Those numbers are 1,500/6,047 for the 'no bounds' policy, and 643/7,303 for the adaptive one. (b) Bottom: p_{BAT} ranges from 1.5 to 5. p_{GP} remains fixed at 7. The strict policy drew 910 *kWh* from the battery and 8,952 *kWh* from grid-premium. Those numbers are 1,500/6,047 for the 'no bounds' policy, and 643/7,303 for the adaptive one.

these guarantees are not honored. (The reward is cheaper access to grid-contract.) In Fig. 4.8b, we keep the cost of accessing grid-premium fixed, and instead gradually increase the cost of drawing from the battery (while keeping it below the grid-premium unit cost at all times). This could correspond to the case when the cluster increases the self-imposed fee of accessing the battery in an attempt to pay off the battery installation costs sooner, or invest in more capacity.

We observe (Fig. 4.8a) that the relative difference between unit costs of small and large customers gradually increases in the 'strict' policy. This happens because the large customers are, in relative terms, more exposed to the grid-premium band than the small customer. (Remember that access to the grid-premium server is unrestricted in all policies.) Therefore, the average unit cost of the 'large-building' tier increases more than that of the small building. Conversely, and following the same logic, the relative difference decreases when the battery costs increase (Fig. 4.8b); for the power level that we picked for the small building, the percentage of its jobs served by the battery server compared to grid-premium is higher than that of the large buildings, where a sizeable portion is served at grid-premium. A similar argument can be made for the increase that the adaptive rule demonstrates in Fig. 4.8b; the percentage of small building jobs served at the battery is higher than that of large buildings jobs (a considerable portion of which is now served at grid-contract).

Notice that for the adaptive policy, as Fig. 4.8a shows, a cost increase in grid-premium leaves the relative unit cost difference between small and large buildings unaffected. Part of the large building jobs that were served in grid-premium under the strict regime, are now served in grid-contract (in those time instances when the instantaneous total load falls below grid-contract threshold). This leaves the same percentage of 'grid-premium'-served jobs for both types of buildings, so the cost increase affects them both in equal measure. This is, again, a side-effect of the way the adaptive policy utilizes the lower-priced servers (when operating under a 'no bounds' rule), and its advantage over the strict strategy. We note that in both scenarios, the adaptive policy remains a fairer policy than the 'no bounds' one, in consistence

with what we saw in the previous section (Fig. 4.7).

The analysis above focuses on (a) the relative ease of access that each type of customer (small/large) has to the grid-contract server, and (b) to the effect the price increases on premium-priced servers have across policies, both by means of the relative unit energy cost difference of small buildings compared to big buildings. Future extensions of this work may also wish to consider the frequency with which each customer accesses the battery when assessing fairness, as extensive battery usage leads to more charging cycles and eventually a degradation of the battery's life.

4.6 Conclusions

The adaptive policy combines the benefits of both the strict and the 'no bounds' policy. In light-load conditions, fairness is not an issue since the lower-tier power resources (grid-contract, and battery) are not congested, and all requests can be accomodated by them. The adaptive policy operates similarly to the 'no bounds' rule, thus allowing a cluster to utilize its own resources (grid-contract and battery) optimally, and to avoid premium charges from the utility. When the network is heavily loaded with power requests, fairness does become an issue; we want to ensure that the unit costs for grid-contract and battery for each building are comparable; no one resource should be dominated by a single building or group of buildings. For that reason, the adaptive rule evolves into a strict regime, limiting each customer's requests per server to their own properly-sized band. The width of these bands is calculated using the resource allocation factor formula introduced (Eq. 4.1).

Because the adaptive policy alternates between these two regimes when it most makes sense, it performs better than an average of the two base policies would. Our sensitivity analysis demonstrates that premium energy usage is closer to the 'no bounds' policy –the optimal in that regard–, than to the strict one. Similarly, its fairness is closer to the 'strict' strategy than to the 'no bounds' one. It is therefore an optimal combination of the two policies, suitable for a microgrid operator where equal weight is given to both cluster-wide cost

minimization and fairness among all customers.

Chapter 5

Directions for Future Work

5.1 Time-lock Puzzles for Encrypted Bids

The market mechanism in our work in Chapter 3 is a double auction market, implemented via a *closed* order book (Section 3.3.1), in accordance to what similar lines of work in the literature have considered. Unlike those lines of work, we actually described in detail how the closed order book specification would be implemented on a blockchain, and also implemented this in [60]. We considered two ways of achieving a closed order book: the preferred, standard way (Section 3.3.2), in which the bidders post bids encrypted with their own key in the bid phase, then decrypt those bids by posting their private key during the reveal phase, and an alternative way (Section 3.3.2) in which the bidders use the key of a rotating auctioneer, as a means to achieve better performance, but at the risk of subverting the resilience of the market (Section 3.4.2).

In the preferred way of encrypting, we still rely on the bidder to "show up" during the reveal phase and post their private key; without it, the bid remains locked. There is no incentive for the bidder not to do this, because the reservation price they set in their bids (Section 3.3.1) acts as a safeguard against their entering a bad trade. But bids may remain encrypted, either because the bidder's node crashed and didn't recover in time, or because their bid wasn't

posted in the network in time due to a temporary network partition (Section 3.3.2) — when that happens, we have a market clearing price for that slot that is calculated on incomplete data.

What we ultimately want then, is a system where the bids can be decrypted *by anyone* after a certain time, and not before[1]. [33] examines this problem in detail. It will be useful to explore an implementation of our work in Chapter 3 that treats the bids as "time-lock puzzles" [186] created via *chained hashes*.

Time-lock puzzles are computational problems that require a certain amount of time to solve. They need to be sequential, so that the CPU time needed to solve them approximates real time; otherwise an attacker can use multiple computers in parallel and find the solution before the desired time. With chained hashes, we can achieve just that; for example in [217], an encryptor with 32 computers available running for 8 hours, can create a ~256 hour timelock (32 8-hour chains).

Do note that the length of a timelock is only an approximation, and as such, a timelock scheme would not work when we want the information released at a precise time-point. As long as the slots in the energy markets are such that allow for some slack in the timed-release of the bids, this scheme could work.

5.2 Effect of Communication Links

In our model in Chapter 3, we asssume broadband links between the nodes in our system (Section 3.3.1); this is the norm for smart inverters that use the consumer's own Wi-Fi network to communicate with the utility's system. However, that is not always the case.

Consider the Solar Partner Program (SPP) [18, 182, 212] in Phoenix, AZ – a project where the regional utility has deployed PV panels across 1,670 rooftops – uses cellular links for communication. Also consider that not all consumers have broadband Internet access — and in some cases, they may not have Internet access at all. Yet utilities are required in certain

[1] That time for our case would be the end of a slot, when we calculate the market clearing price.

jurisdictions to provide equal access to all customers [199].

In general, there is a range of backhaul technologies and bandwidths available for interconnecting the nodes in the distribution grid; broadband (bandwidth: 10s of Mbps), cellular AMI (100s of kbps), or mesh advanced metering infrastructure (AMI) (10s of kbps) are a few of the options available.

It is worth investigating *how* a blockchain-based LEM scales under different bandwidth settings. For instance, can a set of nodes carrying cellular kits (example [202]) sustain a blockchain-based LEM? For comparison, "normal" latency for cellular links is in the hundreds of milliseconds (200 to 600 ms), while for DSL modems it is an order of magnitude smaller (10 to 70 ms) [226]. Related to this is a latency and bandwidth analysis of LTE for smart grid applications published in [231]. Taking this a step further, what happens when 5G — with even bigger improvements in latency (20-30 ms in the real world) — come into the picture?

Finally, it is also worth looking into what kind of hit the market efficiency takes when losses in packets are taken into account. Tangentially related to this is the work in [229], where the authors look into DER coordination when the communication links between generators are lossy.

5.3 Reputation-Based Markets

We covered atomic exchanges in Chapter 2. These work *great* when we exchange purely digital assets. As we saw in Section 2.3, blockchains are the perfect instrument for orchestrating atomic exchanges; they give complete, cryptographically-backed assurances that the exchange will be performed atomically and at the end of it, counterparty A will find themselves holding B's digital asset, and vice-versa.

The problem with all blockchain-based energy markets, however, is that we effectively exchange *promises*; commitments for something that will happen in future. These commitments may not materialize; the buyer in a transaction may become insolvent by the time the market actually clears, or the seller may be unable to meet the output they promised due to, say,

a forecasting error, or an issue with their generator. In all of these cases, we can mitigate the damage by using the exact same tools we use in traditional energy markets; deposits, collaterals, and corrections of the day-ahead market prediction errors in the real-time market.

Concretely, we can collateralize the sale by having the seller put money in escrow until the production actually takes place. When that happens, the seller gets paid back the escrow, plus the agreed-upon price for the amount of energy they produced. Otherwise, the escrow money is used to cover the part of the contract that the seller was unable to fulfill, by purchasing it from the real-time market. As an example: consider a seller that commits to producing M kWh of electricity during slot s_i for \$3. There is a risk that the real-time price at that time is going to be significantly higher than \$3. So, when they make the contract for the sale, the seller sends to a smart contract function that acts as an escrow \$10. If they end up producing M kWh, they get back the escrow amount plus the \$3 for the sale. If not, the escrow amount is used to cover the seller's contract by purchasing out of real-time markets. If this is, for example, \$8, the remainder (\$2) is returned to the seller, along with the \$3 of the sale. So the seller ends up paying for not fulfilling the contract[2].

While all of this can work, there is still a risk that the real-time market price is higher than the escrow; or that a buyer completely depletes their deposit. N.B. that this problem persists even if we introduce cryptocurrencies into the system, because commitments for production of energy are still a part of the equation.

How can one, then, leverage the real-time auditing capabilities that blockchains give us, in order to minimize or mitigate such edge cases of outstanding debt, in order to make "safer" trades. Put differently, how does overlaying a trust system on the local energy market look like? In [151, 233] the authors address the issue of decentralized trust when buying from anonymous vendors, by designing a Bitcoin wallet where people use shared accounts to take calculated risk towards their friends. Treating this as a maximum flow problem, they use the Ford-Fulkerson method [92] and calculate the total risk exposure of a buyer towards a vendor;

[2]As a sidenote, this right there is the economic disincentive for not accurately predicting energy usage.

the buyer can buy up to a calculated allowance amount without exposing themselves to any more risk than they were willing exposing themselves to before. It is worth examining how such a work can be applied to energy markets.

References

[1] *2016 State of the Market Report for the ERCOT Electricity Markets*. Tech. rep. Potomac Economics, May 2017.

[2] *A Pathway to the Distributed Grid*. Tech. rep. SolarCity, Feb. 2016.

[3] *A Stronger, More Resilient New York*. Tech. rep. PlaNYC, June 2013.

[4] Aysajan Abidin et al.
"Secure and Privacy-Friendly Local Electricity Trading and Billing in Smart Grid."
In: (Jan. 2018). arXiv: `1801.08354 [cs.CR]`.

[5] *Accord Project · Industry-Led Specification and Open Source Code for Smart Legal Contracts*. `https://docs.accordproject.org/`. Accessed: 2019-2-24.

[6] Roger Aitken. "IBM & Walmart Launching Blockchain Food Safety Alliance In China With Fortune 500's JD.com." In: *Forbes Magazine* (Dec. 2017).

[7] Muhammad Tanvir Alam, Haozhang Li, and Atul Patidar.
"Smart Trading in Smart Grid Using Bitcoin."
In: *Computer and Information Science* 8.2 (2015), p. 102.

[8] M Alizadeh, A Scaglione, and R J Thomas.
"From Packet to Power Switching: Digital Direct Load Scheduling."
In: *IEEE J. Sel. Areas Commun.* 30.6 (July 2012), pp. 1027–1036.

[9] Yackolley Amoussou-Guenou et al. "Dissecting Tendermint." In: *Networked Systems*. Springer International Publishing, 2019, pp. 166–182.

[10] Merlinda Andoni et al. "Blockchain technology in the energy sector: A systematic review of challenges and opportunities."
In: *Renewable Sustainable Energy Rev.* 100 (Feb. 2019), pp. 143–174.

[11] Elli Androulaki et al.
"Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains."
In: *Proceedings of the Thirteenth EuroSys Conference*. EuroSys '18.
Porto, Portugal: ACM, 2018, 30:1–30:15.

[12] Solar Policy Director at Vote Solar Annie Lappé.
Austin Energy's Value of Solar Tariff: Could It Work Anywhere Else?
`https://www.greentechmedia.com/articles/read/austin-energys-value-of-solar-tariff-could-it-work-anywhere-else`. Accessed: 2018-1-12. Mar. 2013.

[13] *Annual Energy Outlook 2020*. Tech. rep.
U.S. Energy Information Administration, Jan. 2020.

[14] Andreas M Antonopoulos. *Mastering Bitcoin: Unlocking Digital Cryptocurrencies*. en.
"O'Reilly Media, Inc.", Dec. 2014.

[15] *Apache CouchDB*. `http://couchdb.apache.org`.

[16] *Apache Kafka*. `http://kafka.apache.org`.

[17] *Applied Energy - Journal - Elsevier*.
`https://www.journals.elsevier.com/applied-energy`. Accessed: 2020-6-5.

[18] *APS Solar Partner Program: June 2016 Update*. Tech. rep. 3002008977. EPRI, July 2016.

[19] Omid Ardakanian, S Keshav, and Catherine Rosenberg.
"On the Use of Teletraffic Theory in Power Distribution Systems."
In: *Proceedings of the 3rd International Conference on Future Energy Systems: Where Energy, Computing and Communication Meet*. e-Energy '12. Madrid, Spain: ACM, 2012, 21:1–21:10.

[20] Pierre-Louis Aublin et al. "The next 700 BFT protocols."
In: *ACM Trans. Comput. Syst.* 32.4 (2015), p. 12.

[21] Peter Bailis and Kyle Kingsbury. "The network is reliable."
In: *Commun. ACM* 57.9 (Sept. 2014), pp. 48–55.

[22] Galen Barbose et al. *Tracking the Sun: Pricing and Design Trends for Distributed Photovoltaic Systems in the United States 2019 Edition*. Tech. rep.
Lawrence Berkeley National Laboratory, Oct. 2019.

[23] J Benet. *ipfs-cluster - tool to coordinate between nodes · Issue #58 · ipfs/notes*.
`https://github.com/ipfs/notes/issues/58`. Accessed: 2019-2-10. Sept. 2015.

[24] J Benet. *Replication on IPFS – Or, the Backing-Up Content Model · Issue #47 · ipfs/faq*.
`https://github.com/ipfs/faq/issues/47`. Accessed: 2019-2-10. Sept. 2015.

[25] Juan Benet. "IPFS - Content Addressed, Versioned, P2P File System." In: (July 2014).
arXiv: `1407.3561 [cs.NI]`.

[26] Dimitri P Bertsekas, Robert G Gallager, and Pierre Humblet. *Data networks*. Vol. 2.
Prentice-Hall International New Jersey, 1992.

[27] A Bessani, J Sousa, and E E P Alchieri.
"State Machine Replication for the Masses with BFT-SMART." In: *2014 44th Annual*

IEEE/IFIP International Conference on Dependable Systems and Networks. ieeexplore.ieee.org, June 2014, pp. 355–362.

[28] Alysson Bessani, João Sousa, and Marko Vukolić. "A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform." In: *Proceedings of the 1st Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*. 2017, p. 2.

[29] *Bitcoin - Open source P2P money*. `http://bitcoin.org`. Accessed: 2020-1-5.

[30] *Blockchain 2.0: Choosing a platform for peer-to-peer energy trading*. `https://www.engerati.com/energy-retail/article/blockchain/blockchain-20-choosing-platform-peer-peer-energy-trading`. Accessed: 2017-11-3. Nov. 2017.

[31] F Blom and H Farahmand. "On the Scalability of Blockchain-Supported Local Energy Markets." In: *2018 International Conference on Smart Energy Systems and Technologies (SEST)*. ieeexplore.ieee.org, 2018, pp. 1–6.

[32] Gabriel Bracha and Sam Toueg. "Asynchronous consensus and broadcast protocols." In: *J. ACM* 32.4 (1985), pp. 824–840.

[33] Gwern Branwen. *Time-lock encryption - Gwern.net*. `http://www.gwern.net/Self-decrypting-files`. Accessed: 2018-12-30. May 2011.

[34] Paul Brody and Veena Pureswaran. *Device democracy: Saving the future of the Internet of Things*. Tech. rep. IBM Institute for Business Value, Sept. 2014.

[35] Peter Bronski et al. *The Economics of Grid Defection*. Tech. rep. Rocky Mountain Institute, Feb. 2014.

[36] K Brousmichc et al. "Blockchain Energy Market Place Evaluation: An Agent-Based Approach." In: *2018 IEEE 9th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. ieeexplore.ieee.org, Nov. 2018, pp. 321–327.

[37] Ethan Buchman. "Tendermint: Byzantine Fault Tolerance in the Age of Blockchains." PhD thesis. University of Guelph, May 2016.

[38] Navin Budhiraja et al. "The primary-backup approach." In: *Distrib. Sys. Eng.* 2 (1993), pp. 199–216.

[39] Vitalik Buterin. *"Another category of use cases is verifying integrity of processes. For example, in an auction, you might want to verify that everyone's bid that was submitted on time was included, and no late bids were included. If bids are published to chain, even encrypted, you can do this."*

https://twitter.com/vitalikbuterin/status/1072161050550710272. Accessed: 2020-2-10. Dec. 2018.

[40] Vitalik Buterin. *On Stake*. https://blog.ethereum.org/2014/07/05/stake/. Accessed: 2019-1-4. July 2014.

[41] Christian Cachin. "Architecture of the Hyperledger blockchain fabric." In: *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*. 2016.

[42] Christian Cachin, Rachid Guerraoui, and Luís Rodrigues. *Introduction to Reliable and Secure Distributed Programming*. en. Springer Science & Business Media, Feb. 2011.

[43] Christian Cachin, Simon Schubert, and Marko Vukolić. "Non-determinism in Byzantine Fault-Tolerant Replication." In: (Mar. 2016). arXiv: 1603.07351 [cs.DC].

[44] Christian Cachin and Marko Vukolić. "Blockchain Consensus Protocols in the Wild." In: (July 2017). arXiv: 1707.01873 [cs.DC].

[45] Christian Cachin and Marko Vukolić. "Blockchain Consensus Protocols in the Wild." In: *31 International Symposium on Distributed Computing*. Vienna, Austria: drops.dagstuhl.de, 2017.

[46] Jan Camenisch and Els Van Herreweghen. "Design and implementation of the idemix anonymous credential system." In: *Proceedings of the 9th ACM conference on Computer and communications security*. 2002, pp. 21–30.

[47] S Caron and G Kesidis. "Incentive-Based Energy Consumption Scheduling Algorithms for the Smart Grid." In: *2010 First IEEE International Conference on Smart Grid Communications*. ieeexplore.ieee.org, Oct. 2010, pp. 391–396.

[48] *CASE 14-M-0101 - Proceeding on Motion of the Commission in Regard to Reforming the Energy Vision: Notice of Technical Conference Regarding Earnings Impact Mechanisms, Market Based Earnings, Standby Rates and Related Issues*. Tech. rep. Department of Public Service, State of New York, Jan. 2016.

[49] *CASE 14-M-0101 - Proceeding on Motion of the Commission in Regard to Reforming the Energy Vision: Staff White Paper on Ratemaking and Utility Business Models*. Tech. rep. Department of Public Service, State of New York, July 2015.

[50] Michael del Castillo. "Blockchain 50: Billion Dollar Babies." In: *Forbes Magazine* (Apr. 2019).

[51] Miguel Castro and Barbara Liskov.
"Practical Byzantine Fault Tolerance and Proactive Recovery."
In: *ACM Trans. Comput. Syst.* 20.4 (Nov. 2002), pp. 398–461.

[52] Miguel Castro, Barbara Liskov, et al. "Practical Byzantine fault tolerance." In: *OSDI*. Vol. 99. 1999, pp. 173–186.

[53] ChainSafe Systems. *ethermint*. June 2018.

[54] Bernadette Charron-Bost, Fernando Pedone, and Andre Schiper.
Replication: Theory and Practice. en. Vol. 5959. Lecture Notes in Computer Science.
Springer Science & Business Media, Mar. 2010.

[55] K Christidis and M Devetsikiotis.
"Adaptive multi-tiered resource allocation policy for microgrids."
In: *AIMS Energy* 4.2 (2016), pp. 300–312.

[56] K Christidis and M Devetsikiotis.
"Blockchains and Smart Contracts for the Internet of Things."
In: *IEEE Access* 4 (2016), pp. 2292–2303.

[57] Konstantinos Christidis. *cmap*. `http://dx.doi.org/10.5281/zenodo.3871065`.
Dec. 2018.

[58] Konstantinos Christidis. *island-input*. `http://dx.doi.org/10.5281/zenodo.3871075`.
Accessed: 2019-9-14. Dec. 2017.

[59] Konstantinos Christidis. *overlap*. `https://doi.org/10.5281/zenodo.3871070`.
Accessed: 2019-9-13. Dec. 2017.

[60] Konstantinos Christidis and Yun Wang. *island*.
`http://dx.doi.org/10.5281/zenodo.3871077`. Accessed: 2020-2-10. Oct. 2018.

[61] Kostas Christidis. *dauction*. `http://dx.doi.org/10.5281/zenodo.3871067`.
Accessed: 2020-2-18. Mar. 2018.

[62] *cicero*. `https://github.com/accordproject/cicero`. Accessed: 2019-2-24.

[63] Christopher Copeland and Hongxia Zhong.
"Tangaroa: a Byzantine Fault Tolerant Raft." 2014.

[64] Kyle Croman et al. "On Scaling Decentralized Blockchains." In: (Feb. 2016).

[65] Jerry Cuomo. *A Case for Permissioned Blockchains*. en.
`https://www.ibm.com/developerworks/community/blogs/gcuomo/entry/A_Case_for_Permissioned_Blockchains?lang=en`. Accessed: 2019-2-9. Apr. 2016.

[66] *Dataport from Pecan Street*. https://dataport.cloud/. Accessed: 2017-10-13. 2017.

[67] Evangelos Deirmentzoglou. *Rewriting History: A Brief Introduction to Long Range Attacks*. https://blog.positive.com/rewriting-history-a-brief-introduction-to-long-range-attacks-54e473acdba9. Accessed: 2019-1-4. May 2018.

[68] Alan Demers et al. "Epidemic algorithms for replicated database maintenance." In: *Oper. Syst. Rev.* 22.1 (1988), pp. 8–32.

[69] *Developing the USN - Universal Sharing Network*. https://slock.it/usn.html. Accessed: 2019-2-13.

[70] *DigitalOcean – The developer cloud*. https://www.digitalocean.com/. Accessed: 2020-5-13. 2020.

[71] Tien Tuan Anh Dinh et al. "BLOCKBENCH: A Framework for Analyzing Private Blockchains." In: *Proceedings of the 2017 ACM International Conference on Management of Data*. 2017, pp. 1085–1100.

[72] Tien Tuan Anh Dinh et al. "Untangling Blockchain: A Data Processing View of Blockchain Systems." In: (Aug. 2017). arXiv: 1708.05665 [cs.DB].

[73] John R Douceur. "The Sybil Attack." In: *Peer-to-Peer Systems*. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Mar. 2002, pp. 251–260.

[74] *Dual Integration*. https://monax.io/learn/dual_integration/. Accessed: 2019-2-24.

[75] Quinn DuPont. "Experiments in algorithmic governance: A history and ethnography of "The DAO," a failed decentralized autonomous organization." In: *Bitcoin and Beyond*. Routledge, 2017, pp. 157–177.

[76] Cynthia Dwork and Moni Naor. "Pricing via Processing or Combatting Junk Mail." In: *Advances in Cryptology — CRYPTO' 92*. Springer Berlin Heidelberg, 1993, pp. 139–147.

[77] *Electricity Data Browser: Average retail price of electricity, monthly (Texas, Residential)*. https://www.eia.gov/electricity/data/browser/#/topic/7?agg=0,1&geo=0000000002&endsec=8&linechart=ELEC.PRICE.TX-RES.M&columnchart=ELEC.PRICE.TX-RES.M&map=ELEC.PRICE.TX-RES.M&freq=M&start=201210&end=201411&chartindexed=0&ctype=linechart&ltype=pin&rtype=s&maptype=0&rse=0&pin=. Accessed: 2019-9-13. 2019.

[78] Electronic Frontier Foundation.
A Deep Dive on End-to-End Encryption: How Do Public Key Encryption Systems Work?
`https://ssd.eff.org/en/module/deep-dive-end-end-encryption-how-do-public-key-encryption-systems-work`. Accessed: 2018-12-27. Sept. 2014.

[79] *Enterprise Blockchain Solutions*. `https://filament.com/`. Accessed: 2019-2-14.

[80] ERCOT. *ERCOT Market Education: Transmission 101*. 2013.

[81] *Ethereum*. `http://ethereum.org`. Accessed: 2020-1-5.

[82] Ittay Eyal and Emin Gün Sirer.
"Majority Is Not Enough: Bitcoin Mining Is Vulnerable."
In: *Financial Cryptography and Data Security*. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Mar. 2014, pp. 436–454.

[83] Ittay Eyal et al. "Bitcoin-NG: A Scalable Blockchain Protocol." In: (Oct. 2015). arXiv: `1510.02037 [cs.CR]`.

[84] Jim Eyer and Garth Corey.
Energy storage for the electricity grid: Benefits and market potential assessment guide.
Tech. rep. Sandia National Laboratories, 2010, p. 5.

[85] *Fabric*. `https://github.com/hyperledger/fabric`. Accessed: 2020-1-6.

[86] *Fact Sheet: DOE Award Selections for the Development of Next Generation Cybersecurity Technologies and Tools*. Sept. 2017.

[87] Murat Fahrioglu and Fernando L Alvarado.
"Designing incentive compatible contracts for effective demand management."
In: *IEEE Trans. Power Syst.* 15.4 (2000), pp. 1255–1260.

[88] *Feed-in tariff: A policy tool encouraging deployment of renewable electricity technologies - Today in Energy*. `https://www.eia.gov/todayinenergy/detail.php?id=11471`. Accessed: 2018-1-12. May 2013.

[89] *Feed-In Tariffs and similar programs*.
`https://www.eia.gov/electricity/policies/provider_programs.php`.
Accessed: 2018-1-12. June 2013.

[90] Helder Lopes Ferreira et al.
"Characterisation of electrical energy storage technologies."
In: *Energy* 53 (May 2013), pp. 288–298.

[91] R Fielding et al. *HTTP/1.1: Status Code Definitions*. June 1999.

[92] Lester Randolph Ford and Delbert R Fulkerson. "Maximal flow through a network." In: *Classic papers in combinatorics*. Springer, 2009, pp. 243–248.

[93] Paul Ford. "Bitcoin Is Ridiculous. Blockchain Is Dangerous." In: *Bloomberg News* (Mar. 2018).

[94] M Foti, D Greasidis, and M Vavalis. "Viability Analysis of a Decentralized Energy Market Based on Blockchain." In: *2018 15th International Conference on the European Energy Market (EEM)*. ieeexplore.ieee.org, June 2018, pp. 1–5.

[95] Magda Foti and Manolis Vavalis. "Blockchain Based Uniform Price Double Auctions for Energy Markets." In: *Applied Energy* 254 (2019).

[96] *Fujitsu Cloud Service Adopted by Japanese Bankers Association for Blockchain-based Financial Service Testbed - Fujitsu Global*. `http://www.fujitsu.com/global/about/resources/news/press-releases/2017/0914-01.html`. Accessed: 2020-1-10.

[97] Rui Garcia, Rodrigo Rodrigues, and Nuno Preguiça. "Efficient middleware for byzantine fault tolerant database replication." In: *Proceedings of the sixth conference on Computer systems*. 2011, pp. 107–122.

[98] Y Ghiassi-Farrokhfal, S Keshav, and C Rosenberg. "Toward a Realistic Performance Analysis of Storage Systems in Smart Grids." In: *IEEE Trans. Smart Grid* 6.1 (Jan. 2015), pp. 402–410.

[99] Dhananjay K Gode and Shyam Sunder. "Allocative Efficiency of Markets with Zero-Intelligence Traders: Market as a Partial Substitute for Individual Rationality." In: *J. Polit. Econ.* 101.1 (1993), pp. 119–137.

[100] Logan Goldie-Scot. *A Behind the Scenes Take on Lithium-ion Battery Prices*. `https://about.bnef.com/blog/behind-scenes-take-lithium-ion-battery-prices/`. Accessed: 2020-1-25. Mar. 2019.

[101] H Goudarzi and M Pedram. "Multi-dimensional SLA-Based Resource Allocation for Multi-tier Cloud Computing Systems." In: *2011 IEEE 4th International Conference on Cloud Computing*. ieeexplore.ieee.org, July 2011, pp. 324–331.

[102] Government Office for Science. *Distributed ledger technology: Blackett review*. Tech. rep. Jan. 2016.

[103] J Granata et al. *The First Micropayments Marketplace*. https://medium.com/@earndotcom/the-first-micropayments-marketplace-38c321127d12. Accessed: 2019-2-13. Mar. 2016.

[104] G Greenspan. *MultiChain permissions management*. https://www.multichain.com/developers/permissions-management/. Accessed: 2019-2-10.

[105] G Greenspan. *Smart contracts: The good, the bad and the lazy*. http://www.multichain.com/blog/2015/11/smart-contracts-good-bad-lazy/. Accessed: 2018-2-10. Nov. 2015.

[106] Gideon Greenspan. *Avoiding the pointless blockchain project*. http://www.multichain.com/blog/2015/11/avoiding-pointless-blockchain-project/. Accessed: 2018-12-27. Nov. 2015.

[107] Gideon Greenspan. *Delivery versus payment on a blockchain*. https://www.multichain.com/blog/2015/09/delivery-versus-payment-blockchain/. Accessed: 2019-2-14. Sept. 2015.

[108] *GridPlus*. https://gridplus.io/energy. Accessed: 2019-9-12. 2019.

[109] *GridWise Transactive Energy Framework Version 1.0*. Tech. rep. The GridWise Architecture Council, Jan. 2015.

[110] Tom Groenfeldt. "IBM And Maersk Apply Blockchain To Container Shipping." In: *Forbes Magazine* (Mar. 2017).

[111] *gRPC*. http://grpc.io. Accessed: 2020-1-8.

[112] Rachid Guerraoui et al. "Throughput optimal total order broadcast for cluster environments." In: *ACM Trans. Comput. Syst.* 28.ARTICLE (2010), p. 5.

[113] Stuart Haber and W Scott Stornetta. "How to Time-Stamp a Digital Document." In: *Advances in Cryptology-CRYPTO' 90*. Springer Berlin Heidelberg, 1991, pp. 437–455.

[114] Hal Harvey and Sonia Aggarwal. *Rethinking Policy to Deliver a Clean Energy Future*. Tech. rep. Energy Innovation, Sept. 2013.

[115] E Henigin. *Why Does Austin Energy Lose Money on Wholesale Generation, Asks Data Foundry's Henigin*. http://www.texasinsider.org/austin-energy-wholesale-generation-losses-says-data-foundrys-henigin/. Accessed: 2018-1-13. Aug. 2016.

[116] A Hertig. *Ethereum Energy Project Now Powers 700 Households in 10 Cities*. https://www.coindesk.com/ethereum-energy-project-now-powers-700-households-in-10-cities/. Accessed: 2018-11-6. Nov. 2018.

[117] Stan Higgins. *How Bitcoin Brought Electricity to a South African School - CoinDesk*. http://www.coindesk.com/south-african-primary-school-blockchain/. Accessed: 2017-1-15. Mar. 2016.

[118] Anna Hirtenstein. *Green-Power Traders Sought in Utility's Bid to Emulate Airbnb*. https://www.bloomberg.com/news/articles/2016-06-14/green-power-traders-sought-in-utility-s-bid-to-emulate-airbnb. Accessed: 2017-1-15. June 2016.

[119] Anna Hirtenstein and Weixin Zha. *Bitcoin Technology Harnessed to Push Electricity Revolution*. https://www.bloomberg.com/news/articles/2016-09-12/bitcoin-technology-harnessed-to-push-electricity-revolution. Accessed: 2017-1-15. Sept. 2016.

[120] *Hyperledger – Open Source Blockchain Technologies*. http://www.hyperledger.org. Accessed: 2020-1-6.

[121] *IBM announces major blockchain solution to speed global payments*. en. https://www-03.ibm.com/press/us/en/pressrelease/53290.wss. Accessed: 2020-1-10. Oct. 2017.

[122] D Ilic et al. "An energy market for trading electricity in smart grid neighbourhoods." In: *2012 6th IEEE International Conference on Digital Ecosystems and Technologies (DEST)*. June 2012, pp. 1–6.

[123] B Inskeep et al. *The 50 States of Solar: 2015 Policy Review and Q4 Quarterly Report*. Tech. rep. North Carolina Clean Energy Technology Center & Meister Consultants Group, Feb. 2016.

[124] *Interstellar*. https://interstellar.com/. Accessed: 2020-1-6.

[125] *Introduction*. https://github.com/telehash/telehash.github.io/blob/master/v3/intro.md. Accessed: 2019-2-14. Apr. 2015.

[126] F P Junqueira, B C Reed, and M Serafini. "Zab: High-performance broadcast for primary-backup systems." In: *2011 IEEE/IFIP 41st International Conference on Dependable Systems Networks (DSN)*. June 2011, pp. 245–256.

[127] J Kang et al. "Enabling Localized Peer-to-Peer Electricity Trading Among Plug-in Hybrid Electric Vehicles Using Consortium Blockchains." In: *IEEE Trans. Ind. Inf.* PP.99 (2017), pp. 1–1.

[128] Manos Kapritsos et al.
"All about Eve: Execute-verify replication for multi-core servers." In: *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)*. 2012, pp. 237–250.

[129] Manos Kapritsos et al.
"All about Eve: execute-verify replication for multi-core servers."
In: *Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12)*. usenix.org, 2012, pp. 237–250.

[130] Ian Kar. *Estonian citizens will soon have the world's most hack-proof health-care records*. `https://qz.com/628889/this-eastern-european-country-is-moving-its-health-records-to-the-blockchain/`. Accessed: 2018-12-27. Mar. 2016.

[131] R Karp et al. "Randomized rumor spreading."
In: *Proceedings 41st Annual Symposium on Foundations of Computer Science*. Nov. 2000, pp. 565–574.

[132] Jemima Kelly. "Forty big banks test blockchain-based bond trading system."
In: *Reuters* (Mar. 2016).

[133] B Kemme and G Alonso. "A new approach to developing and implementing eager database replication protocols."
In: *ACM Transactions on Database Systems (TODS)* 25.3 (2000), pp. 333–379.

[134] B Kemme, R Jiménez-Peris, and M Patino Martınez. "Database Replication."
In: *Synthesis Lectures on Data Management* 2.1 (2010), pp. 1–153.

[135] Bettina Kemme. "One-Copy-Serializability." In: *Encyclopedia of Database Systems*. Ed. by Ling Liu and M Tamer Özsu. Boston, MA: Springer US, 2009, pp. 1947–1948.

[136] Ruud Kempener and Eric Borden.
Battery Storage for Renewables: Market Status and Technology Outlook. Tech. rep. International Renewable Energy Agency, Jan. 2015.

[137] L Lynne Kiesling. "Implications of Smart Grid Innovation for Organizational Models in Electricity Distribution." In: *Forthcoming, Wiley Handbook of Smart Grid Development*. Wiley, Feb. 2015.

[138] T T Kim and H V Poor. "Scheduling Power Consumption With Price Uncertainty."
In: *IEEE Trans. Smart Grid* 2.3 (Sept. 2011), pp. 519–527.

[139] Fabian Knirsch, Andreas Unterweger, and Dominik Engel. "Privacy-preserving blockchain-based electric vehicle charging with dynamic tariff decisions." en.
In: *Comput Sci Res Dev* (Sept. 2017), pp. 1–9.

[140] Ahmed Kosba et al.
Hawk: The blockchain model of cryptography and privacy-preserving smart contracts.
Tech. rep.
Cryptology ePrint Archive, Report 2015/675, 2015. http://eprint. iacr. org, 2015.

[141] I Kounelis et al. "Fostering consumers' energy market through smart contracts."
In: *2017 International Conference in Energy and Sustainability in Small Developing Economies (ES2DE)*. July 2017, pp. 1–6.

[142] Iordanis Koutsopoulos and Leandros Tassiulas.
"Optimal control policies for power demand scheduling in the smart grid."
In: *IEEE J. Sel. Areas Commun.* 30.6 (2012), pp. 1049–1060.

[143] Karl Kreder. *Introducing the Grid+ Lattice1*.
`https://blog.gridplus.io/introducing-the-grid-lattice1-bc4ff6df5321`.
Accessed: 2019-9-12. Aug. 2018.

[144] Karla Kvaternik et al. "Privacy-Preserving Platform for Transactive Energy Systems."
In: (Sept. 2017). arXiv: `1709.09597 [cs.DC]`.

[145] Jae Kwon and Ethan Buchman. *Tendermint*. `http://tendermint.com`.
Accessed: 2020-1-6. 2016.

[146] Stephen Lacey.
New York's Energy Czar: We Need Clean Energy Markets, Not Programs or Mandates.
`https://www.greentechmedia.com/articles/read/new-york-energy-czar-we-need-clean-energy-markets-not-programs`. Accessed: 2020-2-8. Nov. 2014.

[147] Stephen Lacey.
The Energy Blockchain: How Bitcoin Could Be a Catalyst for the Distributed Grid.
`https://www.greentechmedia.com/articles/read/the-energy-blockchain-could-bitcoin-be-a-catalyst-for-the-distributed-grid`. Accessed: 2018-12-27.
Feb. 2016.

[148] Leslie Lamport, Robert Shostak, and Marshall Pease.
"The Byzantine Generals Problem."
In: *ACM Trans. Program. Lang. Syst.* 4.3 (July 1982), pp. 382–401.

[149] *Lesson Learned: Risks Posed by Firewall Firmware Vulnerabilities*. Tech. rep.
North American Electric Reliability Corporation, Sept. 2019.

[150] Antony Lewis. *What you get for free with blockchains – Bits on Blocks*. `https://bitsonblocks.net/2016/04/04/what-you-get-for-free-with-blockchains/`.
Accessed: 2019-1-5. Apr. 2016.

[151] Orfeas Stefanos Thyfronitis Litos and Dionysis Zindros.
"Trust Is Risk: A Decentralized Financial Trust Platform." In:
Malta: eprint.iacr.org, 2017.

[152] Shengyun Liu et al. "{XFT}: Practical Fault Tolerance beyond Crashes." In: *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 2016, pp. 485–500.

[153] *LRB · John Lanchester · When Bitcoin Grows Up: What is Money?* Apr. 2016.

[154] *"Machine Payable Web" - Balaji Srinivasan*. June 2016.

[155] Posted By: Nichols Martin.
DOE Taps Guardtime, Siemens to Help Develop Blockchain Tech for Energy Grid Security.
`http://blog.executivebiz.com/2017/09/doe-taps-guardtime-siemens-to-help-develop-blockchain-tech-for-energy-grid-security/`. Accessed: 2017-10-18.
Sept. 2017.

[156] Scott W McCollough and Matthew A Henry.
Austin Energy's Tariff Package: 2015 Cost of Service Study and Proposal to Change Base Electric Rates: Data Foundry, Inc.'s Corrected Presentation on Revenue Requirements.
May 2016.

[157] Patric McCorry. *I fell in love with smart contracts not because of "dapps", but their potential to simplify protocol design..*
`https://twitter.com/paddykcl/status/1079677330816151554`. Accessed: 2019-1-5.
Dec. 2018.

[158] Stephen McKeon. *Traditional Asset Tokenization*.
`https://hackernoon.com/traditional-asset-tokenization-b8a59585a7e0`.
Accessed: 2017-10-8. Jan. 2019.

[159] Sarah Meiklejohn et al.
"A fistful of bitcoins: characterizing payments among men with no names."
In: *Proceedings of the 2013 conference on Internet measurement conference*. ACM, Oct. 2013, pp. 127–140.

[160] Esther Mengelkamp, Johannes Gärttner, and Christof Weinhardt.
Decentralizing Energy Systems Through Local Energy Markets: The LAMP-Project. Tech. rep.
Mar. 2018.

[161] Esther Mengelkamp et al.
"A blockchain-based smart grid: towards sustainable local energy markets." en.
In: *Comput Sci Res Dev* (Aug. 2017), pp. 1–8.

[162] Esther Mengelkamp et al.
"Designing microgrid energy markets: A case study: The Brooklyn Microgrid."
In: *Appl. Energy* (June 2017).

[163] Alex Miller, Karl Kreder, and Mark DAgostino. *Grid+ - Welcome to the Future of Energy*. Tech. rep. GridPlus, 2017.

[164] Alex Mizrahi. *A blockchain-based property ownership recording system*. 2015.

[165] *MultiChain*. `http://www.multichain.com/`. Accessed: 2020-1-11.

[166] Satoshi Nakamoto. *Bitcoin: A peer-to-peer electronic cash system*. 2008.

[167] Arvind Narayanan and Jeremy Clark. "Bitcoin's Academic Pedigree."
In: *Queueing Syst.* 15.4 (Aug. 2017), p. 20.

[168] James Newcomb, Virginia Lacy, and Lena Hansen.
New Business Models for the Distribution Edge. Tech. rep.
Rocky Mountain Institute, Apr. 2013.

[169] Sana Noor et al. "Energy Demand Side Management within micro-grid networks enhanced by blockchain." In: *Appl. Energy* 228 (Oct. 2018), pp. 1385–1398.

[170] Diego Ongaro and John Ousterhout.
"In search of an understandable consensus algorithm."
In: *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*. usenix.org, 2014, pp. 305–319.

[171] Don Oparah. *3 Ways That The Blockchain Will Change The Real Estate Market*.
`http://social.techcrunch.com/2016/02/06/3-ways-that-blockchain-will-change-the-real-estate-market/`. Accessed: 2018-12-27. Feb. 2016.

[172] Torben Pryds Pedersen.
"Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing."
In: *Advances in Cryptology — CRYPTO '91*. Springer Berlin Heidelberg, 1992, pp. 129–140.

[173] F Pedone and A Schiper.
"Handling message semantics with Generic Broadcast protocols."
In: *Distrib. Comput.* 15.2 (Apr. 2002), pp. 97–107.

[174] Austin Perea et al. *U.S. Solar Market Insight*. Tech. rep.
Wood Mackenzie Power & Renewables; Solar Energy Industries Association, Dec. 2019.

[175] *Performance Modeling of Hyperledger Sawtooth Blockchain*.
`http://sawtooth.hyperledger.org`. Accessed: 2020-1-11.

[176] Becky Peterson. "IBM told investors that it has over 400 blockchain clients — including Walmart, Visa, and Nestlé." In: *Business Insider* (Mar. 2018).

[177] *Private data*. https://hyperledger-fabric.readthedocs.io/en/release-1.4/private-data/private-data.html. Accessed: 2019-2-15.

[178] *Proof of Existence*. https://www.proofofexistence.com/. Accessed: 2019-2-24.

[179] *Proof of Stake FAQs*. https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQs. Accessed: 2019-1-4.

[180] Protocol Labs. *Filecoin: A Decentralized Storage Network*. Tech. rep. July 2017.

[181] *Public Utilities Code Section 748 Report to the Governor and Legislature on Actions to Limit Utility Cost and Rate Increases*. Tech. rep.
California Public Utilities Commission, June 2013.

[182] Julia Pyper.
Smart Inverters in Action: Initial Findings From APS' Utility-Owned Solar Program.
https://www.greentechmedia.com/articles/read/Smart-Inverters-in-Action-Initial-Findings-From-APS-Utility-Owned. Accessed: 2020-2-20. Aug. 2016.

[183] *Quorum*. http://www.jpmorgan.com/global/Quorum. Accessed: 2020-1-6.

[184] *Quorum Whitepaper*. https://github.com/jpmorganchase/quorum-docs. Accessed: 2020-1-9.

[185] Ian Richardson et al.
"Domestic electricity use: A high-resolution energy demand model."
In: *Energy Build.* 42.10 (Oct. 2010), pp. 1878–1887.

[186] R L Rivest, A Shamir, and D A Wagner. *Time-lock Puzzles and Timed-release Crypto*. Tech. rep. Cambridge, MA, USA, 1996.

[187] David Roberts. *New York's revolutionary plan to remake its power utilities*. https://www.vox.com/2015/10/5/9453131/new-york-utilities-rev. Accessed: 2020-2-8. Oct. 2015.

[188] David Roberts. *Rooftop solar is just the beginning; utilities must innovate or go extinct*. https://grist.org/climate-energy/rooftop-solar-is-just-the-beginning-utilities-must-innovate-or-go-extinct/. Accessed: 2020-2-8. Oct. 2014.

[189] Dorit Ron and Adi Shamir.
"Quantitative Analysis of the Full Bitcoin Transaction Graph."

In: *Financial Cryptography and Data Security*. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Apr. 2013, pp. 6–24.

[190] Aviva Rutkin. "Blockchain-based microgrid gives power to consumers in New York." In: *New Scientist* (Mar. 2016).

[191] Ken Sakakibara. *Blockchain-manged energy grid to be tested in Fukushima- Nikkei Asian Review*. `https://asia.nikkei.com/Business/Trends/Blockchain-manged-energy-grid-to-be-tested-in-Fukushima`. Accessed: 2017-10-8. Sept. 2017.

[192] P Samadi et al. "Optimal Real-Time Pricing Algorithm Based on Utility Maximization for Smart Grid." In: *2010 First IEEE International Conference on Smart Grid Communications*. ieeexplore.ieee.org, Oct. 2010, pp. 415–420.

[193] E B Sasson et al. "Zerocash: Decentralized Anonymous Payments from Bitcoin." In: *2014 IEEE Symposium on Security and Privacy*. May 2014, pp. 459–474.

[194] Fred B Schneider. "Implementing Fault-tolerant Services Using the State Machine Approach: A Tutorial." In: *ACM Comput. Surv.* 22.4 (Dec. 1990), pp. 299–319.

[195] *Script - Bitcoin Wiki*. `https://en.bitcoin.it/wiki/Script`. Accessed: 2019-2-10. May 2018.

[196] Joy Sengupta, Ram Komarraju, and Keith Bear. *Bridging the divide: How CLS and IBM moved to blockchain*. `https://www.ibm.com/thought-leadership/institute-business-value/report/bridgingdivide/`. Accessed: 2020-1-10.

[197] Srinath Setty et al. *Enabling secure and resource-efficient blockchain networks with VOLT*. Tech. rep. Technical Report MSR-TR-2017-38. Microsoft Research, 2017.

[198] Marc Shapiro et al. "Conflict-Free Replicated Data Types." In: *Stabilization, Safety, and Security of Distributed Systems*. Springer Berlin Heidelberg, 2011, pp. 386–400.

[199] Robby Simpson. *IEEE 2030.5-2014 (Smart Energy Profile 2.0) - An Oveview for KSGA*. Apr. 2015.

[200] Atul Singh et al. "BFT Protocols Under Fire." In: *NSDI*. Vol. 8. 2008, pp. 189–204.

[201] Emin Gün Sirer. *There are many consensus protocols (mechanisms), and there are many Sybil control mechanisms. These two kinds of mechanisms are distinct..*

https://twitter.com/el33th4xor/status/1006931901221933056.
Accessed: 2019-1-3. June 2018.

[202] *SolarEdge Cellular Kit*. http:
//www.solaredge.com/us/products/communication/solaredge-cellular-kit/.
Accessed: 2017-1-15.

[203] Saleh Soltan, Prateek Mittal, and H Vincent Poor.
"BlackIoT: IoT botnet of high wattage devices can disrupt the power grid."
In: *27th USENIX Security Symposium (USENIX Security 18)*.
Baltimore, MD: {USENIX} Association, Aug. 2018, pp. 15–32.

[204] S Somasundaram et al.
Reference Guide for a Transaction-Based Building Controls Framework. Tech. rep.
Pacific Northwest National Laboratory, Apr. 2014.

[205] Joao Sousa, Alysson Bessani, and Marko Vukolic. "A Byzantine Fault-Tolerant Ordering Service for the Hyperledger Fabric Blockchain Platform."
In: *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (2018).

[206] Jeff St. John. *An Inside Look at a Groundbreaking Solar-Storage Procurement in California*.
https://www.greentechmedia.com/articles/read/inside-california-community-energy-providers-groundbreaking-solar-storage-p.
Accessed: 2020-1-19. Nov. 2019.

[207] Jeff St. John.
US Storage Market Rebounds as Outage-Scarred California Promises Big 2020 Growth.
https://www.greentechmedia.com/articles/read/us-storage-market-rebounds-in-q3-as-calif-power-outages-loom-large. Accessed: 2020-1-19. Dec. 2019.

[208] *State of the Electric Utility 2015 - Survey Results*. Tech. rep. Utility Dive, Jan. 2015.

[209] Stem, Inc. *Demand Charges 101 - How to lower peak electricity costs*.
https://www.stem.com/resources/demandcharges/. Accessed: 2018-12-22.

[210] William Suberg. *Factom's Latest Partnership Takes on US Healthcare*.
https://cointelegraph.com/news/factoms-latest-partnership-takes-on-us-healthcare. Accessed: 2018-12-27. Apr. 2015.

[211] H Sukhwani et al. "Performance Modeling of PBFT Consensus Process for Permissioned Blockchain Network (Hyperledger Fabric)."
In: *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*.
ieeexplore.ieee.org, Sept. 2017, pp. 253–255.

[212] *SunSpec Midyear Update Open Industry Meeting at Intersolar*. Intersolar. InterContinental Hotel, San Francisco, July 2016.

[213] Andrew S Tanenbaum. "Distributed operating systems anno 1992. What have we learned so far?" In: *Distrib. Sys. Eng.* 1.1 (1993), p. 3.

[214] Luke Tarbi. *How Does Austin Energy Net Metering Work in 2017? — EnergySage*. `https://news.energysage.com/austins-unique-policy-for-counting-solar-power/`. Accessed: 2018-1-12. May 2017.

[215] *The Linux Foundation – Supporting Open Source Ecosystems*. `http://www.linuxfoundation.org`. Accessed: 2020-1-6.

[216] *Time-of-Use rate plans*. `https://www.pge.com/en_US/small-medium-business/your-account/rates-and-rate-options/time-of-use-rates.page`. Accessed: 2018-12-22.

[217] Peter Todd. *Timelock: time-release encryption incentivised by Bitcoins*. `https://www.mail-archive.com/bitcoin-development@lists.sourceforge.net/msg05547.html`. Accessed: 2018-12-30. June 2014.

[218] *Transactive Energy Application Landscape Scenarios*. Tech. rep. SGIP, Dec. 2016.

[219] Suryandaru Triandana. *LevelDB key/value database in Go*. `https://github.com/syndtr/goleveldb`. Accessed: 2020-1-9.

[220] Florian Tschorsch and Björn Scheuermann. "Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies." In: *IACR Cryptology ePrint Archive* 2015 (2015), p. 464.

[221] Katherine Tweed. *New York Launches Major Regulatory Reform for Utilities*. `https://www.greentechmedia.com/articles/read/new-york-launches-major-regulatory-reform-for-utilities`. Accessed: 2020-2-8. Apr. 2014.

[222] *U.S. Energy Storage Monitor Q4 2019*. Tech. rep. Wood Mackenzie Power & Renewables; ESA U.S. Energy Storage Monitor, Dec. 2019.

[223] Ben Vandiver et al. "Tolerating byzantine faults in transaction processing systems using commit barrier scheduling." In: *ACM SIGOPS Operating Systems Review*. Vol. 41. 2007, pp. 59–72.

[224] M Venkataraman et al. *Adopting blockchain for enterprise asset management (EAM)*. `https://developer.ibm.com/tutorials/cl-adopting-blockchain-for-enterprise-asset-management-eam/`. Accessed: 2020-1-10. Mar. 2017.

[225] Marko Vukolić.
"The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication."
In: *Open Problems in Network Security*. Springer International Publishing, 2016, pp. 112–125.

[226] *What's "normal" for latency and packet loss?* `https://www.pingman.com/kb/42`. Accessed: 2017-1-15. Dec. 2014.

[227] *Whisper*. `https://github.com/ethereum/wiki/wiki/Whisper`. Accessed: 2019-2-13. Aug. 2018.

[228] Gavin Wood. *Ethereum: A secure decentralised generalised transaction ledger*. 2014.

[229] J Wu et al. "Distributed Optimal Dispatch of Distributed Energy Resources Over Lossy Communication Networks." In: *IEEE Trans. Smart Grid* 8.6 (Nov. 2017), pp. 3125–3137.

[230] Pieter Wuille. *BIP:32 - Hierarchical Deterministic Wallets*. Feb. 2012.

[231] Yuzhe Xu. "Latency and Bandwidth Analysis of LTE for a Smart Grid."
MA thesis. KTH Royal Institute of Technology, 2011.

[232] Jian Yin et al.
"Separating agreement from execution for byzantine fault tolerant services."
In: *ACM SIGOPS Operating Systems Review*. Vol. 37. 2003, pp. 253–267.

[233] Dionysis Zindros. *Trust is Risk: A Decentralized Trust System*. `https://www.openbazaar.org/blog/trust-is-risk-a-decentralized-trust-system/`. Accessed: 2017-10-9. Aug. 2017.

Appendices

Appendix A

Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains

An edited version of this chapter was originally published in [11]. This is a joint work with IBM and IBM Research. K. Christidis was supported by an IBM PhD Fellowship award during part of this work. His main contribution is on the ordering service component (Section A.4.2). K. Christidis has also served as a core maintainer of the Hyperledger Fabric open source project [85].

A.1 Introduction

A blockchain can be defined as an immutable *ledger* for recording *transactions*, maintained within a distributed network of mutually untrusting *peers*. Every peer maintains a copy of the ledger. The peers execute a *consensus protocol* to validate transactions, group them into blocks, and build a hash chain over the blocks. This process forms the ledger by ordering the transactions, as is necessary for consistency. Blockchains have emerged with Bitcoin [29] and

are widely regarded as a promising technology to run trusted exchanges in the digital world.

In a *public* or *permissionless* blockchain anyone can participate without a specific identity. Public blockchains typically involve a native cryptocurrency and often use consensus based on "proof of work" (PoW) and economic incentives. *Permissioned* blockchains, on the other hand, run a blockchain among a set of known, identified participants. A permissioned blockchain provides a way to secure the interactions among a group of entities that have a common goal but which do not fully trust each other, such as businesses that exchange funds, goods, or information. By relying on the identities of the peers, a permissioned blockchain can use traditional Byzantine-fault tolerant (BFT) consensus.

Blockchains may execute arbitrary, programmable transaction logic in the form of *smart contracts*, as exemplified by Ethereum [81]. The scripts in Bitcoin were a predecessor of the concept. A smart contract functions as a *trusted distributed application* and gains its security from the blockchain and the underlying consensus among the peers. This closely resembles the well-known approach of building resilient applications with state-machine replication (SMR) [194]. However, blockchains depart from traditional SMR with Byzantine faults in important ways: (1) not only one, but many distributed applications run concurrently; (2) applications may be deployed dynamically and by anyone; and (3) the application code is untrusted, potentially even malicious. These differences necessitate new designs.

Many existing smart-contract blockchains follow the blueprint of SMR [194] and implement so-called *active* replication [54]: a protocol for *consensus* or *atomic broadcast* first orders the transactions and propagates them to all peers; and second, each peer executes the transactions sequentially. We call this the *order-execute architecture*; it requires all peers to execute every transaction and all transactions to be deterministic. The order-execute architecture can be found in virtually all existing blockchain systems, ranging from public ones such as Ethereum (with PoW-based consensus) to permissioned ones (with BFT-type consensus) such as Tendermint [145], Interstellar [124] and Quorum [183]. Although the order-execute design is not immediately apparent in all systems, because the additional transaction validation step may

blur it, its limitations are inherent in all: every peer executes every transaction and transactions must be deterministic.

Prior permissioned blockchains suffer from many limitations, which often stem from their permissionless relatives or from using the order-execute architecture. In particular:

- Consensus is *hard-coded* within the platform, which contradicts the well-established understanding that there is no "one-size-fits-all" (BFT) consensus protocol [200];
- The *trust model* of transaction validation is determined by the consensus protocol and cannot be adapted to the requirements of the smart contract;
- Smart contracts must be written in a *fixed, non-standard, or domain-specific language*, which hinders wide-spread adoption and may lead to programming errors;
- The *sequential execution* of all transactions by all peers *limits performance*, and complex measures are needed to prevent denial-of-service attacks against the platform originating from untrusted contracts (such as accounting for runtime with "gas" in Ethereum);
- Transactions must be *deterministic*, which can be difficult to ensure programmatically;
- Every smart contract runs on *all* peers, which is at odds with *confidentiality*, and prohibits the dissemination of contract code and state to a subset of peers.

In this chapter we describe *Hyperledger Fabric* or simply *Fabric*, an open-source [85] blockchain platform that overcomes these limitations. Fabric is one of the projects of Hyperledger [120] under the auspices of the Linux Foundation [215]. Fabric is used in more than 400 prototypes, proofs-of-concept, and in production distributed-ledger systems, across different industries and use cases. These use cases include but are not limited to areas such as dispute resolution, trade logistics, FX netting, food safety, contract management, diamond provenance, rewards point management, low liquidity securities trading and settlement, identity management, and settlement through digital currency.

Fabric introduces a new blockchain architecture aiming at resiliency, flexibility, scalability, and confidentiality. Designed as a modular and extensible general-purpose permissioned blockchain, Fabric is the first blockchain system to support the execution of distributed applications written in *standard programming languages*, in a way that allows them to be executed consistently across many nodes, giving impression of execution on a single globally-distributed blockchain computer. This makes Fabric the first *distributed operating system* [213] for permissioned blockchains.

The architecture of Fabric follows a novel *execute-order-validate* paradigm for distributed execution of untrusted code in an untrusted environment. It separates the transaction flow into three steps, which may be run on different entities in the system: (1) *executing* a transaction and checking its correctness, thereby *endorsing* it (corresponding to "transaction validation" in other blockchains); (2) *ordering* through a consensus protocol, irrespective of transaction semantics; and (3) transaction *validation* per application-specific trust assumptions, which also prevents race conditions due to concurrency.

This design departs radically from the order-execute paradigm in that Fabric typically executes transactions before reaching final agreement on their order. It combines the two well-known approaches to replication, *passive* and *active*, as follows.

First, Fabric uses *passive* or *primary-backup replication* [38, 54] as often found in distributed databases, but with middleware-based asymmetric update processing [133, 134] and ported to untrusted environments with Byzantine faults. In Fabric, every transaction is executed (endorsed) only by a subset of the peers, which allows for parallel execution and addresses potential non-determinism, drawing on "execute-verify" BFT replication [129]. A flexible endorsement policy specifies which peers, or how many of them, need to vouch for the correct execution of a given smart contract.

Second, Fabric incorporates *active replication* in the sense that the transaction's effects on the ledger state are only written after reaching consensus on a total order among them, in the deterministic validation step executed by each peer individually. This allows Fabric to respect

application-specific trust assumptions according to the transaction endorsement. Moreover, the ordering of state updates is delegated to a modular component for consensus (i.e., atomic broadcast), which is stateless and logically decoupled from the peers that execute transactions and maintain the ledger. Since consensus is modular, its implementation can be tailored to the trust assumption of a particular deployment. Although it is readily possible to use the blockchain peers also for implementing consensus, the separation of the two roles adds flexibility and allows one to rely on well-established toolkits for CFT (crash fault-tolerant) or BFT ordering.

Overall, this *hybrid replication* design, which mixes passive and active replication in the Byzantine model, and the *execute-order-validate* paradigm, represent the main innovation in Fabric architecture. They resolve the issues mentioned before and make Fabric a scalable system for permissioned blockchains supporting flexible trust assumptions.

To implement this architecture, Fabric contains modular building blocks for each of the following components:

- An *ordering service* atomically broadcasts state updates to peers and establishes consensus on the order of transactions.
- A *membership service provider* is responsible for associating peers with cryptographic identities. It maintains the permissioned nature of Fabric.
- An optional *peer-to-peer gossip service* disseminates the blocks output by ordering service to all peers.
- *Smart contracts* in Fabric run within a container environment for isolation. They can be written in standard programming languages but do not have direct access to the ledger state.
- Each peer locally maintains the *ledger* in the form of the append-only blockchain and as a snapshot of the most recent state in a key-value store.

The remainder of this chapter describes the architecture of Fabric and our experience with it. Section A.2 summarizes the state of the art and explains the rationale behind various design decisions. Section A.3 introduces the architecture and the execute-order-validate approach of Fabric in detail, illustrating the transaction execution flow. In Section A.4, the key components of Fabric are defined, in particular, the ordering service, membership service, peer-to-peer gossip, ledger database, and smart-contract API. Results and insights gained in a performance evaluation of Fabric with a Bitcoin-inspired cryptocurrency, deployed in a cluster and WAN environments on commodity public cloud VMs, are given in Section A.5. They show that Fabric achieves, in popular deployment configurations, throughput of more than 3500 tps, achieving finality [225] with latency of a few hundred ms and scaling well to over 100 peers. In Section A.6 we discuss a few real production use cases of Fabric. Finally, Section A.7 discusses related work.

A.2 Background

A.2.1 Order-Execute Architecture for Blockchains

All previous blockchain systems, permissioned or not, follow the order-execute architecture. This means that the blockchain network orders transactions first, using a consensus protocol, and then executes them in the same order on all peers sequentially.[1]

For instance, a PoW-based permissionless blockchain such as Ethereum combines consensus and execution of transactions as follows: (1) every peer (i.e., a node that participates in consensus) assembles a block containing valid transactions (to establish validity, this peer already pre-executes those transactions); (2) the peer tries to solve a PoW puzzle [166]; (3) if the peer is lucky and solves the puzzle, it disseminates the block to the network via a gossip protocol; and (4) every peer receiving the block validates the solution to the puzzle *and* all transactions in the block. Effectively, every peer thereby repeats the execution of the lucky

[1] In many blockchains with a hard-coded primary application, such as Bitcoin, this transaction execution is called "transaction validation." Here we call this step *transaction execution* to harmonize the terminology.

peer from its first step. Moreover, all peers execute the transactions *sequentially* (within one block and across blocks). The order-execute architecture is illustrated by Fig. A.1.

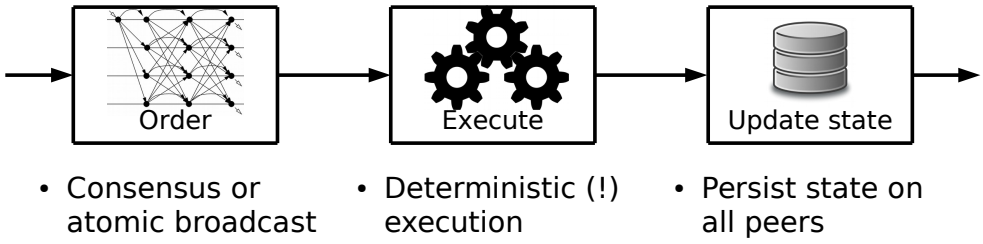


Figure A.1 Order-execute architecture in replicated services.

Existing permissioned blockchains such as Tendermint, Chain, or Quorum typically use BFT consensus [44], provided by PBFT [51] or other protocols for atomic broadcast. Nevertheless, they all follow the same order-execute approach and implement classical *active* SMR [54, 194].

A.2.2 Limitations of Order-Execute

The order-execute architecture is conceptually simple and therefore also widely used. However, it has several drawbacks when used in a general-purpose permissioned blockchain. We discuss the three most significant ones next.

Sequential execution. Executing the transactions sequentially on all peers limits the effective throughput that can be achieved by the blockchain. In particular, since the throughput is inversely proportional to the execution latency, this may become a performance bottleneck for all but the simplest smart contracts. Moreover, recall that in contrast to traditional SMR, the blockchain forms a universal computing engine and its payload applications might be deployed by an adversary. A denial-of-service (DoS) attack, which severely reduces the performance of such a blockchain, could simply introduce smart contracts that take a very long time to execute. For example, a smart contract that executes an infinite loop has a fatal effect, but

cannot be detected automatically because the halting problem is unsolvable.

To cope with this issue, public programmable blockchains with a cryptocurrency account for the execution cost. Ethereum [228], for example, introduces the concept of *gas* consumed by a transaction execution, which is converted at a *gas price* to a cost in the cryptocurrency and billed to the submitter of the transaction. Ethereum goes a long way to support this concept, assigns a cost to every low-level computation step, introducing its own VM for controlling execution. Although this appears to be a viable solution for public blockchains, it is not adequate in the permissioned model for a general-purpose system without a native cryptocurrency.

The distributed-systems literature proposes many ways to improve performance compared to sequential execution, for instance through parallel execution of unrelated operations [173]. Unfortunately, such techniques are still to be applied successfully in the blockchain context of smart contracts. For instance, one challenge is the requirement for deterministically inferring all dependencies across smart contracts, which is particularly challenging when combined with possible confidentiality constraints. Furthermore, these techniques are of no help against DoS attacks by contract code from untrusted developers.

Non-deterministic code. Another important problem for an order-execute architecture are non-deterministic transactions. Operations executed after consensus in active SMR must be deterministic, or the distributed ledger "forks" and violates the basic premise of a blockchain, that all peers hold the same state. This is usually addressed by programming blockchains in domain-specific languages (e.g., Ethereum Solidity) that are expressive enough for their applications but limited to deterministic execution. However, such languages are difficult to design for the implementer and require additional learning by the programmer. Writing smart contracts in a general-purpose language (e.g., Go, Java, C/C++) instead appears more attractive and accelerates the adoption of blockchain solutions.

Unfortunately, generic languages pose many problems for ensuring deterministic execution.

Even if the application developer does not introduce obviously non-deterministic operations, hidden implementation details can have the same devastating effect (e.g., a map iterator is not deterministic in Go). To make matters worse, on a blockchain the burden to create deterministic applications lies on the potentially untrusted programmer. Only one non-deterministic contract created with malicious intent is enough to bring the whole blockchain to a halt. A modular solution to filter diverging operations on a blockchain has also been investigated [43], but it appears costly in practice.

Confidentiality of execution. According to the blueprint of public blockchains, many permissioned systems run all smart contracts on all peers. However, many intended use cases for permissioned blockchains require *confidentiality*, i.e., that access to smart-contract logic, transaction data, or ledger state can be restricted. Although cryptographic techniques, ranging from data encryption to advanced zero-knowledge proofs [193] and verifiable computation [140], can help to achieve confidentiality, this often comes with a considerable overhead and is not viable in practice.

Fortunately, it suffices to *propagate the same state* to all peers instead of running the same code everywhere. Thus, the execution of a smart contract can be restricted to a subset of the peers trusted for this task, that vouch for the results of the execution. This design departs from active replication towards a variant of *passive* replication [38], adapted to the trust model of blockchain.

A.2.3 Further Limitations of Existing Architectures

Fixed trust model. Most permissioned blockchains rely on asynchronous BFT replication protocols to establish consensus [225]. Such protocols typically rely on a security assumption that among $n > 3f$ peers, up to f are tolerated to misbehave and exhibit so-called *Byzantine faults* [32]. The same peers often execute the applications as well, under the same security assumption (even though one could actually restrict BFT execution to fewer peers [232]).

However, such a quantitative trust assumption, irrespective of peers' roles in the system, may not match the trust required for smart-contract execution. In a flexible system, trust at the application level should not be fixed to trust at the protocol level. A general-purpose blockchain should decouple these two assumptions and permit flexible trust models for applications.

Hard-coded consensus. Fabric is the first blockchain system that introduced pluggable consensus. Before Fabric, virtually all blockchain systems, permissioned or not, came with a hard-coded consensus protocol. However, decades of research on consensus protocols have shown there is no such "one-size-fits-all" solution. For instance, BFT protocols differ widely in their performance when deployed in potentially adversarial environments [200]. A protocol with a "chain" communication pattern exhibits provably optimal throughput on a LAN cluster with symmetric and homogeneous links [112], but degrades badly on a wide-area, heterogeneous network. Furthermore, external conditions such as load, network parameters, and actual faults or attacks may vary over time in a given deployment. For these reasons, BFT consensus should be inherently reconfigurable and ideally adapt dynamically to a changing environment [20]. Another important aspect is to match the protocol's trust assumption to a given blockchain deployment scenario. Indeed, one may want to replace BFT consensus with a protocol based on an alternative trust model such as XFT [152], or a CFT protocol, such as Paxos/Raft [170] and ZooKeeper [126], or even a permissionless protocol.

A.2.4 Experience with Order-Execute Blockchains

Prior to realizing the execute-order-validate architecture of Fabric, we gained experience with building a permissioned blockchain platform in the order-execute model, with PBFT [51] for consensus. Namely, previous versions of Fabric (up to v0.6, released in September 2016) have been architected following the 'traditional' order-execute architecture.

From feedback obtained in many proof-of-concept applications, the limitations of this

approach became immediately clear. For instance, users often observed diverging states at the peers and reported a bug in the consensus protocol; in *all cases*, closer inspection revealed that the culprit was non-deterministic transaction code. Other complaints addressed limited performance, e.g., "only five transactions per second," until users confessed that their average transaction took 200ms to execute. We have learned that the key properties of a blockchain system, namely consistency, security, and performance, must *not* depend on the knowledge and goodwill of its users, in particular since the blockchain should run in an untrusted environment.

A.3 Architecture

In this section, we introduce the three-phase *execute-order-validate* architecture and then explain the transaction flow.

A.3.1 Fabric Overview

Fabric is a distributed operating system for permissioned blockchains that executes distributed applications written in general-purpose programming languages (e.g., Go, Java, Node.js). It securely tracks its execution history in an append-only replicated ledger data structure and has no cryptocurrency built in.

Fabric introduces the *execute-order-validate* blockchain architecture (illustrated in Fig. A.2) and does not follow the standard order-execute design, for reasons explained in Section A.2. In a nutshell, a distributed application for Fabric consists of two parts:

- A smart contract, called *chaincode*, which is program code that implements the application logic and runs during the *execution phase*. The chaincode is the central part of a distributed application in Fabric and may be written by an untrusted developer. Special chaincodes exist for managing the blockchain system and maintaining parameters, collectively called *system chaincodes* (Section A.4.6).

- An *endorsement policy* that is evaluated in the *validation phase*. Endorsement policies cannot be chosen or modified by untrusted application developers. An endorsement policy acts as a static library for transaction validation in Fabric, which can merely be parameterized by the chaincode. Only designated *administrators* may have a permission to modify endorsement policies through system management functions. A typical endorsement policy lets the chaincode specify the endorsers for a transaction in the form of a set of peers that are necessary for endorsement; it uses a monotone logical expression on sets, such as "three out of five" or "$(A \wedge B) \vee C$." Custom endorsement policies may implement arbitrary logic (e.g., our Bitcoin-inspired cryptocurrency in Section A.5.1).

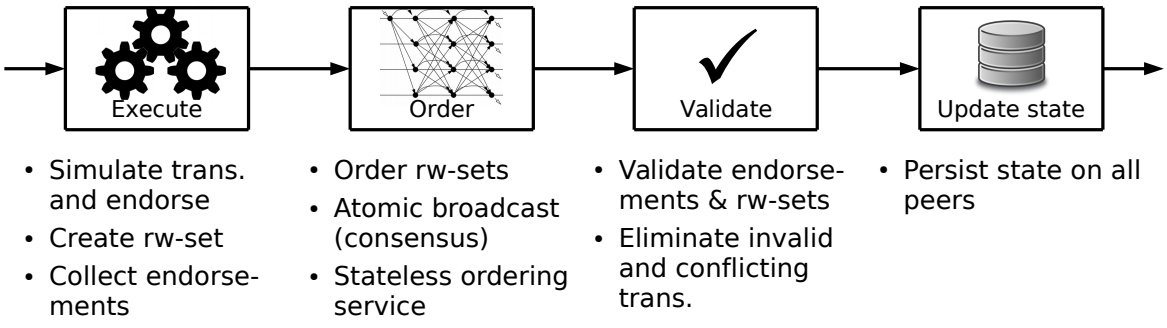


Figure A.2 Execute-order-validate architecture of Fabric (*rw-set* means a readset and writeset as explained in A.3.2).

A client sends transactions to the peers specified by the endorsement policy. Each transaction is then executed by specific peers and its output is recorded; this step is also called *endorsement*. After execution, transactions enter the *ordering phase*, which uses a pluggable consensus protocol to produce a totally ordered sequence of endorsed transactions grouped in blocks. These are broadcast to all peers, with the (optional) help of gossip. Unlike standard active replication [194], which totally orders transaction *inputs*, Fabric orders transaction *outputs* combined with state dependencies, as computed during the execution phase. Each peer

then validates the state changes from endorsed transactions with respect to the endorsement policy and the consistency of the execution in the *validation phase*. All peers validate the transactions in the same order and validation is deterministic. In this sense, Fabric introduces a novel *hybrid replication* paradigm in the Byzantine model, which combines passive replication (the pre-consensus computation of state updates) and active replication (the post-consensus validation of execution results and state changes).

A Fabric blockchain consists of a set of *nodes* that form a *network* (see Fig. A.3). As Fabric is *permissioned*, all nodes that participate in the network have an identity, as provided by a modular *membership service provider (MSP)* (Section A.4.1). Nodes in a Fabric network take up one of three roles:

- *Clients* submit *transaction proposals* for *execution*, help orchestrate the execution phase, and, finally, broadcast *transactions* for ordering.

- *Peers* execute transaction proposals and *validate* transactions. All peers maintain the blockchain *ledger*, an append-only data structure recording all transactions in the form of a hash chain, as well as the *state*, a succinct representation of the latest ledger state. Not all peers execute all transaction proposals, only a subset of them called *endorsing peers* (or, simply, *endorsers*) does, as specified by the policy of the chaincode to which the transaction pertains.

- *Ordering Service Nodes (OSN)* (or, simply, *orderers*) are the nodes that collectively form the *ordering service*. In short, the ordering service establishes the *total order* of all transactions in Fabric, where each transaction contains state updates and dependencies computed during the execution phase, along with cryptographic signatures of the endorsing peers. Orderers are entirely unaware of the application state, and do not participate in the execution nor in the validation of transactions. This design choice renders consensus in Fabric as modular as possible and simplifies replacement of consensus protocols in Fabric.

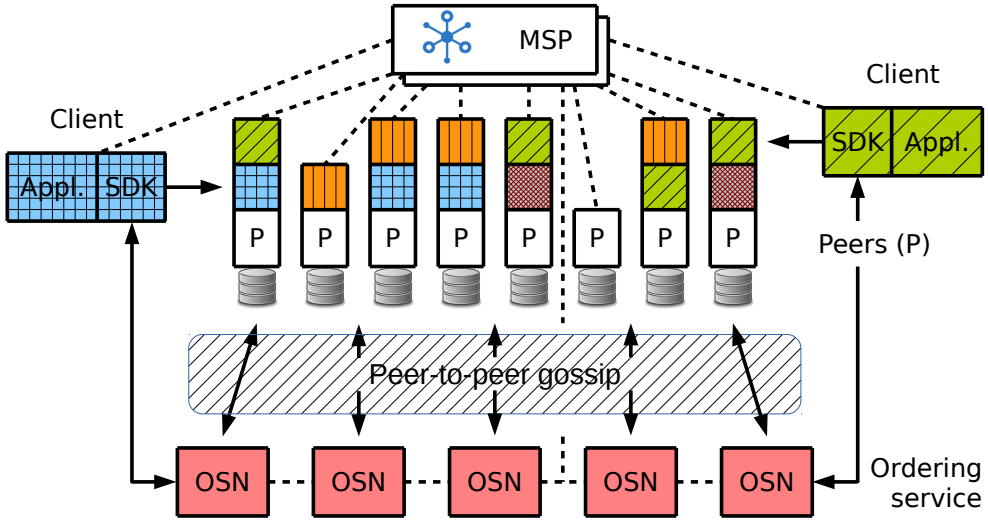


Figure A.3 A Fabric network with federated MSPs and running multiple (differently shaded and colored) chaincodes, selectively installed on peers according to policy.

A Fabric network actually supports multiple blockchains connected to the same ordering service. Each such blockchain is called a *channel* and may have different peers as its members. Channels can be used to partition the state of the blockchain network, but consensus across channels is not coordinated and the total order of transactions in each channel is separate from the others. Certain deployments that consider all orderers as trusted may also implement by-channel access control for peers. In the following we mention channels only briefly and concentrate on one single channel.

In the next three sections we explain the transaction flow in Fabric (depicted in Fig. A.4) and illustrate the steps of the execution, ordering, and validation phases. Then, we summarize the trust and fault model of Fabric (Section A.3.5).

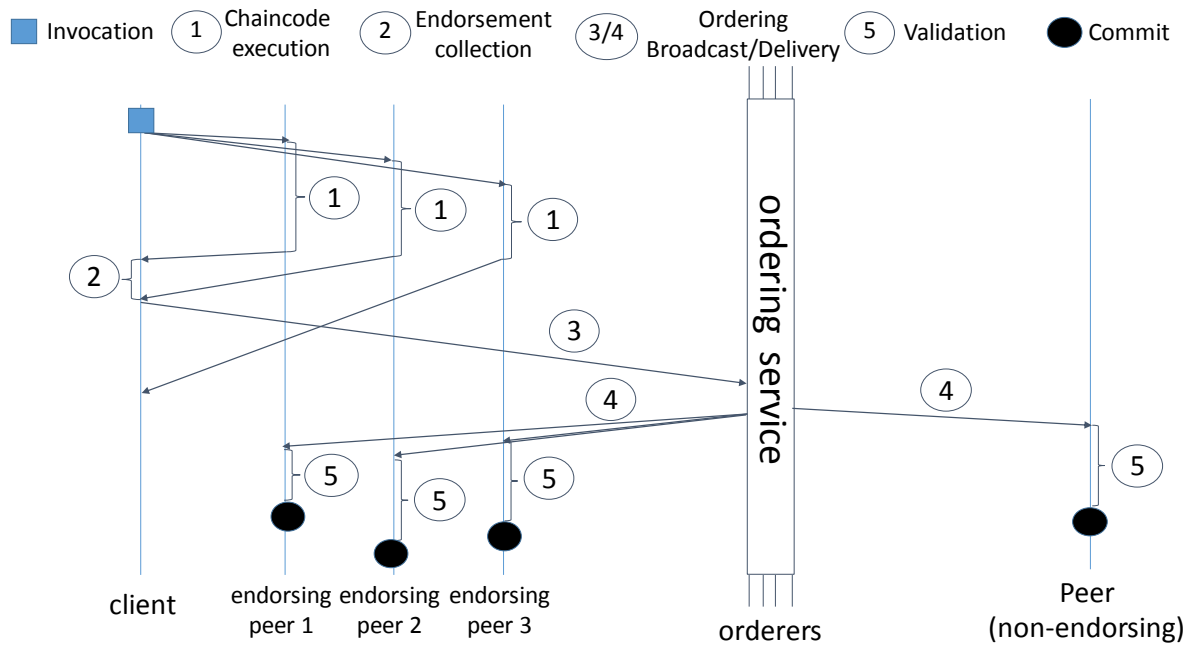


Figure A.4 Fabric high-level transaction flow.

A.3.2 Execution Phase

In the execution phase, clients sign and send the *transaction proposal* (or, simply, *proposal*) to one or more endorsers for execution. Recall that every chaincode implicitly specifies a set of endorsers via the endorsement policy. A proposal contains the identity of the submitting client (according to the MSP), the transaction payload in the form of an operation to execute, parameters, and the identifier of the chaincode, a nonce to be used only once by each client (such as a counter or a random value), and a transaction identifier derived from the client identifier and the nonce.

The endorsers *simulate* the proposal, by executing the operation on the specified chaincode, which has been installed on the blockchain. The chaincode runs in a Docker container, isolated from the main endorser process.

A proposal is simulated against the endorser's local blockchain state, without synchronization with other peers. Moreover, endorsers do not persist the results of the simulation

to the ledger state. The state of the blockchain is maintained by the *peer transaction manager (PTM)* in the form of a versioned key-value store, in which successive updates to a key have monotonically increasing version numbers (Section A.4.4). The state created by a chaincode is scoped exclusively to that chaincode and cannot be accessed directly by another chaincode. Note that the chaincode is not supposed to maintain the local state in the program code, only what it maintains in the blockchain state that is accessed with *GetState*, *PutState*, and *DelState* operations. Given the appropriate permission, a chaincode may invoke another chaincode to access its state within the same channel.

As a result of the simulation, each endorser produces a value *writeset*, consisting of the state updates produced by simulation (i.e., the modified keys along with their new values), as well as a *readset*, representing the version dependencies of the proposal simulation (i.e., all keys read during simulation along with their version numbers). After the simulation, the endorser cryptographically signs a message called *endorsement*, which contains *readset* and *writeset* (together with metadata such as transaction ID, endorser ID, and endorser signature) and sends it back to the client in a *proposal response*. The client collects endorsements until they satisfy the endorsement policy of the chaincode, which the transaction invokes (see Section A.3.4). In particular, this requires all endorsers as determined by the policy to produce the same execution result (i.e., identical *readset* and *writeset*). Then, the client proceeds to create the transaction and passes it to the ordering service.

Discussion on design choices. As the endorsers simulate the proposal without synchronizing with other endorsers, two endorsers may execute it on different states of the ledger and produce different outputs. For the standard endorsement policy which requires multiple endorsers to produce the same result, this implies that under high contention of operations accessing the same keys, a client may not be able to satisfy the endorsement policy. This is a new consideration compared to primary-backup replication in replicated databases with synchronization through middleware [133]: a consequence of the assumption that no single

peer is trusted for correct execution in a blockchain.

We consciously adopted this design, as it considerably simplifies the architecture and is adequate for typical blockchain applications. As demonstrated by the approach of Bitcoin, distributed applications can be formulated such that contention by operations accessing the same state can be reduced, or eliminated completely in the normal case (e.g., in Bitcoin, two operations that modify the same "object" are not allowed and represent a double-spending attack [166]). In the future, we plan to gradually enhance the liveness semantics of Fabric under contention, in particular to support CRDTs [198] for complementing the current version dependency checks, as well as a per-chaincode lead-endorser that would act as a transaction sequencer.

Executing a transaction before the ordering phase is critical to tolerating non-deterministic c chaincodes (see also Section A.2). A chaincode in Fabric with non-deterministic transactions can only endanger the liveness of its own operations, because a client might not gather a sufficient number of endorsements, for instance. This is a fundamental advantage over order-execute architecture, where non-deterministic operations lead to inconsistencies in the state of the peers.

Finally, tolerating non-deterministic execution also addresses DoS attacks from untrusted chaincode as an endorser can simply abort an execution according to a local policy if it suspects a DoS attack. This will not endanger the consistency of the system, and again, such unilateral abortion of execution is not possible in order-execute architectures.

A.3.3 Ordering Phase

When a client has collected enough endorsements on a proposal, it assembles a *transaction* and submits this to the ordering service. The transaction contains the transaction payload (i.e., the chaincode operation including parameters), transaction metadata, and a set of endorsements. The ordering phase establishes a total order on all submitted transactions per channel. In other words, ordering atomically broadcasts [42] endorsements and thereby establishes consensus

on transactions, despite faulty orderers. Moreover, the ordering service batches multiple transactions into *blocks* and outputs a hash-chained sequence of blocks containing transactions. Grouping or batching transactions into blocks improves the throughput of the broadcast protocol, which is a well-known technique used in fault-tolerant broadcasts.

At a high level, the interface of the ordering service only supports the following two operations invoked by a peer and implicitly parameterized by a channel identifier:

- *broadcast*(tx): A client calls this operation to *broadcast* an arbitrary transaction tx, which usually contains the transaction payload and a signature of the client, for dissemination.
- $B \leftarrow$ *deliver*(s): A client calls this to retrieve block B with non-negative sequence number s. The block contains a list of transactions $[tx_1, \ldots, tx_k]$ and a hash-chain value h representing the block with sequence number $s-1$, i.e., $B = ([tx_1, \ldots, tx_k], h)$. As the client may call this multiple times and always returns the same block once it is available, we say the peer *delivers* block B with sequence number s when it receives B *for the first* time upon invoking *deliver*(s).

The ordering service ensures that the *delivered* blocks on one channel are totally ordered. More specifically, ordering ensures the following safety properties for each channel:

Agreement: For any two blocks B delivered with sequence number s and B' delivered with s' at correct peers such that $s = s'$, it holds $B = B'$.

Hash chain integrity: If some correct peer delivers a block B with number s and another correct peer delivers block $B' = ([tx_1, \ldots, tx_k], h')$ with number $s+1$, then it holds $h' = H(B)$, where $H(\cdot)$ denotes the cryptographic hash function.

No skipping: If a correct peer p delivers a block with number $s > 0$ then for each $i = 0, \ldots, s-1$, peer p has already delivered a block with number i.

No creation: When a correct peer delivers block B with number s, then for every $tx \in B$ some client has already broadcast tx.

For liveness, the ordering service supports at least the following "eventual" property:

Validity: If a correct client invokes $broadcast(tx)$, then every correct peer eventually delivers a block B that includes tx, with some sequence number.

However, every individual ordering implementation is allowed to come with its own liveness and fairness guarantees with respect to client requests.

Since there may be a large number of peers in the blockchain network, but only relatively few nodes are expected to implement the ordering service, Fabric can be configured to use a built-in *gossip service* for disseminating delivered blocks from the ordering service to all peers (Section A.4.3). The implementation of gossip is scalable and agnostic to the particular implementation of the ordering service, hence it works with both CFT and BFT ordering services, ensuring the modularity of Fabric.

The ordering service may also perform access control checks to see if a client is allowed to broadcast messages or receive blocks on a given channel. This and other features of the ordering service are further explained in Section A.4.2.

Discussion on design choices. It is very important that the ordering service does not maintain any state of the blockchain, and neither validates nor executes transactions. This architecture is a crucial, defining feature of Fabric, and makes Fabric the first blockchain system to totally separate consensus from execution and validation. This makes consensus as modular as possible, and enables an ecosystem of consensus protocols implementing the ordering service. The *hash chain integrity* property and the chaining of blocks exist only to make the integrity verification of the block sequence by the peers more efficient. Finally, note that we do not require the ordering service to prevent transaction duplication. This simplifies its implementation and is not a concern since duplicated transactions are filtered in the read-write check by the peers during validation.

A.3.4 Validation Phase

Blocks are delivered to peers either directly by the ordering service or through gossip. A new block then enters the validation phase which consists of three sequential steps:

1. The *endorsement policy evaluation* occurs in parallel for all transactions within the block. The evaluation is the task of the so-called *validation system chaincode (VSCC)*, a static library that is part of the blockchain's configuration and is responsible for validating the endorsement with respect to the endorsement policy configured for the chaincode (see Section A.4.6). If the endorsement is not satisfied, the transaction is marked as invalid and its effects are disregarded.

2. A *read-write conflict check* is done for all transactions in the block sequentially. For each transaction it compares the versions of the keys in the *readset* field to those in the current state of the ledger, as stored locally by the peer, and ensures they are still the same. If the versions do not match, the transaction is marked as invalid and its effects are disregarded.

3. The *ledger update phase* runs last, in which the block is appended to the locally stored ledger and the blockchain state is updated. In particular, when adding the block to the ledger, the results of the validity checks in the first two steps are persisted as well, in the form of a bit mask denoting the transactions that are valid within the block. This facilitates the reconstruction of the state at a later time. Furthermore, all state updates are applied by writing all key-value pairs in *writeset* to the local state.

The default VSCC in Fabric allows monotone logical expressions over the set of endorsers configured for a chaincode to be expressed. The VSCC evaluation verifies that the set of peers, as expressed through valid signatures on endorsements of the transaction, satisfy the expression. Different VSCC policies can be configured statically, however.

Discussion on design choices. The ledger of Fabric contains all transactions, including those that are deemed invalid. This follows from the overall design, because ordering service, which is agnostic to chaincode state, produces the chain of the blocks and because the validation is done by the peers post-consensus. This feature is needed in certain use cases that require tracking of invalid transactions during subsequent audits, and stands in contrast to other blockchains (e.g., Bitcoin and Ethereum), where the ledger contains only valid transactions. In addition, due to the permissioned nature of Fabric, detecting clients that try to mount a DoS attack by flooding the network with invalid transactions is easy. One approach would be to black-list such clients according to a policy that could be put in place. Furthermore, a specific deployment could implement transaction fees (using our currency implementation from Section A.5.1 or another approach) to charge for transaction invocation, which would render a DoS attack prohibitively expensive.

A.3.5 Trust and Fault Model

Fabric can accommodate flexible trust and fault assumptions. In general, any client is considered potentially malicious or *Byzantine*. Peers are grouped into *organizations* and every organization forms one trust domain, such that a peer trusts all peers within its organization but no peer of another organization. The ordering service considers all peers (and clients) as potentially Byzantine.

The integrity of a Fabric network relies on the consistency of the ordering service. The trust model of the ordering service depends directly on its implementation (see Section A.3.3). As of release v1.0.6, Fabric supports a centralized, single-node implementation, used in development and testing, and a CFT ordering service running on a cluster. A third implementation, a proof of concept based on *BFT-SMaRt* [27], tolerates up to one third of Byzantine OSNs [205].

Note that Fabric decouples the trust model for applications from the trust model of consensus. Namely, a distributed application can define its own trust assumptions, which are conveyed through the endorsement policy, and are independent from those of consensus

implemented by the ordering service (see also Section A.3.4).

A.4 Fabric Components

Fabric is written in Go and uses the gRPC framework [111] for communication between clients, peers, and orderers. In the following we describe some important components in more detail. Fig. A.5 shows the components of a peer.

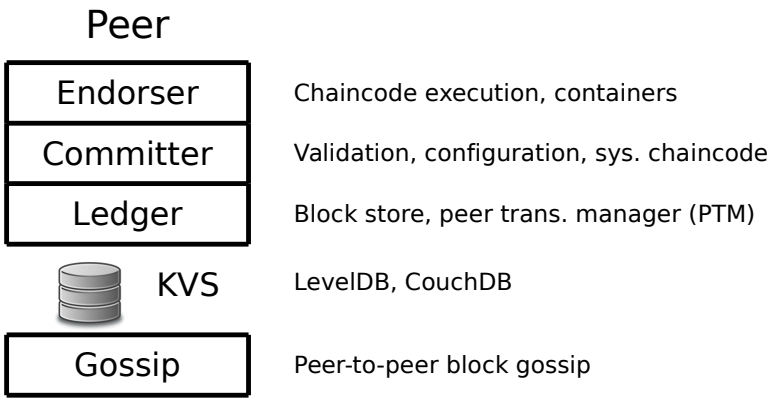


Figure A.5 Components of a Fabric peer.

A.4.1 Membership Service

The *membership service provider (MSP)* maintains the identities of all nodes in the system (clients, peers, and OSNs) and is responsible for issuing node credentials that are used for authentication and authorization. Since Fabric is *permissioned*, all interactions among nodes occur through messages that are authenticated, typically with digital signatures. The membership service comprises a component at each node, where it may authenticate transactions, verify the integrity of transactions, sign and validate endorsements, and authenticate other blockchain operations. Tools for key management and registration of nodes are also part of the MSP.

The MSP is an abstraction for which different instantiations are possible. The default MSP implementation in Fabric handles standard PKI methods for authentication based on digital signatures and can accommodate commercial certification authorities (CAs). A stand-alone CA is provided as well with Fabric, called *Fabric-CA*. Furthermore, alternative MSP implementations are envisaged, such as one relying on anonymous credentials for authorizing a client to invoke a transaction without linking this to an identity [46].

Fabric allows two modes for setting up a blockchain network. In *offline mode*, credentials are generated by a CA and distributed out-of-band to all nodes. Peers and orderers can only be registered in offline mode. For enrolling clients, Fabric-CA provides an *online mode* that issues cryptographic credentials to them. The MSP configuration must ensure that all nodes, especially all peers, recognize the same identities and authentications as valid.

The MSP permits identity federation, for example, when multiple organizations operate a blockchain network. Each organization issues identities to its own members and every peer recognizes members of all organizations. This can be achieved with multiple MSP instantiations, for example, by creating a mapping between each organization and an MSP.

A.4.2 Ordering Service

The ordering service manages multiple channels. On every channel, it provides the following services:

1. *Atomic broadcast* for establishing order on transactions, implementing the *broadcast* and *deliver* calls (Section A.3.3).

2. *Reconfiguration* of a channel, when its members modify the channel by broadcasting a *configuration update transaction* (Section A.4.6).

3. Optionally, *access control*, in those configurations where the ordering service acts as a trusted entity, restricting broadcasting of transactions and receiving of blocks to specified clients and peers.

The ordering service is bootstrapped with a *genesis block* on the *system channel*. This block carries a *configuration transaction* that defines the ordering service poroperties.

The current production implementation consists of *ordering-service nodes (OSNs)* that implement the operations described here and communicate through the system channel. The actual atomic broadcast function is provided by an instance of Apache Kafka [16] which offers scalable publish-subscribe messaging and strong consistency despite node crashes, based on ZooKeeper. Kafka may run on physical nodes separate from the OSNs. The OSNs act as proxies between the peers and Kafka.

An OSN directly injects a newly received transaction to the atomic broadcast (e.g., to the Kafka broker). OSNs *batch* transactions received from the atomic broadcast and *form blocks*. A block is *cut* as soon as one of three conditions is met: (1) the block contains the specified maximal number of transactions; (2) the block has reached a maximal size (in bytes); or (3) an amount of time has elapsed since the first transaction of a new block was received, as explained below.

This batching process is *deterministic* and therefore produces the same blocks at all nodes. It is easy to see that the first two conditions are trivially deterministic, given the stream of transactions received from the atomic broadcast. To ensure deterministic block production in the third case, a node starts a timer when it reads the first transaction in a block from the atomic broadcast. If the block is not yet cut when the timer expires, the OSN broadcasts a special *time-to-cut* transaction on the channel, which indicates the sequence number of the block which it intends to cut. On the other hand, every OSN immediately cuts a new block upon receiving the *first* time-to-cut transaction for the given block number. Since this transaction is atomically delivered to all connected OSNs, they all include the same list of transactions in the block. The OSNs persist a range of the most recently delivered blocks directly to their filesystem, so they can answer to peers retrieving blocks through *deliver*.

The ordering service based on Kafka is one of three implementations currently available. A centralized orderer, called *Solo*, runs on one node and is used for development. A proof-

of-concept ordering service based on *BFT-SMaRt* [27] has also been made available [205]; it ensures the atomic broadcast service, but not yet reconfiguration and access control. This illustrates the modularity of consensus in Fabric.

A.4.3 Peer Gossip

One advantage of separating the execution, ordering, and validation phases is that they can be scaled independently. However, since most consensus algorithms (in the CFT and BFT models) are bandwidth-bound, the throughput of the ordering service is capped by the network capacity of its nodes. Consensus cannot be scaled up by adding more nodes [64, 225], rather, throughput will decrease. However, since ordering and validation are decoupled, we are interested in efficiently broadcasting the execution results to all peers for validation, after the ordering phase. This, along with *state transfer* to newly joining peers and peers that were disconnected for a long time, is precisely the goal of the gossip component. Fabric gossip utilizes epidemic multicast [68] for this purpose. The blocks are signed by the ordering service. This means that a peer can, upon receiving all blocks, independently assemble the blockchain and verify its integrity.

The communication layer for gossip is based on gRPC and utilizes TLS with mutual authentication, which enables each side to bind the TLS credentials to the identity of the remote peer. The gossip component maintains an up-to-date *membership view* of the online peers in the system. All peers independently build a local view from periodically disseminated membership data. Furthermore, a peer can reconnect to the view after a crash or a network outage.

Fabric gossip uses two phases for information dissemination: during *push*, each peer selects a random set of active neighbors from the membership view, and forwards them the message; during *pull*, each peer periodically probes a set of randomly selected peers and requests missing messages. It has been shown [68, 131] that using both methods in tandem is crucial to optimally utilize the available bandwidth and to ensure that all peers receive all messages

with high probability. In order to reduce the load of sending blocks from the ordering nodes to the network, the protocol also *elects a leader peer* that pulls blocks from the ordering service on their behalf and initiates the gossip distribution. This mechanism is resilient to leader failures.

A.4.4 Ledger

The ledger component at each peer maintains the ledger and the state on persistent storage and enables *simulation*, *validation*, and *ledger-update* phases. Broadly, it consists of a *block store* and a *peer transaction manager*.

The *ledger block store* persists transaction blocks and is implemented as a set of append-only files. Since the blocks are immutable and arrive in a definite order, an append-only structure gives maximum performance. In addition, the block store maintains a few indices for random access to a block or to a transaction in a block.

The *peer transaction manager (PTM)* maintains the *latest state* in a versioned key-value store. It stores one tuple of the form (key, val, ver) for each unique entry *key* stored by any chaincode, containing its most recently stored value *val* and its latest version *ver*. The version consists of the block sequence number and the sequence number of the transaction (that stores the entry) within the block. This makes the version unique and monotonically increasing. The PTM uses a local key-value store to realize its versioned variant, with implementations using LevelDB (in Go) [219] and Apache CouchDB [15].

During simulation the PTM provides a stable snapshot of the latest state to the transaction. As mentioned in Section A.3.2, the PTM records in *readset* a tuple (key, ver) for each entry accessed by *GetState* and in *writeset* a tuple (key, val) for each entry updated with *PutState* by the transaction. In addition, the PTM supports range queries, for which it computes a cryptographic hash of the query results (a set of tuples (key, ver)) and adds the query string itself and the hash to *readset*.

For transaction validation (Section A.3.4), the PTM validates all transactions in a block sequentially. This checks whether a transaction conflicts with any preceding transaction (within

the block or earlier). For any key in *readset*, if the version recorded in *readset* differs from the version present in the latest state (assuming that all preceding valid transactions are committed), then the PTM marks the transaction as invalid. For range queries, the PTM re-executes the query and compares the hash with the one present in *readset*, to ensure that no phantom reads occur. This read-write conflict semantics results in one-copy serializability [135].

The ledger component tolerates a crash of the peer during the ledger update as follows. After receiving a new block, the PTM has already performed validation and marked transactions as valid or invalid within the block, using a bit mask as mentioned in Section A.3.4. The ledger now writes the block to the ledger block store, flushes it to disk, and subsequently updates the block store indices. Then the PTM applies the state changes from *writeset* of all valid transactions to the local versioned store. Finally, it computes and persists a value *savepoint*, which denotes the largest successfully applied block number. The value *savepoint* is used to recover the indices and the latest state from the persisted blocks when recovering from a crash.

A.4.5 Chaincode Execution

Chaincode is executed within an environment loosely coupled to the rest of the peer, which supports plugins for adding new chaincode programming languages. Currently Go, Java, and Node are supported.

Every user-level or application chaincode runs in a separate process within a Docker container environment, which isolates the chaincodes from each other and from the peer. This also simplifies the management of the lifecycle for chaincodes (i.e., starting, stopping, or aborting chaincode). The chaincode and the peer communicate using gRPC messages. Through this loose coupling, the peer is agnostic of the actual language in which chaincode is implemented.

In contrast to application chaincode, system chaincode runs directly in the peer process. System chaincode can implement specific functions needed by Fabric and may be used in

situations where the isolation among user chaincodes is overly restrictive. More details on system chaincodes are given in the next section.

A.4.6 Configuration and System Chaincodes

Fabric is customized through *channel configuration* and through special chaincodes, known as *system chaincodes*.

Recall each channel in Fabric forms one logical blockchain. The *configuration* of a channel is maintained in metadata persisted in special *configuration blocks*. Each configuration block contains the full channel configuration and does not contain any other transactions. Each blockchain begins with a configuration block known as the *genesis block* which is used to bootstrap the channel. The channel configuration includes: (1) definitions of the MSPs for the participating nodes, (2) the network addresses of the OSNs, (3) shared configuration for the consensus implementation and the ordering service, such as batch size and timeouts, (4) rules governing access to the ordering service operations (*broadcast*, and *deliver*), and (5) rules governing how each part the channel configuration may be modified.

The configuration of a channel may be updated using a *channel configuration update transaction*. This transaction contains a representation of the changes to be made to the configuration, as well as a set of signatures. The ordering service nodes evaluate whether the update is valid by using the current configuration to verify that the modifications are authorized using the signatures. The orderers then generate a new configuration block, which embeds the new configuration and the configuration update transaction. Peers receiving this block validate whether the configuration update is authorized based on the current configuration; if valid, they update their current configuration.

The application chaincodes are deployed with a reference to an *endorsement system chaincode (ESCC)* and to a *validation system chaincode (VSCC)*. These two chaincodes are selected such that the output of the ESCC (an endorsement) may be validated as part of the input to the VSCC. The ESCC takes as input a proposal and the proposal simulation results. If the

results are satisfactory, then the ESCC produces a response, containing the results and the endorsement. For the default ESCC, this endorsement is simply a signature by the peer's local signing identity. The VSCC takes as input a transaction and outputs whether that transaction is valid. For the default VSCC, the endorsements are collected and evaluated against the endorsement policy specified for the chaincode. Further system chaincodes implement other support functions, such as chaincode lifecycle.

A.5 Evaluation

Even though Fabric is not yet performance-tuned and optimized, we report in this section on some preliminary performance numbers. Fabric is a complex distributed system; its performance depends on many parameters including the choice of a distributed application and transaction size, the ordering service and consensus implementation and their parameters, the network parameters and topology of nodes in the network, the hardware on which nodes run, the number of nodes and channels, further configuration parameters, and the network dynamics. Therefore, in-depth performance evaluation of Fabric is postponed to future work.

In the absence of a standard benchmark for blockchains, we use the most prominent blockchain application for evaluating Fabric, a simple authority-minted cryptocurrency that uses the data model of Bitcoin, which we call *Fabric coin* (abbreviated hereafter as *Fabcoin*). This allows us to put the performance of Fabric in the context of other permissioned blockchains, which are often derived from Bitcoin or Ethereum. For example, it is also the application used in benchmarks of other permissioned blockchains [184, 197].

In the following, we first describe Fabcoin (Section A.5.1), which also demonstrates how to customize the validation phase and endorsement policy. In Section A.5.2 we present the benchmark and discuss our results.

A.5.1 Fabric Coin (Fabcoin)

UTXO cryptocurrencies The data model introduced by Bitcoin [166] has become known as "unspent transaction output" or *UTXO*, and is also used by many other cryptocurrencies and distributed applications. UTXO represents each step in the evolution of a data object as a separate atomic state on the ledger. Such a state is created by a transaction and destroyed (or "consumed") by another unique transaction occurring later. Every given transaction destroys a number of *input states* and creates one or more *output states*. A "coin" in Bitcoin is initially created by a *coinbase* transaction that rewards the "miner" of the block. This appears on the ledger as a *coin state* designating the miner as the owner. Any coin can be *spent* in the sense that the coin is assigned to a new owner by a transaction that atomically destroys the current coin state designating the previous owner and creates another coin state representing the new owner.

We capture the UTXO model in the key-value store of Fabric as follows. Each UTXO state corresponds to a unique KVS entry that is created once (the coin state is "unspent") and destroyed once (the coin state is "spent"). Equivalently, every state may be seen as a KVS entry with logical version 0 after creation; when it is destroyed again, it receives version 1. There should not be any concurrent updates to such entries (e.g., attempting to update a coin state in different ways amounts to double-spending the coin).

Value in the UTXO model is transferred through transactions that refer to several input states that all belong to the entity issuing the transaction. An entity owns a state because the public key of the entity is contained in the state itself. Every transaction creates one or more output states in the KVS representing the new owners, deletes the input states in the KVS, and ensures that the sum of the values in the input states equals the sum of the output states' values. There is also a policy determining how value is created (e.g., *coinbase* transactions in Bitcoin or specific *mint* operations in other systems) or destroyed.

Fabcoin implementation Each state in Fabcoin is a tuple of the form $(key, val) = (txid.j, (amount, owner, label))$, denoting the coin state created as the j-th output of a transaction with identifier *txid* and allocating *amount* units labeled with *label* to the entity whose public key is *owner*. Labels are strings used to identify a given type of a coin (e.g., 'USD', 'EUR', 'FBC'). Transaction identifiers are short values that uniquely identify every Fabric transaction. The Fabcoin implementation consists of three parts: (1) a client wallet, (2) the Fabcoin chaincode, and (3) a custom VSCC for Fabcoin implementing its endorsement policy.

Client wallet By default, each Fabric client maintains a *Fabcoin wallet* that locally stores a set of cryptographic keys allowing the client to spend coins. For creating a SPEND transaction that transfers one or more coins, the client wallet creates a Fabcoin request $request = (inputs, outputs, sigs)$ containing: (1) a list of *input coin states*, $inputs = [in, \dots]$ that specify coin states $(in, \dots)$ the client wishes to spend, as well as (2) a list of *output coin states*, $outputs = [(amount, owner, label), \dots]$. The client wallet signs, with the private keys that correspond to the input coin states, the concatenation of the Fabcoin request and a nonce, which is a part of every Fabric transaction, and adds the signatures in a set *sigs*. A SPEND transaction is valid when the sum of the amounts in the input coin states is at least the sum of the amounts in the outputs and when the output amounts are positive. For a MINT transaction that creates new coins, *inputs* contains only an identifier (i.e., a reference to a public key) of a special entity called *Central Bank (CB)*, whereas *outputs* contains an arbitrary number of coin states. To be considered valid, the signatures of a MINT transaction in *sigs* must be a cryptographic signature under the public key of CB over the concatenation of the Fabcoin request and the aforementioned nonce. Fabcoin may be configured to use multiple CBs or specify a threshold number of signatures from a set of CBs. Finally, the client wallet includes the Fabcoin request into a transaction and sends this to a peer of its choice.

Fabcoin chaincode A peer runs the chaincode of Fabcoin which simulates the transaction and creates readsets and writesets. In a nutshell, in the case of a SPEND transaction, for every

input coin state $in \in inputs$ the chaincode first performs $GetState(in)$; this includes in in the readset along with its current version (Section A.4.4). Then the chaincode executes $DelState(in)$ for every input state in, which also adds in to the writeset and effectively marks the coin state as "spent." Finally, for $j = 1, \ldots, |outputs|$, the chaincode executes $PutState(txid.j, out)$ with the j-th output $out = (amount, owner, label)$ in $outputs$. In addition, a peer may run the transaction validation code as described next in the VSCC step for Fabcoin; this is not necessary, since the Fabcoin VSCC actually validates transactions, but it allows the (correct) peers to filter out potentially malformed transactions. In our implementation, the chaincode runs the Fabcoin VSCC without cryptographically verifying the signatures (these are verified only in the actual VSCC).

Custom VSCC Finally, every peer validates Fabcoin transactions using a custom VSCC. This verifies the cryptographic signature(s) in *sigs* under the respective public key(s) and performs semantic validation as follows. For a MINT transaction, it checks that the output states are created under the matching transaction identifier (*txid*) and that all output amounts are positive. For a SPEND transaction, the VSCC additionally verifies (1) that for all input coin states, an entry in the readset has been created and that it was also added to the writeset and marked as deleted, (2) that the sum of the amounts for all input coin states equals the sum of amounts of all output coin states, and (3) that input and output coin labels match. Here, the VSCC obtains the input coin amounts by retrieving their current values from the ledger.

Note that the Fabcoin VSCC does not check transactions for double spending, as this occurs through Fabric's standard validation that runs after the custom VSCC. In particular, if two transactions attempt to assign the same unspent coin state to a new owner, both would pass the VSCC logic but would be caught subsequently in the read-write conflict check performed by the PTM. According to Section A.3.4 and Section A.4.4, the PTM verifies that the current version number stored in the ledger matches the one in the readset; hence, after the first transaction has changed the version of the coin state, the transaction ordered second will be

recognized as invalid.

A.5.2 Experiments

Setup Unless explicitly mentioned differently, in our experiments: (1) nodes run on Fabric version v1.1.0-preview[2] instrumented for performance evaluation through local logging, (2) nodes are hosted in a single IBM Cloud (SoftLayer) data center (DC) as dedicated VMs interconnected with 1Gbps (nominal) networking, (3) all nodes are 2.0 GHz 16-vCPU VMs running Ubuntu with 8GB of RAM and SSDs as local disks, (4) a single-channel ordering service runs a typical Kafka orderer setup with 3 ZooKeeper nodes, 4 Kafka brokers and 3 Fabric orderers, all on distinct VMs, (5) there are 5 peers in total, all belonging to different organizations (orgs) and all being Fabcoin endorsers, and (6) signatures use the default 256-bit ECDSA scheme. In order to measure and stage latencies in the transaction flow spanning multiple nodes, the node clocks are synchronized with an NTP service throughout the experiments. All communication among Fabric nodes is configured to use TLS.

Methodology In every experiment, in the first phase we invoke transactions that contain only Fabcoin MINT operations to produce the coins, and then run a second phase of the experiment in which we invoke Fabcoin SPEND operation on previously minted coins (effectively running single-input, single-output SPEND transactions). When reporting throughput measurements, we use an increasing number of Fabric CLI clients (modified to issue concurrent requests) running on a single VM, until the end-to-end throughput is saturated, and state the throughput just *below* saturation. Throughput numbers are reported as the average measured during the steady state of an experiment, disregarding the "tail," where some client threads already stop submitting their share of transactions. In every experiment, the client threads collectively invoke at least 500k MINT and SPEND transactions.

[2]Patched with commit IDs 9*e*770062 and *eb*437*dab* in the Fabric *master* branch.

Experiment 1: Choosing the block size A critical Fabric configuration parameter that impacts both throughput and latency is block size. To fix the block size for subsequent experiments, and to evaluate the impact of block size on performance, we ran experiments varying block size from 0.5MB to 4MB. Results are depicted in Fig. A.6, showing peak throughput measured at the peers along with the corresponding average end-to-end (e2e) latency.

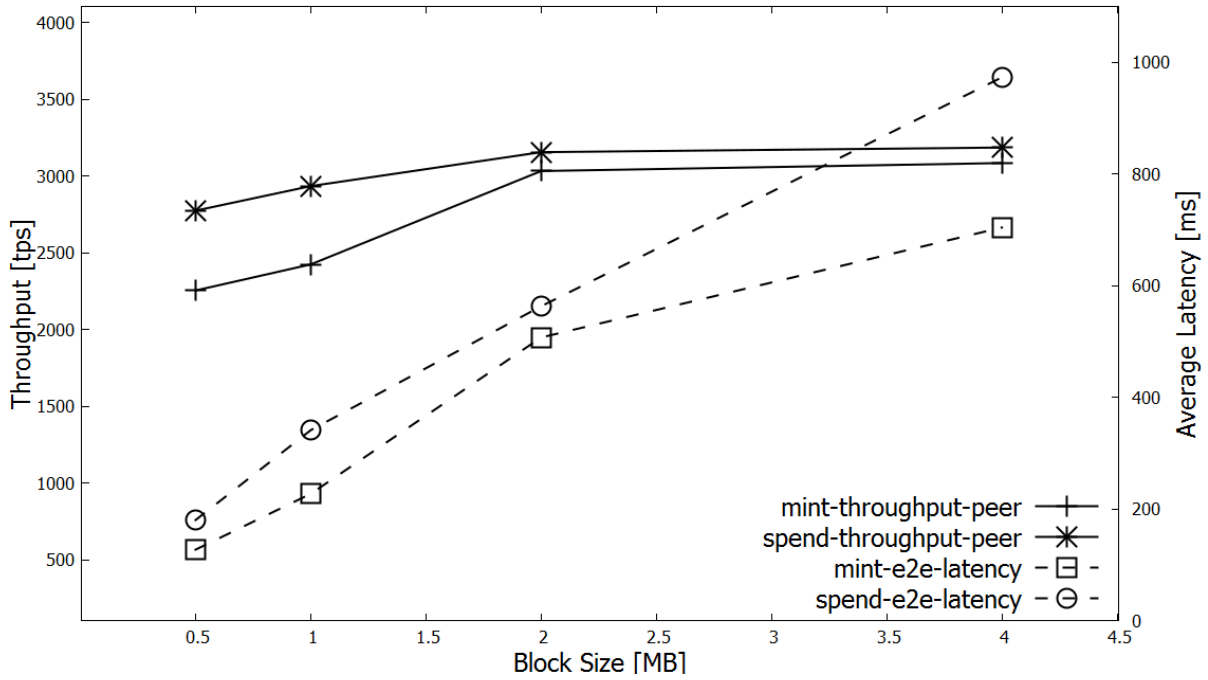


Figure A.6 Impact of block size on throughput and latency.

We can observe that throughput does not significantly improve beyond a block size of 2MB, but latency gets worse (as expected). Therefore, we adopt 2MB as the block size for the following experiments, with the goal of maximizing the measured throughput, assuming the end-to-end latency of roughly 500ms is acceptable.

Size of transactions During this experiment, we also observed the size MINT and SPEND transactions. In particular, the 2MB-blocks contained 473 MINT or 670 SPEND transactions, i.e.,

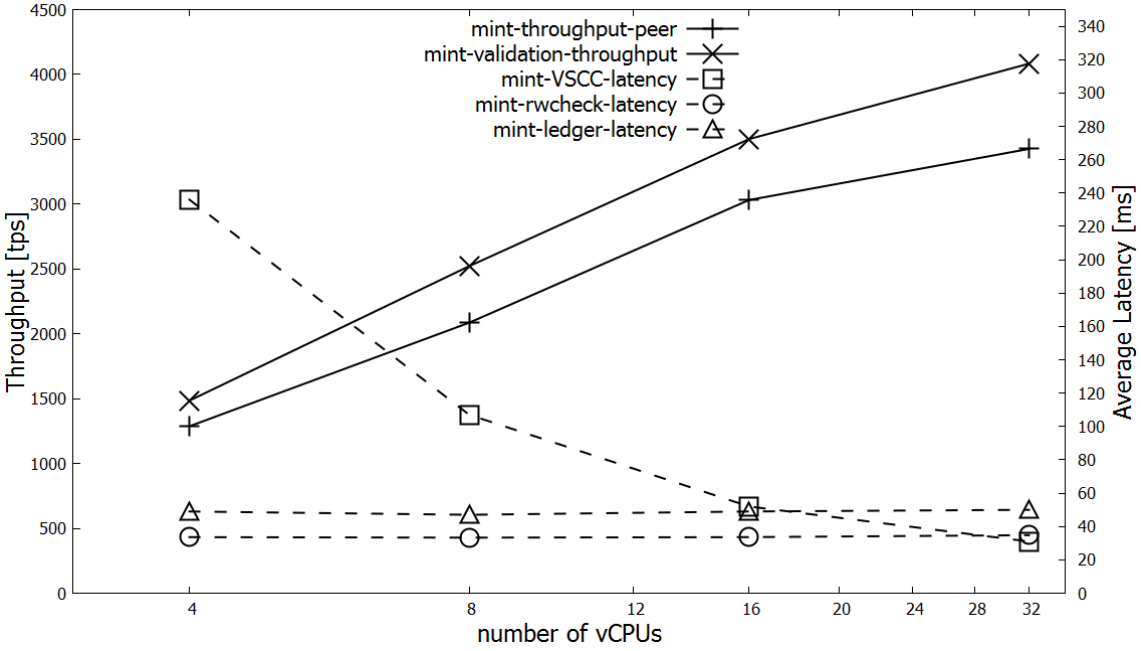
the average transaction size is 3.06kB for SPEND and 4.33kB for MINT. In general, transactions in Fabric are large because they carry certificate information. Besides, MINT transactions of Fabcoin are larger than SPEND transactions because they carry CB certificates. This is an avenue for future improvement of both Fabric and Fabcoin.

Experiment 2: Impact of peer CPU Fabric peers run many CPU-intensive cryptographic operations. To estimate the impact of CPU power on throughput, we performed a set of experiments in which 4 peers run on 4, 8, 16, and 32 vCPU VMs, while also doing coarse-grained latency staging of block validation to identify bottlenecks.

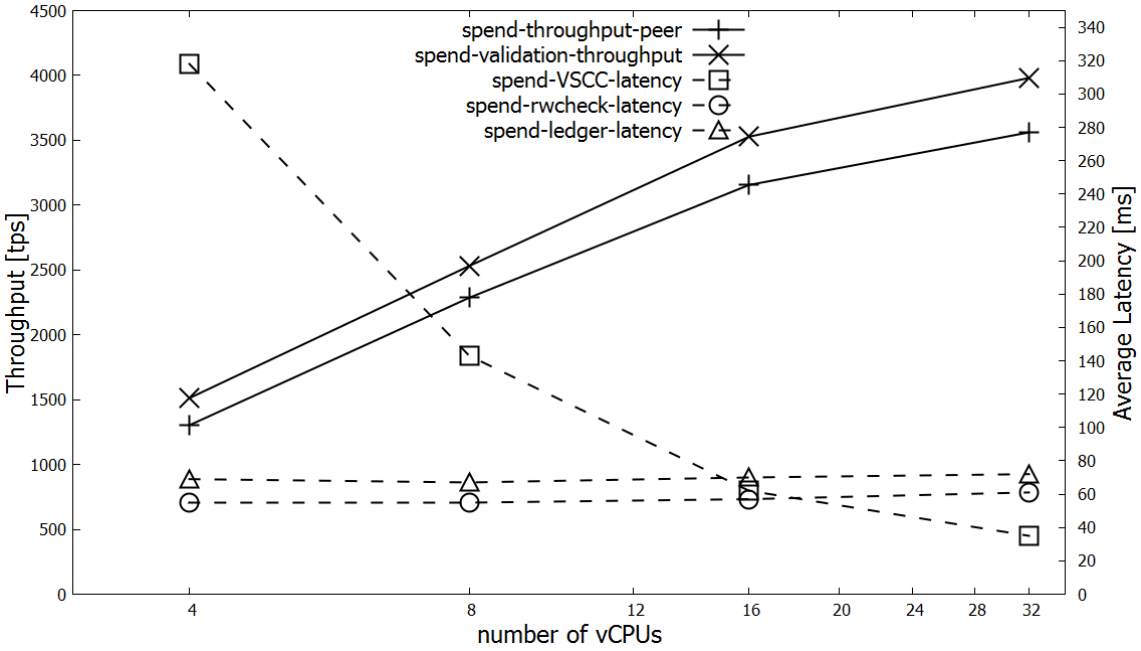
Our experiment focused on the validation phase, as ordering with the Kafka ordering service has never been a bottleneck in our cluster experiments (within one data center). The validation phase, and in particular the VSCC validation of Fabcoin, is computationally intensive, due to its many digital signature verifications. We calculate the validation throughput at the peer based on measuring validation phase latency locally at the peer.

The results, with 2MB blocks, are shown in Fig. A.7, for blocks containing MINT (Fig. A.7a) and SPEND (Fig. A.7b) operations. For both operations the measured throughput and latency scale in the same way with the number of vCPUs. We can observe that the validation effort clearly limits the achievable (end-to-end) throughput. Furthermore, the validation performance by the Fabcoin VSCC scales quasi-linearly with CPU, as the endorsement policy verification by Fabric's VSCC is embarrassingly parallel. However, the read-write-check and ledger-access stages are sequential and become dominant with a larger number of cores (vCPUs). This is in particular noticeable for SPEND transactions, since more SPEND than MINT transactions can fit into a 2MB block, which prolongs the duration of the sequential validation stages (i.e., read-write-check and ledger-access).

This experiment suggests that future versions of Fabric could profit from pipelining the validation stages (which are now sequential), removing sequential overhead in the peer that causes a noticeable difference between validation and end-to-end throughput, optimizing



(a) Blocks containing only MINT transactions.



(b) Blocks containing only SPEND transactions.

Figure A.7 Impact of peer CPU on end-to-end throughput, validation throughput and block validation latency.

stable-storage access, and parallelizing read-write dependency checks.

Finally, in this experiment, we measured over 3560 transactions per second (tps) average SPEND (end-to-end) throughput at the 32-vCPU peer. The MINT throughput is, in general, slightly lower than that of SPEND, but the difference is within 10%, with 32-vCPU peer reaching over 3420 tps average MINT throughput.

Latency profiling by stages We further performed coarse-grained profiling of latency during our previous experiment at the peak reported throughput. Results are depicted in Table A.1. The ordering phase comprises broadcast-deliver latency and internal latency within a peer before validation starts. The table reports average latencies for MINT and SPEND, standard deviation, and tail latencies.

Table A.1 Latency statistics in milliseconds (ms) for MINT and SPEND, broken down into five stages at a 32-vCPU peer with 2MB blocks. Validation (6) comprises stages 3, 4, and 5; the end-to-end latency contains stages 1–5.

	avg	st.dev	99%	99.9%
(1) endorsement	5.6 / 7.5	2.4 / 4.2	15 / 21	19 / 26
(2) ordering	248 / 365	60.0 / 92.0	484 / 624	523 / 636
(3) VSCC val.	31.0 / 35.3	10.2 / 9.0	72.7 / 57.0	113 / 108.4
(4) R/W check	34.8 / 61.5	3.9 / 9.3	47.0 / 88.5	59.0 / 93.3
(5) ledger	50.6 / 72.2	6.2 / 8.8	70.1 / 97.5	72.5 / 105
(6) validation (3+4+5)	116 / 169	12.8 / 17.8	156 / 216	199 / 230
(7) end-to-end (1+2+6)	371 / 542	63 / 94	612 / 805	646 / 813

We observe that ordering dominates the overall latency. We also see that average latencies lie below 550ms with sub-second tail latencies. In particular, the highest end-to-end latencies in our experiment come from the first blocks, during the load build-up. Latency under lower load can be regulated and reduced using the time-to-cut parameter of the orderer (see Section A.3.3), which we basically do not use in our experiments, as we set it to a large value.

Experiment 3: SSD vs. RAM disk To evaluate the potential gains related to stable storage, we repeated the previous experiment with RAM disks (tmpfs) mounted as stable storage at all peer VMs. The benefits are limited, as tmpfs only helps with the ledger phase of the validation at the peer. We measured sustained peak throughput at 3870 SPEND tps at 32-vCPU peer, roughly a 9% improvement over SSD.

Experiment 4: Scalability on LAN In this and the following experiment we increase the number of peers (with 16 vCPUs each) to evaluate the scalability of Fabric.

In this experiment we maintain one peer per organization hosted in a single IBM Cloud DC (Hong Kong, HK). All peers receive blocks directly from the ordering service without gossip. We start from 20 peers (10 of which are Fabcoin endorsers) and increase the number of peers to 100. The achievable peak throughput in function of the number of peers is depicted in Fig. A.8 ("LAN" suffix).

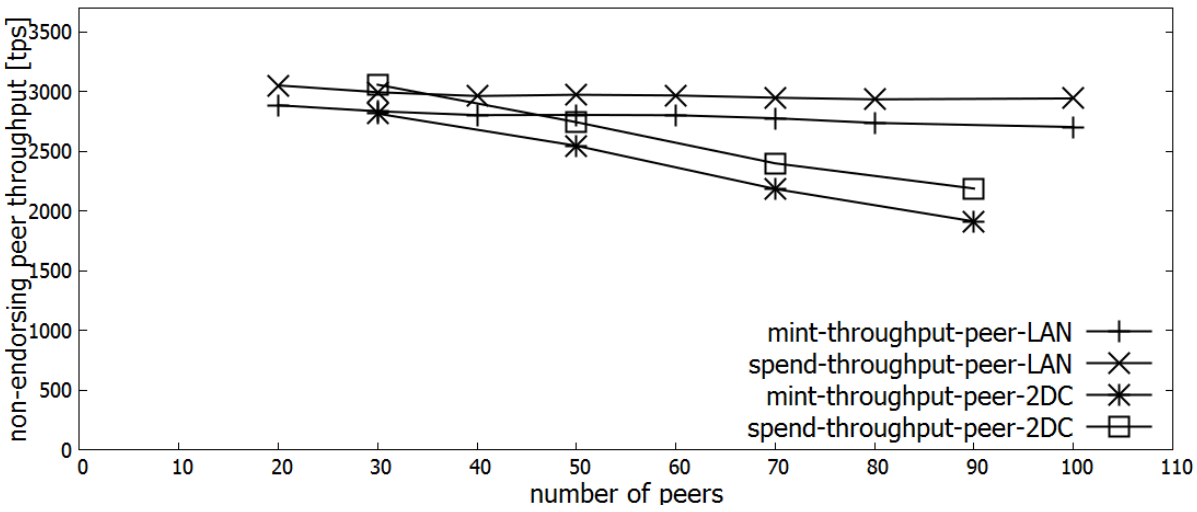


Figure A.8 Impact of varying number of peers on non-endorsing peer throughput.

We observe that the Kafka ordering service handles the added number of peers well and scales with the increase. As peers connect randomly to OSNs, the bandwidth of 3 OSNs should

become a bottleneck with the 1Gbps nominal throughput of the network. However, this does not occur. We tracked down the reason for this and found that the provisioned bandwidth in the IBM Cloud was higher than nominal one, with *netperf* reporting consistently 5-6.5Gbps between pairs of nodes.

Experiment 5: Scalability over two DCs and impact of gossip In a follow-up experiment, we moved the ordering service, 10 endorsers, and the clients to the nearby Tokyo (TK) data center, leaving the non-endorsing peers in the HK DC. The goal in this (and the next) experiment is to evaluate the system when the network bandwidth becomes the bottleneck. We varied the number of non-endorsing peers in HK from 20 to 80, maintaining direct connectivity with the ordering service (i.e., one peer per org), in addition to 10 endorsing peers in TK. The single-TCP netperf throughput reported between two VMs in TK and HK is 240 Mbps on average.

The peak throughput in function of the (total) number of peers is depicted in Fig. A.8 ("2DC" suffix). We clearly see that the throughput is basically the same as in the previous experiment with 30 peers, but it drops when the number of peers increases. The throughput is reduced since the network connections of 3 OSNs in TK are saturated. We measured 1910 tps MINT and 2190 tps SPEND throughput (at HK peers) with a total of 90 peers in this configuration.

To cope with this, and to improve the scalability over a WAN, Fabric may employ gossip (Section A.4.3). We repeated the last measurement with 80 peers in HK (totaling 90 peers) but reconfigured these peers into 8 orgs of 10 peers each. In this configuration, only one leader peer per org connects directly to the ordering service and gossips the blocks to the others in its org. This experiment (with a gossip fanout of 7) achieves 2544/2753 tps MINT/SPEND average peak throughput at HK peers, which means that gossip nicely serves its intended function. The throughput is somewhat lower than in the LAN experiment, as org leader peers (directly connected to OSNs in both experiments) now need to manage gossip as well.

Experiment 6: Performance over multiple data centers (WAN) Finally, we extend the last experiment to 5 different data centers: Tokyo (TK), Hong Kong (HK), Melbourne (ML), Sydney (SD), and Oslo (OS), with 20 peers in each data center, totaling 100 peers. As in the previous experiment, the ordering service, 10 endorsers, and all clients are in TK. We run this experiment without gossip (one peer per org) and with gossip (10 orgs of 10 peers, 2 orgs per data center, fanout 7). The results are summarized in Table A.2, averaged across peers belonging to same data center. For reference, the first row of the table shows the netperf throughput between a VM in a given data center and TK.

Table A.2 Experiment with 100 peers across 5 data centers.

	HK	ML	SD	OS
netperf to TK [Mbps]	240	98	108	54
peak MINT / SPEND throughput [tps] (without gossip)	1914 / 2048	1914 / 2048	1914 / 2048	1389 / 1838
peak MINT / SPEND throughput [tps] (with gossip)	2553 / 2762	2558 / 2763	2271 / 2409	1484 / 2013

The results again clearly show the benefits of using gossip when the peers are scattered over a WAN. We observe interesting results with the peers in OS and SD compared to HK and ML. The lower throughput with gossip for SD is due to CPU limitations of VMs in SD; with the same specification, they achieve lower validation throughput than peers in HK and ML. In OS the total throughput is much lower. The bottleneck, however, is not the bandwidth of the ordering service but the single-TCP connection bandwidth from OS to TK, as our netperf measurement suggests. Hence, the true benefits of gossip in OS cannot be observed; we attribute the slight improvement in throughput in OS in the experiment with gossip to fewer TCP connections running from OS to TK.

A.6 Applications and Use Cases

Major cloud operators already offer (or have announced) "blockchain-as-a-service" running Fabric, including Oracle, IBM, and Microsoft. Moreover, Fabric currently powers more than 400 prototypes and proofs-of-concepts of distributed ledger technology and several production systems, across different industries and use cases [176]. Examples include a food-safety network [6], cloud-service blockchain platforms for banking [96], and a digital global shipping trade [110] solution. In this section, we illustrate some real use cases where Fabric has been deployed.

Foreign exchange (FX) netting A system for bilateral payment netting of foreign exchange runs on Fabric. It uses a Fabric channel for each pair of involved client institutions for privacy. A special institution (the "settler") responsible for netting and settlement is a member of all channels and runs the ordering service. The blockchain helps to resolve trades that are not settling and maintains all the necessary information in the ledger. This data can be accessed in real time by clients and helps with liquidity, resolving disputes, reducing exposures, and minimizing credit risk [196].

Enterprise asset management (EAM) This solution tracks hardware assets as they move from manufacturing to deployment and eventually to disposal, capturing additionally licenses of software assets associated with the hardware. The blockchain records the various life-cycle events of assets and the associated evidence. The ledger serves as a transparent system of record between all participants who are involved with the asset, which improves the data quality that traditional solutions struggle with. The multi-party consortium blockchain runs among the manufacturer, shippers, receivers, customers, and the installers. It uses a 3-tiered architecture, with a user interface connecting through the Fabric client to the peers. A detailed description of the first version is available online [224].

Global cross-currency payments In collaboration with Stellar.org and KlickEx Group, IBM has operated a cross-currency payment solution since October 2017, which processes transactions among partners in the APFII organization in the Pacific region [121]. The Fabric blockchain records financial payments in the form of transactions endorsed by the participants, together with the conditions they agree on. All appropriate parties have access and insight into the clearing and settlement of financial transactions.

The solution is designed for all payment types and values, and allows financial institutions to choose the settlement network. In particular, settlement may use different methods, and Fabric makes a decision on how to settle a payment, depending on the configuration of the participants. One possible kind of settlement is through Lumens (Stellar's cryptocurrency), other ways are based on the type of the traded financial instrument.

A.7 Related Work

The architecture of Fabric resembles that of a middleware-replicated database as pioneered by Kemme and Alonso [133]. However, all existing work on this addressed only crash failures, not the setting of distributed trust that corresponds to a BFT system. For instance, a replicated database with asymmetric update processing [134, Sec. 6.3] relies on one node to execute each transaction, which would not work on a blockchain. The execute-order-validate architecture of Fabric can be interpreted as a generalization of this work to the Byzantine model, with practical applications to distributed ledgers.

Byzantium [97] and HRDB [223] are two further predecessors of Fabric from the viewpoint of BFT database replication. Byzantium allows transactions to run in parallel and uses active replication, but totally orders BEGIN and COMMIT/ROLLBACK using a BFT middleware. In its optimistic mode, every operation is coordinated by a single master replica; if the master is suspected to be Byzantine, all replicas execute the transaction operations for the master and it triggers a costly protocol to change the master. HRDB relies in an even stronger way on a correct master. In contrast to Fabric, both systems use active replication, cannot handle a

flexible trust model, and rely on deterministic operations.

In *Eve* [129] a related architecture for BFT SMR has also been explored. Its peers execute transactions concurrently and then verify that they all reach the same output state, using a consensus protocol. If the states diverge, they roll back and execute operations sequentially. Eve contains the element of independent execution, which also exists in Fabric, but offers none of its other features.

A large number of distributed ledger platforms in the permissioned model have come out recently, which makes it impossible to compare to all (some prominent ones are Tendermint [145], Quorum [183], Chain Core [124], Multichain [165], Hyperledger Sawtooth [175], the Volt proposal [197], and more, see references in recent overviews [44, 72]). All platforms follow the order-execute architecture, as discussed in Section A.2. As a representative example, take the Quorum platform [184], an enterprise-focused version of Ethereum. With its consensus based on *Raft* [170], it disseminates a transaction to all peers using gossip and the Raft leader (called *minter*) assembles valid transactions to a block, and distributes this using Raft. All peers execute the transaction in the order decided by the leader. Therefore it suffers from the limitations mentioned in Section A.1 and Section A.2.

A.8 Conclusions

Fabric is a modular and extensible distributed operating system for running permissioned blockchains. It introduces a novel architecture that separates transaction execution from consensus and enables policy-based endorsement and that is reminiscent of middleware-replicated databases.

Through its modularity, Fabric is well-suited for many further improvements and investigations. Future work will address (1) performance by exploring benchmarks and optimizations, (2) scalability to large deployments, (3) consistency guarantees and more general data models, (4) other resilience guarantees through different consensus protocols, (5) privacy and confidentiality for transactions and ledger data through cryptographic techniques, and much

more.

Appendix B

Acronyms

Sorted alphabetically.

ACL Access Control Layer

AMI Advanced Metering Infrastructure

API Application Programming Interface

AUC Average Unit Cost

BFT Byzantine Fault Tolerance

BGP Border Gateway Protocol

BLE Bluetooth Low Energy

BMG Brooklyn Microgrid

BP Bid Phase

CA Certification Authority

CB Central Bank

CFT Crash Fault Tolerance

CLI Command-Line Interface

CPU Central Processing Unit

CRDT Conflict-Free Replicated Data type

DAM Day-Ahead Market

DC Datacenter

DER Distributed Energy Resource

DoS Denial of Service

DSO Distribution System Operator

E2E End-to-End

EAM Enterprise Asset Management

ECDSA Elliptic Curve Digital Signature Algorithm

EIA Energy Information Administration

EIM Earnings Impact Mechanism

ERCOT Electric Reliability Council of Texas

ESCC Endorsement System Chaincode

ESCO Energy Service Company

ETL Extract, Transform, and Load

EV Electric Vehicle

FCFS First-Come, First-Served

FX Foreign Exchange

GSM Global System for Mobile communications

GSP Grid Service Provider

HDIAC Homeland Defense and Security Information Analysis Center

ICT Information and Communications Technology

IoT Internet of Things

ISO Independent System Operator

KVS Key-Value Store

KYC Know Your Customer

LAMP Landau Microgrid Project

LAN Local Area Network

LCOE Levelized Cost of Energy

LEM Local energy market

LTE Long-Term Evolution

M2M Machine-to-Machine

MCP Market-Clearing Price

MSP Membership Service Provider

MVCC Multiversion Concurrency Control

OSN Ordering Service Node

PBFT Practical Byzantine Fault Tolerance

PBR Performance-Based Regulation

PoW Proof-of-Work

PoS Proof-of-Stake

PPU Price Per Unit

PSC Public Service Commission

PTM Peer Transaction Manager

PUC Public Utilities Commission

PV Photovoltaic

Q# Quarter #

RAF Resource Allocation Factor

REV Reforming the Energy Vision

RP Reveal Phase

RTM Real-Time Market

SAIDI System Average Interruption Duration Index

SAIFI System Average Frequency Duration Index

SLA Service-Level Agreement

SMR State-Machine Replication

SPP Solar Partner Program

TE Transactive Energy

TES Transactive Energy System

TLS Transport Layer Security

TMP Transaction Management Platform

TPS Transactions Per Second

TTP Trusted Third Party

US United States

USD US Dollar

UTXO Unspent Transaction Output

UUID Unique Identification Number

vCPU Virtual Central Processing Unit

VSCC Validation System Chaincode

VM Virtual Machine

VOST Value of Solar Tariff

WAN Wide Area Network