Resolving Challenges in Multi-Stakeholder CPS Using Distributed Ledgers

By

Scott Eisele

Dissertation

Submitted to the Faculty of the

Graduate School of Vanderbilt University

in partial fulfillment of the requirements

for the degree of

DOCTOR OF PHILOSOPHY

in

Electrical Engineering

Dec 12, 2020

Nashville, Tennessee

Approved:
Abhishek Dubey, Ph.D.
Gabor Karsai, Ph.D.
Ted Bapty, Ph.D.
Aron Laszka, Ph.D.
Janos Sztipanovits, Ph.D.
Marisol Garcia Valls, Ph.D.

ACKNOWLEDGMENTS

This research was supported in part by the National Institute of Standards and Technology, under grant 70NANB18H198, the Advanced Research Projects Agency-Energy (ARPA-E), U.S. Department of Energy, under Award Number DE-AR0000666, a grant from Siemens Corporate Technology, and the National Science Foundation under award numbers CNS-1647015, CNS-1818901, CNS1840052, and DE-IA0000025.

I would like to express my gratitude to my advisor Dr. Abhishek Dubey, for the encouragement, and support that he has provided as a mentor for the last several years. I would also like to thank Dr. Aron Laszka and his students, Taha Eghtesad and Nicholas Troutman for their collaboration, guidance and insights over the course of this work. I am grateful to Dr. Ted Bapty for his encouragement and advice. I am also grateful to Dr. Gabor Karsai for the opportunities to work on interesting projects. Thank you also to my other committee members Dr. Janos Sztipanovits and Dr. Marisol García Valls for your support and insights into this work.

I am also grateful to Martin Lehofer and Dr. Karla Kvaternik for the opportunity to collaborate with them at Siemens Corporate Technology. The work there provided the seeds from which this work grew.

I am grateful to my friends in the SCOPE-Lab: Geoffrey Pettet, Shreyas Ramakrishna, Michael Wilbur for the many discussions, collaborations, and moral support, as well as Dr. Ayan Mukhopadhyay, Keegan Campanelli, Purboday Ghosh, Dr. Carlos Barreto for the opportunities to work with them.

I am especially grateful to my family: to my mother Linda Eisele for her love and support; to my father, Fred Eisele, for the many discussions and the insights he provided; and to my wife, Brynna Eisele–for bearing this endeavor with me. I can not imagine a better companion.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Background and Introduction

1.1 Cyber-Physical Systems

Cyber-Physical Systems (CPS) consist of devices that integrate physical and computational processes. While the integration of physical and computation processes has itself been in existence for quite some time in the form of embedded systems[1], those are typically designed as isolated systems. In contrast, the vision of CPS is that the devices are open and networked, allowing them to communicate and interact [2]. Allowing communication and interaction across devices allows for the development of new systems with capacities that extend beyond those of isolated embedded systems.

An example of CPS can be seen in Fig. 1.1 which shows a traditional power grid consisting of the power plant, transmission lines, substations, and meters. These systems rely on distributed and heterogeneous devices, some of which may have some combination of sensors, actuators, computing or storage resources, and communication capabilities. Thus, applications that use these devices must be built to rely on loosely connected components that run on different processors.

In addition to fulfilling their intended functional purposes, CPS systems must satisfy the following non-functional requirements:

- **Safety and Security**: Safety is the freedom from unacceptable loss [3]. This archetypally includes loss of life and damage to property; in the context of CPS, for example, this can include damage to users' hardware or other belongings. Security, at least in the field of information resources, has been defined as having three components: "confidentiality (protection against disclosure to unauthorized individuals), integrity (protection against alteration or corruption), and availability (protection against inter-

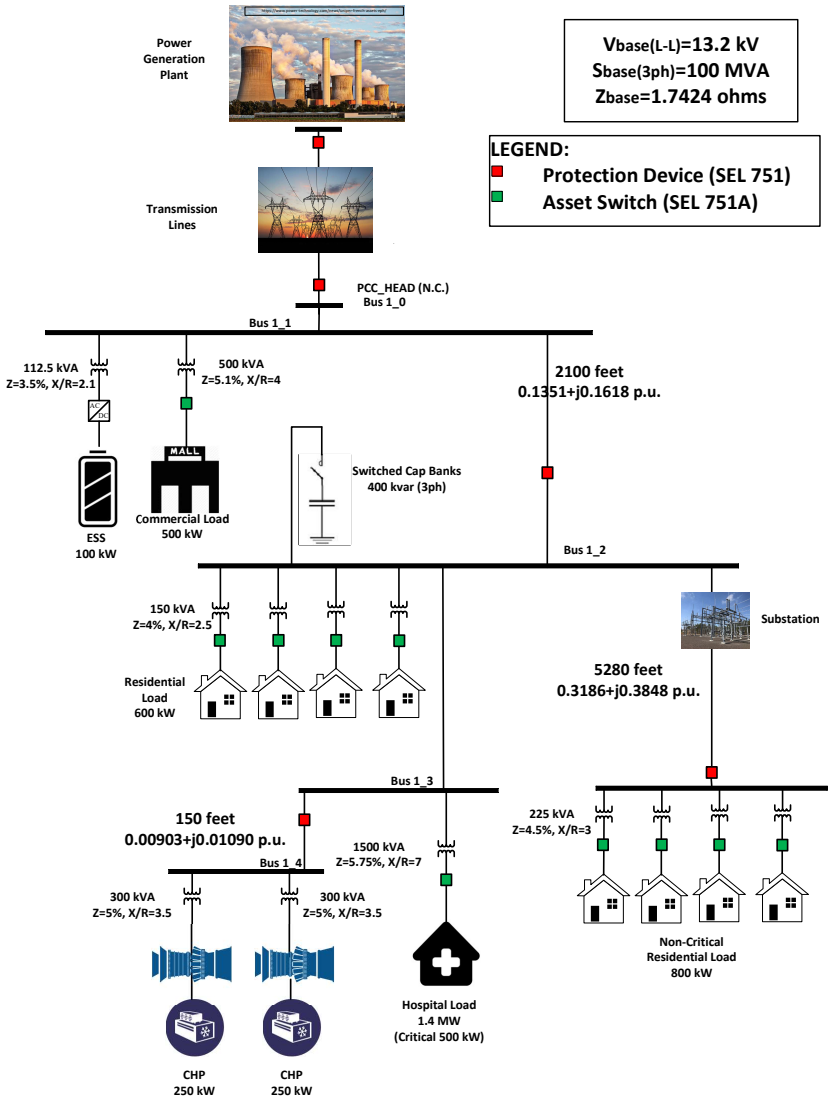


Figure 1.1: Vertically integrated electric utility. A single company owns all the devices, from the power generation, to the meters at the homes.

ference with the means to access the resources)" [4]. To accommodate this definition to cover CPS, we must consider not only information security but also hardware security. Since the hardware is directly controllable via software, we require physical and cyber access control. We must also still preserve integrity by accounting for changes that can be made to the hardware physically or via software by those who have access. Finally, to preserve availability we must consider the consequences of losing access to our communication with the hardware.

- **Resilience and Reliability**: Resilience is the ability to withstand and recover from failures, regardless of whether the cause is a fault or an attack. Traditional CPS systems typically assume that the agents within the system are honest and only consider crash failures [5][6]. Some distributed CPS also consider Byzantine failure [7] for cases when agents may be compromised (as in the case of an attack), but if an agent is not compromised it is presumed honest. Reliability is the ability of a system to consistently do what it is intended to do. Reliability is achieved in part through security (preventing failures) but also by resilience (withstanding and recovering from failures). The reliability of a system can be measured with respect to the number of failures it can sustain before violating some invariant. Systems are designed to withstand some number of failures. For example, power systems are designed to handle the failure of any single component [8].

- **Efficiency and Scalability**: Efficiency, generally speaking, is minimizing the resources required to fulfill the other requirements.

  Efficiency is generally cost-motivated and introduces constraints to the system design. For example, providing reliability may require redundant systems, potentially using replicas. However, there are costs associated with each replica, leading the designer to consider potential trade-offs. Coulouris *et al.* [4] have described a system as being scalable "if it will remain effective when there is a significant increase in the

number of resources and the number of users".

Many of the most interesting applications of CPS such as smart cities and networked autonomous vehicles need to be able to function if there is a large influx of users.

This list is not exhaustive, however, it does provide criteria for evaluating CPS solutions. These requirements must be satisfied even if some of the system agents are malicious, compromised, or faulty.

1.2 The Concept of Multiple Stakeholders

Many distributed CPS are designed, owned, and operated by a single stakeholder, or a few collaborating stakeholders where cyber and physical components are developed jointly. However, there is growing interest in developing CPS applications that can be constructed *ad hoc* from existing devices as applications become relevant. For much of the history of the United States, for example, power was generated, harnessed, and provided to the public by individual utility companies that provided power to their respective regions, operating as a distributed CPS (see Fig. 1.1). Over time, however, the provision of power to the public has become spread across interacting companies that coordinate and operate power grids. In recent years, new technologies (e.g. solar panels) have allowed an ever-expanding number of people to operate their sources of energy. Power systems now include these members of the public who are in theory capable of trading their energy resources via novel *transactive energy systems* (TES). There is a potential for nearly unlimited participants in TES, and each may have unique goals and motives when it comes to energy production and trade.

TES are an example of industrial MSCPS that will generate massive amounts of data. Other such applications appear in systems such as smart cities and networked autonomous vehicles which consist of many distinct users. The data can be used to, over time, train and develop algorithms for maintaining the system - for example the algorithms to control the

prosumer trading strategy or identifying the criteria for raising alarms. These data-driven algorithms are resource-intensive and generate data in volumes and rates that are typically only seen in data centers. Due to the volume, the networks potentially connecting these applications to data centers do not have sufficient bandwidth to provide timely analysis. This has resulted in interest among the community in turning toward edge computing systems for these purposes.

Edge-computation (also referred to as edge-cloud computation) refers to computations carried out on various distributed nodes as opposed to central cloud servers. These nodes may be centrally managed, but many independently owned devices (including those which comprise the Internet of Things, or IoT) have computation capabilities. These devices are often underutilized, and there is interest in opportunistically harnessing their compute power. As the IoT and the number of devices with processing capability rapidly expands, necessary computations can be distributed among local, decentralized networks of devices instead of being sent to centrally managed cloud servers [9]. Owners of these devices, however, may have inherently distinct interests and priorities.

Edge-cloud computation can be utilized in TES and other industrial IoT systems. The correctness of operation of the TES, in this case, depends upon the correctness of results and timeliness of analysis obtained from the edge cloud.

We refer to these kinds of systems (both TES and the edge-cloud system used to deploy industrial IoT applications) as *multi-stakeholder cyber-physical systems* (MSCPS), and they add an additional layer of complexity due to potentially conflicting objectives of the participants and the mutual distrust that consequently may exist between them.

1.2.1 Challenges in MSCPS

The overarching challenge unique to MSCPS is the coordination between stakeholders. The reason this is so challenging is because to coordinate, the agents comprising the system must share information by sending messages to one another. How do we know that

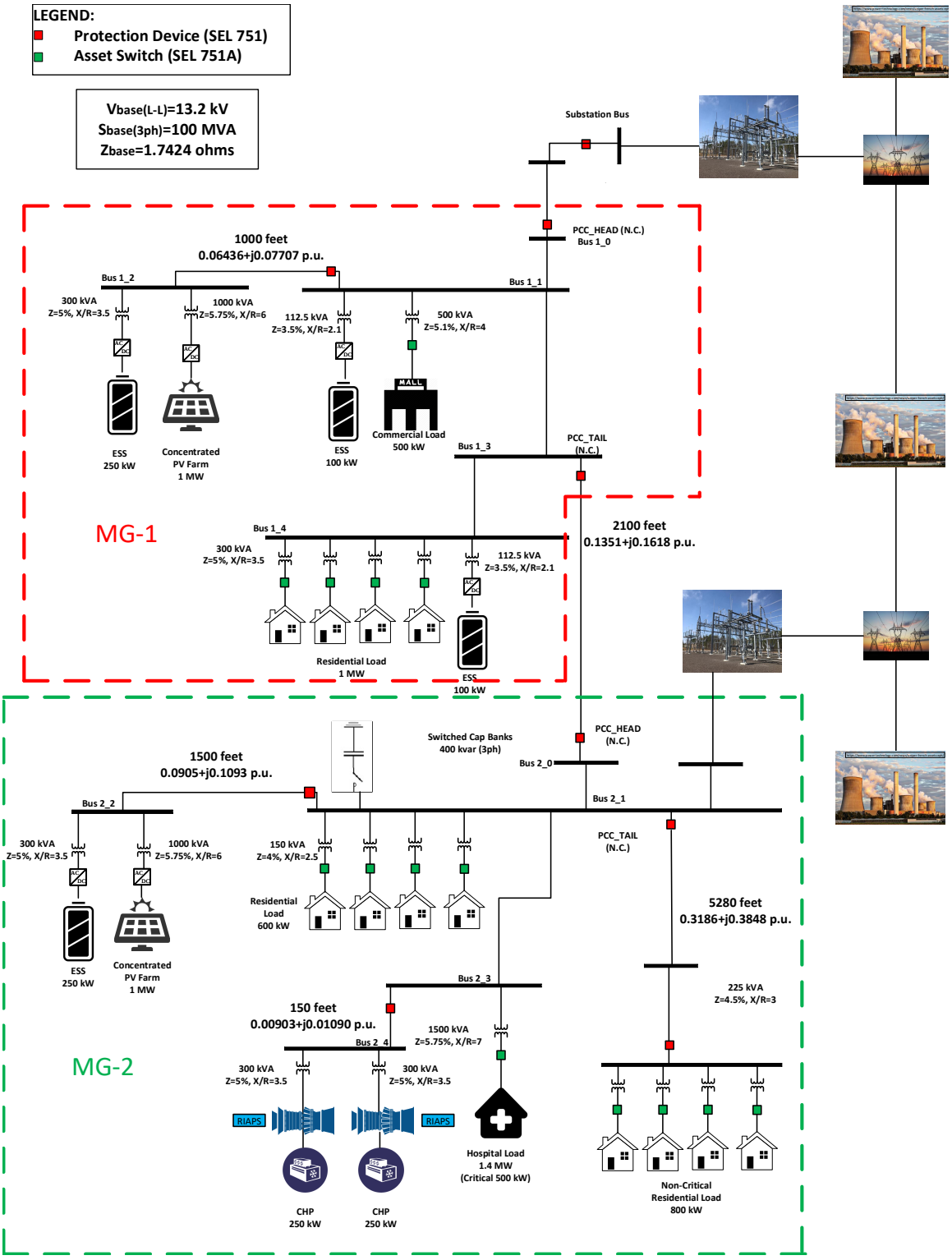


Figure 1.2: Distributed power grid with multiple stakeholders, including homeowners that coordinate via energy markets.

messages are being sent and received? Since the stakeholders cannot be trusted, can the messages be trusted? These questions force us to consider the following core questions:

- How will the agents be able to reach consensus? Will the agents follow through with any agreed-upon actions?
- How do we cope with the needs of agents to maintain individual privacy?
- How do we ensure that the system is accomplishing its goals?

We discuss these problems and other important considerations below.

Trust. For agents to coordinate appropriately, they must exhibit some degree of *trust* in the system and be able to reach a *consensus* about what actions to take. In the example of TES, the participants must agree on how much energy each will produce or consume in order to ensure that they are balanced. Balance is critical because an imbalance can shift the frequency of the AC power; this changes the behavior of the devices connected to the power line and could cause them to operate faster or generate more heat. If the error is sufficiently large and persists long enough, at some point it will damage the physical system and compromise safety. Many works have investigated ways to achieve consensus across agents in MSCPS[10]. Consensus generally requires information-sharing across devices, but information-sharing can fail in indistinguishable ways. For example, if a message sent by one device does not arrive, does that mean the device has failed, or the network failed? Or does it mean that the message has simply not yet arrived? This challenge is exacerbated by conflicting objectives among stakeholders who may withhold messages or send conflicting messages. Determining how to prevent, detect, and cope with resultant failures generally requires us to make assumptions about the intentions of the stakeholders.

- If we assume all agents are honest, then failures to achieve consensus are a result of system failures (including crashes, corrupted messages, and network failures). Consensus protocols have been developed specifically to cope with these types of issues [11, 12, 13].

- If we instead assume agents are rational or malicious, then they may share false information or lie about actions taken. This requires the use of consensus protocols that are Byzantine fault-tolerant (BFT) [7, 14, 15]. Some additional BFT protocols that use game theoretical techniques [16, 17] are feasible for rational agents since such agents will do what benefits them most.

- In systems that rely on incentivizing agents to achieve consensus, a means of detecting misbehavior is essential. One way this can be achieved is by keeping records of the stated intentions of the participants so that they can be audited through comparison against actual events (a feature known as *auditability*).

The methods employed to facilitate consensus must therefore consider the goals of the system overall, its agents, and its applications when determining what considerations must be in place to ensure consensus is possible. To preserve trust in the system, there must also exist mechanisms to *verify* that the agents are cooperating and that the system is indeed operating correctly.

Privacy. Importantly, the need to share information across nodes can compromise the privacy of participants using the system. Since these systems are intrinsically integrated with humans, privacy preservation is an important requirement. *Privacy* is a subtype of safety, where an agent is free from unwanted observation and unwarranted interference is limited. Privacy is critical in TES, for example, because energy usage patterns can be used to identify system participants and learn personal information about them. These issues could be accounted for, for example, by obscuring participants' identities or by obscuring data regarding their transactions.

There has been extensive work on ways to preserve privacy in distributed systems. *Differential privacy* protocols add noise to communicated data so information can be gathered without learning private information about individuals. Encryption schemes, including *homomorphic encryption* [18, 19, 20], allows computations to be performed on encrypted data while minimizing access to decryption keys. *Mixing* [21, 22, 23] is an approach that hides

the identities of the participants within a group and allows the data needed for consensus to be shared as plaintext. Many privacy protocols also implement *multi-party computation* (MPC) [24] which allows a group of agents to collectively compute a result without sharing their private inputs. These various approaches require us to make certain considerations. For example, you cannot augment privacy without considering the effects it will have on other important properties of an MSCPS, including the ease with which it can communicate to reach consensus across agents, its efficiency of operation, and the overall safety of the system. For example, consensus protocols run more efficiently when agents are working with unencrypted information [25]. This raises important questions, such as: how do we manage a need for privacy with the need to achieve consensus? Implementation of privacy techniques also impairs how efficiently the system operates. How do we balance the trade-off between preserving privacy and maintaining efficiency? Additionally, privacy techniques must take into consideration the need to keep certain information accessible so that system safety can be monitored.

Integration and Correctness. MSCPS can be comprised of many heterogeneous devices. TES, for example, consists of power plants, transmission lines, substations, smart meters, solar panels, etc. These devices may be legacy devices that were part of existing power grids, or they may be newer devices capable of running more state-of-the-art technologies. Many MSCPS are also open systems, which need to be able to adapt to agents entering and exiting the system (scalability). In the example of transactive energy, an individual may acquire the capacity to generate power and wish to be added to an existing energy trading system. These *integrative* challenges contribute to MSCPS being complex to design, build, and manage. To cope with these complexities, middleware platforms for CPS [26] have been developed to provide an abstraction layer that enables applications to be developed without having to worry about the details of a secure communication protocol, for example. To ensure an effective MSCPS, we must also ensure *correctness* of both the applications and the middleware they are built upon. Doing so again requires the use

of *verification* mechanisms, which often utilize models or formal mathematical verification methods [27, 28]. The solutions and results produced by the system applications must be verified as well, and we must be able to identify and track any system failures so they can be appropriately addressed.

1.2.2 Role of Distributed Ledgers in MSCPS

To address these problems in MSCPS there has been intense interest in distributed ledger technologies (DLT)[29].

DLT refers to a family of protocols that allow mutually distrustful parties to achieve consensus in the absence of a centralized authority. These protocols are distributed across many stakeholders which each maintain a copy of a ledger, which is an immutable, append-only record of communications or transactions within the system. The permanence of this ledger makes the transactions auditable, which provides a mechanism to ensure system security by checking for dishonest behavior that would disrupt the ability to achieve correct results and consensus without a central authority. The maintenance of a single replicated ledger across distributed agents allows the system to be scaled to accommodate entering and exiting agents.

The prototypical example of DLT is Bitcoin[16]. DLT gained popularity because of the success of Bitcoin, which uses a blockchain data structure, which is a specific type of distributed ledger. Bitcoin is a platform that was designed to enable financial transactions without a trusted third party. To achieve this, Bitcoin implemented an incentive-compatible game to determine who has write-access (the ability to make updates) to a ledger via a proof-of-work consensus algorithm. The result is an open ledger that is effectively immutable (due to prohibitively high cost and complexity of tampering with data in the ledger), auditable (since all transactions are publicly accessible), and which accurately records transactions without reliance upon a central authority. The developers of Ethereum [30][31] recognized that a platform like Bitcoin could be used for more than just trans-

ferring tokens/currency between mistrusting parties since distributed ledgers can be used to model state machines. They introduced the concept of smart contracts, which are programs that are appended to the ledger and can later be executed with the same guarantees as manual transactions. Smart contracts greatly expand the capabilities of distributed ledgers allowing them to perform trusted computations, again without a trusted central authority. Though it is clear that distributed ledgers can benefit MSCPS systems particularly in achieving consensus, non-repudiation, and correctness of computation, DLT does not simply solve all the major problems in MSCPS. The application of DLT to MSCPS opens up some additional issues we must consider, especially with respect to privacy and integration (*e.g.*, real-time operation).

1.3 Motivating Research Questions

The overarching question motivating the work in this dissertation is how can DLT be used to address the problems in MSCPS, and what additional challenges does this introduce? I confront this issue by specifically addressing the following questions:

1. **In MSCPS that require non-repudiation to incentivize correct operation, how do we preserve safety while enabling privacy?** In TES, energy producers and consumers submit offers to trade energy. If the trades in the system are not balanced, or exceed the safety constraints of the system, then they can cause physical damage. Safe trades can be incentivized by recording trades and fining prosumers for deviating from them. However, this requires knowledge about which prosumer has made a given offer, which could violate privacy concerns. I, along with collaborators, have aimed to develop a trading platform based on DLT that enables a safe and private energy trading market.

2. **In a system with multiple stakeholders, how can trust and consensus be achieved efficiently?** In an edge-cloud system used to deploy CPS applications, since the

participants are mutually mistrusting, we cannot implicitly trust that a computation provider will execute the computation correctly. There are ways to check the correctness of results, but the costs of ensuring correctness need to be competitive with cloud computation costs for an edge-cloud to be viable. Thus a focus has aimed to explore how DLT can be used to implement a secure edge-cloud marketplace that enables complex computations while preserving efficiency.

3. **Can we make it easier to make well-integrated MSCPS applications? What features does this require?** The Android operating system provides many services and features that facilitate the development of phone applications. Similarly, MSCPS applications require many services that are difficult to implement correctly, and so would benefit from a platform to aid in application development. DLT can be used to provide many of these services, but there are other services that DLT cannot provide. Additionally, well-integrated MSCPS must include measures to ensure that the systems are constructed and are operating correctly. We have worked to develop a platform that provides these additional services and can integrate with DLT and thus support MSCPS applications.

1.4 Outline and Scope of Dissertation

The remainder of this dissertation describes my efforts to answer these questions and my contributions in this field. Primarily, I describe the development of two applications (for trading energy and for trading computational capacity) which are intended to address some of the aforementioned challenges of MSCPS.

In Chapter 2 a decentralized platform designed to enable a forward-trading peer-to-peer transactive energy market that ensures trading safety while preserving anonymity called TRANSAX is presented.

MODiCuM (Mechanisms for Outsourcing Computation via a Decentralized Market) aims to enable a market for opportunistic utilization of idle compute resources at the edge

Table 1.1: Summary of Key Publications Addressing the Research Questions

Question	Relevant Publications
Q1	[32, 33]
Q2	[34, 35, 36]
Q3	[37, 38, 39]

(Chapter 3) It takes a minimalist approach to consensus, by assuming that only a task-giver needs to be convinced of the correctness of a computation. MODiCuM provides probabilistic assurance that jobs are executed correctly. Our initial implementation of MODiCuM was limited to *batch* (or *offline*) jobs, which were isolated from outside systems. Recognizing the need to support *streaming* (or *online*) computations to access the potential of MSCPS systems we built upon the principles established in MODiCuM to construct a market that supports trusted outsourcing of online computation (Chapter 4). This is an essential preliminary step for industrial internet applications that have soft real-time requirements.

In Chapters 5 and 6 we describe elements necessary to support general MSCPS applications. Specifically, since MSCPS must be able to interface with heterogeneous devices Chapter 5 we utilize a middleware to abstract away high complexity details associated with "infrastructure protocols". Then in Chapter 6 we build upon that foundation and construct a generalized platform for arbitrating resource consumption across different domains. This blockchain-based platform, SolidWorx, provides key design patterns for enabling resource exchange, including a hybrid solver architecture.

The specific contributions of this work, in addition to the development of these applications, include:

- An analysis and design of privacy groups that can achieve k-anonymity while maintaining safety in MSCPS - Some applications require data in order to take control actions to keep the system safe, but the sharing of that information may violate user privacy. By allocating an in-network currency (representing safe trading limits based on physical system constraints) and using a mixing protocol, we demonstrated an

ability to provide k-anonymity to participants in a transactive market. ([36])

- An analysis of a verifier strategy for MSCPS - As part of building an incentive-compatible market for outsourcing computation, we performed a game-theoretic analysis of participant strategies to verify that a rational participant would behave correctly with overwhelming probability. ([32])

- Integration of a blockchain-based distributed ledger and a production-grade streaming middleware to enable the trusted outsourcing of stream computations. The two tools are integrated through the construction of a protocol that enforces a game that verifies participant outputs to ensure that rational participants will behave in expected ways. ([33])

- Generalization of the solution using Middleware and a general market Platform - To generalize the solutions developed in this dissertation we focused on the development of a) a connection substrate [38] and integrating it with the distributed ledger and b) design of a generalized resource sharing market platform called SolidWorx. ([37])

- A hybrid solver pattern - In systems requiring complex computations, there may exist a computationally limited but secure compute resource, as well as a computationally powerful but non-secure resource. We have developed a hybrid solver pattern that enables the system to execute computation with the non-secure resource and use the secure resource to verify the result. ([37])

The remainder of this dissertation proceeds as follows: In Chapter 2, we describe the development of the TRANSAX platform and its major components and how this application addresses issues of privacy and efficiency in the MSCPS of transactive energy systems. In Chapter 3 we discuss the MODiCuM platform and describe how it addresses integrative issues and verifies the correctness of computations in a distributed computing marketplace. In Chapter 4, we describe the extension of MODiCuM to online (streaming) applications.

In Chapters 5 and 6, we discuss the integration with RIAPS and the SolidWorx platform, including the hybrid solver. Finally, in Chapter 7 the dissertation concludes with a discussion and possible future work in this field.

Chapter 2

Privacy and Resilience Through Blockchains For Energy Systems

2.1 Overview

Privacy describes a state of being free from unwanted interference. MSCPS, which access and process information from many sources and users, are typically only attractive to users if they can guarantee some degree of privacy when it comes to the use of participants' information.

When considering privacy, though, we must also consider safety. Privacy necessitates hiding information, however safety requires information. This apparent conflict comes up frequently as a concept in the role of government. Governments want to monitor more aspects of their populace to improve safety, but there is always a concern that if private data is not carefully protected then it can be used to harm its owner[1]. Examples of such harms include identity theft or using private information to target a specific group of people.

Privacy and safety are both critical in transactive energy systems (TES). If the transactions in a community are public, the energy consumption patterns of a household can be exploited to harm participants. Therefore, the amount of energy produced, consumed, bought, or sold by any prosumer should be anonymous to other prosumers and perhaps only shared confidentially with the distributed system operator (DSO) when paying the monthly bill. However, the system must also prevent negligent or malicious trading from endangering the stability or physical safety of the microgrid by rejecting trades that are unlikely to be delivered or that would violate line capacity constraints. Ensuring that the system operates appropriately and within these constraints therefore requires information about the participants' actions and locations within a network.

[1] https://www.newyorker.com/magazine/2018/06/18/why-do-we-care-so-much-about-privacy

There are a variety of tools that have been used in conjunction with DLT to enhance privacy in distributed systems. Examination of these techniques makes it apparent that degrees of privacy can be assured at the expense of system efficiency. The eminent challenge is to achieve sufficient levels of privacy and safety without sacrificing too much efficiency [40].

Achieving this goal requires us to determine specifically what information should be kept private. In the case of TES, we must be able to provide privacy of communication. Without this, an adversary can discern who is making a function call or sending a message over a network based on the sender's MAC address, IP address, or route to the destination. However, communication confidentiality is not sufficient for anonymous trading, as the accounts used to make trades must also not be associated with their owners. We must therefore anonymize individual users' identities.

To satisfy the seemingly conflicting requirements of privacy, efficiency, and safety in MSCPS, we have developed a platform for transactive energy system microgrids which trades efficiency for safety and privacy. To demonstrate the feasibility of our platform, we perform experiments with dozens of embedded devices and energy production and consumption profiles from a real dataset.

The works comprising this chapter have been accepted for publication in Transactions on Cyber-Physical Systems (TCPS) [36] and have been published in IEEE Computer [35]. This builds upon [34] which was published in the 24th IEEE International Conference on Parallel and Distributed Systems (ICPADS).

- S. Eisele, T. Eghtesad, K. Campanelli, P. Agrawal, A. Laszka, and A. Duby, "Safe and Private Forward-Trading Platform for Transactive Microgrids," 2020, Accepted, Pending Publication.

- Scott Eisele, Carlos Barreto, Abhishek Dubey, Xenofon Koutsoukos, Taha Eghtesad, Aron Laszka, and Anastasia Mavridou. Blockchains for Transactive Energy Systems: Opportunities, Challenges, and Approaches. Computer, 53(9):66–76, 2020

- A. Laszka, S. Eisele, A. Dubey, G. Karsai, and K. Kvaternik, "Transax: A blockchain-based decentralized forward-trading energy exchange for transactive microgrids," in Proceedings of the 24th IEEE International Conference on Parallel and Distributed Systems (ICPADS), 2018, pp. 918–927.

2.2 Introduction

The traditional setup of the power grid is rapidly changing. Solar panel capacity is estimated to grow from 4% in 2015 to 29% in 2040 [41], and with the decreasing costs of battery technology, it is becoming increasingly feasible to support almost 99% of the total load with renewable sources, by balancing out the intermittence with batteries [42]. These changes are also leading to the development of a decentralized vision for the future of power-grid operations, in which local peer-to-peer energy trading within microgrids can be used to both reduce the load on distribution system operators (DSO) and help them plan better, leading to the development of transactive energy systems (TES) [43, 44, 45, 46]. A transactive energy system is a set of market-based constructs for dynamically balancing the demand and supply across the electrical infrastructure [46]. In this approach, customers connected by transmission and distribution lines can participate in an open market, trading and exchanging energy locally. Customers participating in these markets are known as *prosumers*. There are typically three phases in the operation of this market: posting offers to buy or sell energy, matching selling offers with buying offers, and synchronized energy transfer to and from the grid.

In theory, these interactions could happen in a centralized manner by communicating all offers to a centralized market, which would match the offers and broadcast the trades back to the individual prosumers. However, in a centralized solution [47], the market presents a single point of failure.

In the last decade, there has been an emphasis on decentralizing the operations of electrical power grids [48] due to their vulnerability to natural disasters, such as Hurricane

Maria, and cyber threats, such as the Ukraine power grid attack. In the absence of centralized control, the "prosumers" (customers with both electrical energy production and consumption capability) can collaborate to dynamically balance the demand and supply across their microgrid, improving system reliability. However, this requires a financial market at the distribution level, where participants can trade energy assets. It also requires control strategies to keep local energy sources stable due to the low system inertia compared to a conventional grid [49]. This is the main concept behind transactive energy systems (TES) [44].

Prosumers that change consumption (demand response) as part of market-based transactive control were demonstrated in the Olympic Peninsula Project [50] in 2006. Both local production and consumption in a limited "transactive" system were demonstrated by the LO3 project in Brooklyn [51]. There are ongoing studies, such as the work done by Wörner et al. [52] in a town in Switzerland.

However, large-scale deployments are still missing. The primary reason for this is the complexity of integration between financial markets, predictive algorithms, information platforms, and physical control. While the research community has made progress in managing the control of the system [53] and developing predictive algorithms [54], the integration with a decentralized information architecture and market remains a challenge due to problems of trust, correctness, and privacy.

Our research team—and several other teams as shown by a recent survey [55]— proposed addressing the challenges of trust in TES through the use of blockchains. The motivation behind this is in part due to the success of Bitcoin, a prototypical example application of blockchains. Bitcoin stores transactions in a public distributed ledger, which is called a blockchain because the records are stored in blocks that are cryptographically linked to previous blocks, forming a chain. Any entity can read the ledger; however, to append a new block to the ledger, the Bitcoin network uses a probabilistic consensus protocol based on *proof-of-work* (PoW). This consensus protocol solves both trust and fault-tolerance is-

sues since the majority of participants will reach consensus on the ledger state. Further, it provides censorship-resistant, immutable, tamper-proof, and transparent transactions, thus enabling trusted transactions without a trusted third party. Enabling trusted transactions without a trusted third-party is a crucial factor for TES. Some blockchain implementations also enable participants to implement *smart contracts*—programs that are stored and executed by the blockchain network, benefiting from its trust properties.

While the idea of integrating blockchains into TES is conceptually appealing, several challenges must be addressed before protocols and implementations can live up to their potential. The outline for this article is as follows: first, we describe several of the key challenges which prevent the widespread adoption of decentralized TES. Then, we present TRANSAX, our solution for implementing blockchain-based TES, and show how it addresses these challenges.

2.3 Challenges for Blockchains in TES

The key challenges of using blockchains in transactive energy systems can be summarized as (a) code complexity and immutability; (b) privacy issues; (c) high computation costs, especially when trying to process complex market operations through smart contracts; (d) integration challenges due to a lack of suitable patterns to interact with physical devices and to ensure time synchronization; and (e) security concerns of blockchain-based systems. Table 2.1 summarizes these challenges and how we address them.

Code Complexity and Immutable Bugs Coding errors frequently occur due to incorrect assumptions about the execution semantics of smart contracts [58]. For example, Luu et al. [59] analyzed 19,366 smart contracts and found that 8,833 contracts had one or more security issues. These errors can result in devastating security incidents, such as the "DAO attack," where $50 million in cryptocurrency was stolen, and the multi-signature Parity Wallet library hack, where $280 million in cryptocurrency was lost.

Blockchain-based platforms are designed to provide immutability, which prevents patch-

Table 2.1: Summary of challenges integrating blockchain technologies with power systems and our relevant contributions.

Challenge	Description	Contributions
Immutable bugs	Blockchains' design guarantees immutability; however, this means bugs are also immutable	Build and verify smart contracts using VeriSolid [56]
Efficiency	Smart contracts require all verifier nodes to replicate the computations in a transaction	Limit the computations executed on the smart contract to checking correctness
Integration	Existing power grid equipment does not have the capabilities for managing a distributed set of blockchain nodes integrated with the power equipment	Use the middleware services (time synchronization, discovery) of RIAPS for integration [57]
Privacy	Transaction details can be open and attributable to prosumers	Energy assets, cryptographic mixing, and groups to provide k-anonymity to prosumers while ensuring feeder level safety
Cybersecurity	Although blockchains protect against some attacks, adversaries can compromise information before it is processed by the blockchain	Design policies to mitigate attacks (future work)

ing smart contracts or reverting malicious transactions. Developers can work around this by separating the code into distinct contracts, a "frontend" and a "backend," where the frontend references the backend library. Then, to change the functionality of the frontend, developers can simply change the reference to point to a new version of the backend. However, this can also erode trust since a contract may be changed and no longer satisfy its original terms. In more extreme cases, transactions can be reverted via a hard fork, but this requires the consensus of all the stakeholders and introduces security issues such as replay attacks.

To tackle these security risks and vulnerabilities in TRANSAX, we use formal methods developed by our team to generate code from the high-level, graphical, and FSM-based language to low-level smart contract code. Rooting the whole process in rigorous semantics allows the integration of formal analysis tools, which can be used to verify safety and

security properties, thereby enabling the development of correct-by-design smart contracts.

Computational Efficiency Smart contracts are not suitable for executing complex market mechanisms, because the majority of verifier nodes responsible for verifying the computation in a given transaction must perform the computation to ensure correct execution, making computations very costly. This is sometimes referred to as *on-chain* computation. To limit the potential for abuse of the network, Ethereum sets an upper bound on the amount of computation that may be performed in a single transaction.

To provide complex market functionality, the computation must be performed *off-chain* and only the results should be evaluated and verified by the smart contract on-chain. This is apparent in the implementation of transactive energy systems where the trades must be decided optimally based on a complex set of equations considering the feeder design and various power limits. Such complex computations are not possible to implement in smart contract languages like Solidity. Therefore, we have developed a novel hybrid solver pattern for TRANSAX where we integrate external solvers with smart contracts. This enables us to perform the computations off-chain and verify them on the blockchain.

Privacy Concerns Although it is possible to make anonymous transactions with cryptocurrencies, energy trades may need information that reveals the traders' identities. For example, the trades must be associated with a specific feeder to ensure that the maximum power transferred through the feeder is less than the rated capacity. This poses a challenge for privacy, because a trader may need to reveal its location to permit constraint checks and validate trades.

If the information is available publicly, then the inference of energy usage patterns can be exploited, for example, to infer the presence or absence of a person in their home. Brenzikofer et al. [60] address privacy while incentivizing stability through dynamic grid tariffs. However, their safety checks are limited to the total aggregated grid load rather than per feeder constraints, which are essential in a power network. In TRANSAX, we use the concept of tradeable and mixable energy assets in a transactive energy system to provide a

level of anonymity to the users while ensuring that system calculations at the feeder level are still safe.

Integration Concerns Integrating legacy infrastructure with blockchains is challenging since most existing smart meters lack the computational capabilities required to participate in a blockchain network [61]. An alternative to directly participating is for the devices to send their data to nodes that are connected to the blockchain network. However, this requires configuring each device to connect to a suitable gateway and mechanisms to handle lost connections and gateway failures. Moreover, while the ledger provides consensus on when to produce or consume power, participants still need to time synchronize their energy transfer to avoid instabilities in the system.

TRANSAX solves the integration concerns using RIAPS (Resilient Information Architecture Platform for Smart Grids) [57], a platform for building distributed, fault-tolerant smart-grid applications. RIAPS provides key services, like time-synchronization and discovery. Discovery facilitates the integration of legacy hardware with blockchain applications by automating the network connections between them via RIAPS nodes, which have been developed to run on low-cost embedded devices. Each component in the TRANSAX is either a RIAPS node or interfaces with a RIAPS node.

Security Threats Research on power systems security has investigated cyber-attacks with different goals and strategies. Some attacks exploit the centralized nature of the system, for example, by compromising the utility's network to access control systems (such as in the attacks against Ukraine's power utilities). Other scenarios consider adversaries that target IoT or smart appliances to create disturbances in the system (e.g., turning all the A/Cs on at the same time).

The distributed nature of blockchain prevents some attacks that are feasible in centralized systems. For example, some *false data injection* attacks that modify utility's messages (e.g., price signals) may fail because the devices can verify such information with multiple sources (blockchain nodes). Hence, an adversary may have to compromise multiple

blockchain nodes to deceive smart appliances. However, some attacks remain. Since prosumers must connect to the blockchain-based system through gateway nodes, an adversary can still attempt to "cut off" prosumers from the system by targeting these gateway nodes and making them unavailable. For example, an adversary can launch a (distributed) denial of service attack against a gateway node to prevent a set of bids from arriving at the market on time. Using this attack, the adversary, who may be affiliated with one of the market participants, can increase (or decrease) market prices by delaying a set of lower (or higher) price bids. We are still in the preliminary stages of developing active mitigation strategies in TRANSAX to prevent these attacks.

Building a decentralized market for transactive microgrids is challenging because the system must satisfy several requirements, which are often in conflict with each other.

- First, the market must be *efficient*, *i.e.*, the system should maximize the utilization of local supply—in meeting local demand—by matching the prosumers in the microgrid, taking advantage of their temporal flexibility in production and consumption. This requirement is crucial since effective trading is the purpose of the system; all other requirements are supporting this one.
- Second, the market must be *safe*, *i.e.*, the system must reject trades that would endanger the stability or physical safety of the microgrid [2].
- Third, the market must be *privacy-preserving*, *i.e.*, the system should conceal information that could be used to infer the prosumers' energy usage patterns. The amount of energy produced, consumed, bought, or sold by any prosumer should be anonymous to other prosumers and limited to the monthly bill for the DSO. This is necessary because such information could be exploited, *e.g.*, to determine when a resident is at home.
- Fourth, the market must be *secure*, *i.e.*, the system must ensure authenticity, data integrity, and auditability for offers, trades, and bills.

[2]Note that this is orthogonal to the *physical enforcement of safety* which is provided by overcurrent protection units that limit the total current flowing through the microgrid.

- Finally, the market must be *resilient*, *i.e.*, it must retain availability even if some nodes or entities (*e.g.*, DSO) are unavailable.

Addressing the requirements of privacy, safety, and efficiency simultaneously in a decentralized system is essential because removing any of these requirements significantly simplifies the problem. For example, if we do not consider safety, then privacy is easy since all offers can be made anonymously. If we do not consider privacy, then safety is easy since the safety constraints are associated with the offers and can be checked. If we do not consider efficiency, then we can simply say no trades are allowed, preserving privacy and safety. If we allow a centralized market, then the centralized market can keep the offers confidential and can check the safety constraints. However, such a system has a single point of failure. Assurance that the system is secure and resilient is crucial in practice, *e.g.*, because communities are facing increasing cyber threats as well as natural disasters that disrupt infrastructure.

The research community is increasingly advocating the use of distributed ledgers in the energy sector [55]. This is primarily because a distributed ledger can provide an immutable, complete, and fully auditable record of all transactions that have occurred within a system. However, there are still research gaps. For example, Andoni et al. [55] surveyed 140 applications of distributed ledgers and found that 33% were focused on decentralized energy trading, with privacy preservation being a key challenge that has not been addressed. They also highlighted the balancing of supply to demand (stability) as another critical issue. The work presented in this paper addresses the problem of privacy while ensuring that safety constraints can be enforced for trades.

Contributions: In this paper, we introduce TRANSAX[3], a blockchain-based decentralized transactive energy system that provides privacy while preserving safety without using a centralized market. The underlying communication substrate is implemented by using a distributed middleware, called Resilient Information Architecture Platform for Smart Grid

[3]This paper is a significant extension of our previously published conference papers [62, 34, 37].

(RIAPS) [63, 64, 65], which provides resilience against failure of individual services. The specific contributions of this paper are as follows.

1. To enable safe and anonymous trading, we introduce production and consumption assets that allow us to dissociate safety from privacy (Section 2.6.3). To provide privacy, we integrate a decentralized mixing protocol [22] that enables prosumers to anonymize their production and consumption assets within their groups, and hence trade anonymously. This also enables privacy-preserving billing such that no information is disclosed to the DSO other than the billed amount.
2. We show in Section 2.8.2 that prosumers can be anonymous within groups (Section 2.6.2) while preserving system safety; however, there is potential for some loss of trading efficiency. We provide an analysis of efficiency loss and under what conditions.
3. To improve trading efficiency, we provide prosumers with the ability to specify production and consumption offers with temporal flexibility (Section 2.6.4). We solve the trading problem as a linear program, maximizing the energy traded over a long time horizon[4], and introduce a hybrid architecture for solving this linear program.
4. We show in Section 2.8.1.2 that the hybrid architecture can combine the resilience of distributed ledgers with the computational efficiency of conventional compute platforms when solving the energy allocation problem. This hybrid architecture ensures the integrity of data and computational results— if the majority of the ledger nodes are secure—while allowing the complex computation to be performed by a set of redundant and efficient solvers.
5. In Section 2.9, we provide a testbed and experimental analysis of our proof-of-concept implementation of TRANSAX. We show that our approach is feasible for private blockchains in the grid-connected microgrid setting.

Outline: We introduce the background concepts in Section 2.4. TRANSAX compo-

[4]In Section 2.8.1.3, we show that temporal flexibility can improve trading efficiency.

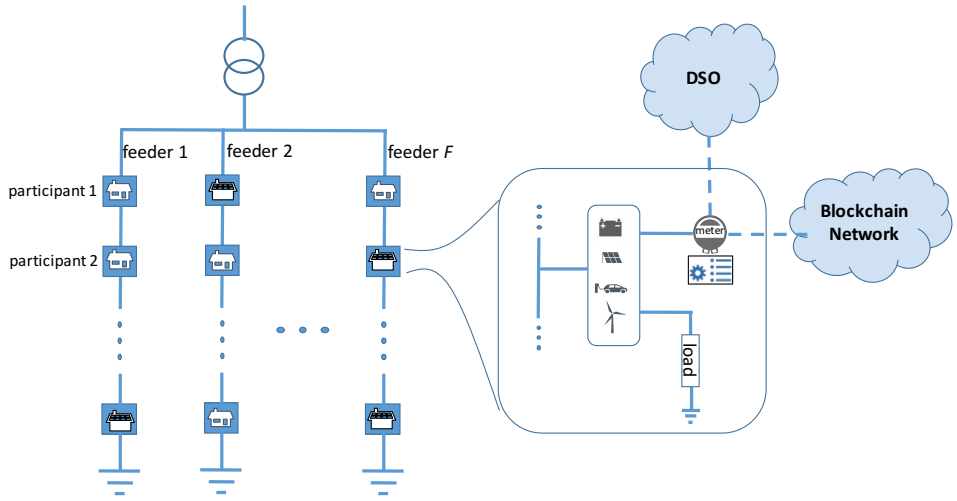


Figure 2.1: Illustration of a multi-feeder system. Each feeder is protected by an overcurrent relay at the junction of the common bus. The inset figure shows that a node in the network has different kinds of loads, some of which can be scheduled, making it possible for a consumer to bid in advance for those loads. The smart meter ensures proper billing per node. The blockchain network is an immutable record of all transactions and is used for scheduling energy transfers between homes in the microgrid and between homes and the DSO.

nents are discussed in Section 2.5. We present the energy trading approach, including extensions to support safety and privacy in Section 2.6. Then, we describe the protocol that implements the energy trading approach in Section 2.7. We analyze how we meet key requirements and discuss the tradeoff between privacy and efficiency in Section 2.8. We present an integrated testbed using GridLAB-D [66] and numerical results in Section 2.9. Finally, we present related research in Section 2.10 followed by conclusions in Section 2.11.

2.4 Background

To explain the concepts of TRANSAX, we first need to provide an overview of basic concepts and assumptions and how we use them in TRANSAX.

Assumption on the Microgrid Architecture We consider a microgrid with a set of feeders.

A feeder has a fixed set of nodes, each representing a residential load or a combination of load and distributed energy resources, such as rooftop solar and batteries, as shown in Fig. 2.1. Each node is associated with a participant in the local peer-to-peer energy trading market, and each participant is independent and has control over its energy utilization. The participants may be able to predict their future production and consumption based on historical data and anticipated utilization.

Resilient Information Architecture for Smart Grid RIAPS is an *open application platform* for smart grids that distributes intelligence and control capability to local endpoints to reduce total network traffic, avoid latency, and decrease dependency on multiple devices and communication interfaces, thereby enhancing reliability. RIAPS also provides platform services to power-system applications running on remote nodes [67], including 1. resource-management framework to control the use of computational resources, 2. fault--management framework to detect and mitigate faults in all layers of the system, 3. security framework to protect the confidentiality, integrity, and availability of a system under cyber-attacks, 4. fault-tolerant time-synchronization service, 5. discovery framework to establish the network of interacting actors for an application, and 6. deployment and management framework for administration of the distributed applications from a control room. In TRANSAX, RIAPS is used as the base middleware and application-management substrate, allowing all actors to communicate and the protocol—discussed later in this paper—to be implemented. As a note, actor interactions are supplemented with interactions with the distributed ledger and the smart contract. We will discuss these interactions in detail in the protocol section. For more details on RIAPS, we refer the interested reader to [38].

Distributed Ledgers

Distributed ledgers refer to distributed databases where nodes in a network simultaneously reconcile their copies of the data and sequence of actions through Byzantine consensus to achieve a shared truth so that data in the shared ledger can be verified and is tamper-aware. Data is added to the ledger via transactions and all transactions submitted

to the ledger are associated with an account, which is typically not anonymous. The immutability of actions is crucial for providing safety and security, *i.e.*, after a transaction has been recorded, it cannot be modified or removed from the ledger. The ledger is distributed to enhance fault tolerance. Since a distributed ledger is maintained by multiple nodes, nodes must reach a consensus on which transactions are valid and stored on the ledger. This consensus must be reached both quickly and reliably, even in the presence of erroneous or malicious (*e.g.*, compromised) ledger nodes. We make no assumptions about the particulars of the consensus algorithm. In practice, a distributed ledger can be implemented using, *e.g.*, *blockchains* with proof-of-stake consensus or a practical Byzantine fault tolerance algorithm [68]. In TRANSAX, we use Ethereum [69] as the ledger.

Mixing The use of blockchains in building a transactive energy platform is appealing also because they elegantly integrate the ability to immutably record the ownership and transfer of assets, with essential distributed computing services such as Byzantine fault-tolerant consensus on the ledger state as well as event chronology. The ability to establish consensus on state and timing is important in the context of TES since these systems are envisioned to involve the participation of self-interested parties, interacting with one another via a distributed computing platform that executes transaction management. However, this also leads to the problem of privacy as the records in the blockchain can be attributed to the prosumers. The earliest approach to solve the privacy problem in blockchains was mixing. The key concept in mixing is to hide the linkage between the inputs and outputs of a transaction by combining them with other transactions. In TRANSAX, we use CoinShuffle [22]. A simplified example of how this works is that each participating prosumer provides an anonymous output account and shuffles them with the others so that only the owner knows who owns a specific account. Then, if each prosumer inputs the same amount, all the transaction must do is transfer that amount from each of the public accounts to each of the anonymous accounts. Since the anonymous accounts were shuffled, no anonymous account can be linked conclusively to its owner. Note that this does not hide the quantity

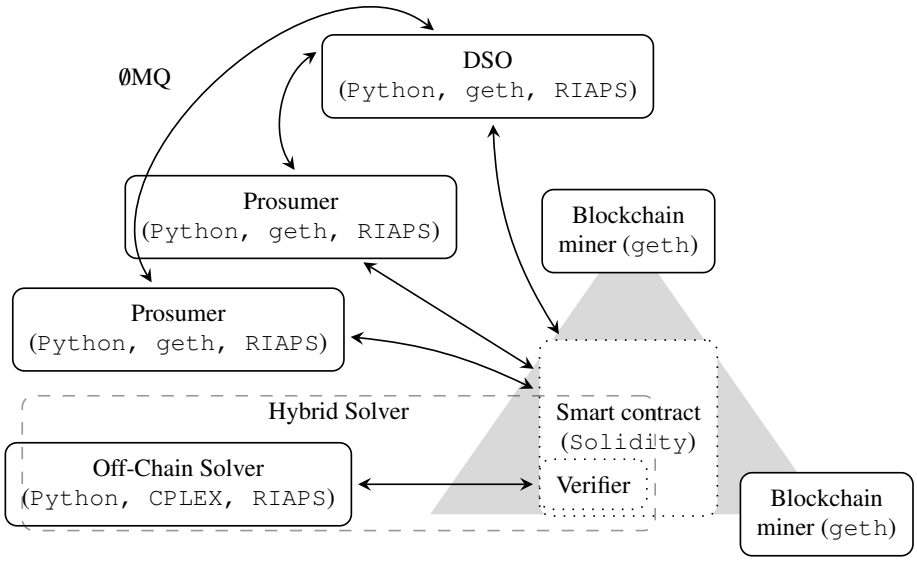


Figure 2.2: Components of the energy trading system. In our reference implementation, we use Ethereum as the decentralized computation platform for smart contracts, and the other components interact with the blockchain network using the `geth` Ethereum client. The smart contract is implemented in Solidity, a high-level language for Ethereum, and it is executed by a private network of `geth` mining nodes. The off-chain solver uses CPLEX.

of assets stored in each anonymous account.

2.5 TRANSAX Components

In this section, we introduce the components of TRANSAX (see Fig. 2.2).

Distribution System Operator We assume the existence of a distribution system operator (DSO) that participates in the market and may use the market to incentivize timed energy production within the microgrid to aid in grid stabilization and promotion of related ancillary services [70] through updates to the price policy. The participants settle trades in advance using automated matching which allows them to schedule their transfer of energy into the local distribution system. The DSO meets the participants' residual demand and supply, *i.e.*, consumption and production that they did not trade in advance due to estimation error or lack of trade partners[5].

[5]Note that this requires the presence of a secondary controller that balances voltage and frequency in the

DSO is also responsible for handling financial operations, such as sending monthly bills, and functional operations, such as registering new smart meters[6]. Registration means adding a new smart meter to the platform when it is installed for a prosumer. Thus, DSO provides *safety* and *security* by limiting access to the distributed ledger to prosumers that have registered.

Producers and Consumers The participants in the market are the producers and consumers of energy, collectively referred to as prosumers. The prosumers are built upon the RIAPS middleware which provides communication services via the ZeroMQ messaging library [7]. RIAPS additionally provides time synchronization between the prosumers allowing them to synchronize production and consumption, ensuring that they remain balanced. The prosumers can construct offers and trade with other prosumers. The trades are submitted to a *Smart Contract* (described below) via a geth client. Geth [8] is the Go implementation of the Ethereum protocol and is used to instantiate and interact with an Ethereum blockchain. The prosumers use a light client that interacts with the full clients that constitute the blockchain. Each prosumer has a smart meter that measures the prosumer's energy production and consumption. The smart meter aggregates this data and provides it to the DSO periodically, which is a necessary element of privacy-preserving billing. Prosumers work together to achieve privacy via the execution of mixing protocol.

Distributed Ledger The blockchain in Fig. 2.2 provides the basis for the market functionality of TRANSAX. It provides immutable storage service for offers, solutions, safety constraints, and a notification log of events. Prosumers and solvers check for these events and perform actions based on them. Nodes in the network host full Ethereum clients to provide this substrate. *Hybrid Solver* The hybrid solver in Fig. 2.2 is an innovation of

microgrid as described in our prior work in [71].

[6]In practice, these smart meters must be tamper-resistant to prevent prosumers from "stealing electricity" by tampering with their meters. After a smart meter has measured the net amount of energy consumed by the prosumer in some time interval, it can send this information to the DSO for billing purposes. This way, the DSO has no fine-grained information on the energy profile of the consumer, the DSO only knows the amount that needs to be paid for the energy consumed during that cycle.

[7]https://zeromq.org/

[8]https://geth.ethereum.org/downloads/

TRANSAX, which enables a distributed market architecture that has the auditability and resilience of blockchains through a smart contract, and can yet use high-performance computers to solve the computationally expensive optimization problem *off-blockchain*. The hybrid solver consists of smart contract elements as well as an off-chain solver.

- *Smart contract* — The smart contract in Fig. 2.2 provides the essential functionality of the market, such as financial transactions and enabling offers to be submitted and then matched into trades that satisfy the safety constraints of the system. The matching of the offers is a complex optimization problem, and since smart contracts are limited in the number of computations they can perform, we do not use the smart contract to match offers into trades. Rather, we only use it for validation of energy trading solutions provided by the *off-chain solver*.
- *Off-Chain Solver* — The off-chain solver in Fig. 2.2 consists of a set of solvers. Any participant of the system can act in the solver role since all offers posted on the blockchain are public. Prosumers are incentivized to act as solvers — especially if no dedicated solvers are available — since they can create trades that benefit them. Note that this is safe because the smart contract verifies each solution and accepts a new solution only if it is feasible and strictly better than the current solution. Each solver can use whatever strategy it chooses for solving because the solutions will still be verified. In this work, we implement an efficient linear programming solver using CPLEX [72], which can be run off-blockchain, on any capable computer (or multiple computers for increased reliability). The solver is run periodically to find a solution to the energy trading problem based on the latest set of offers posted. Once a solution is found by the matching solver, it is submitted to the smart contract in a blockchain transaction, which is validated by the smart contract. Note that if new offers have been posted since the solver started working on its solution, the solution computed by the solver will still be considered valid by the smart contract because any solution that is valid for a set of offers is also valid for a superset of those offers. Since solvers

may fail, the smart contract should accept solutions from multiple off-blockchain solvers to preserve the reliability provided by the blockchain. However, these solvers might provide different solutions. Thus, the smart contract must be able to choose from multiple solutions (some of which may come from compromised nodes).

2.6 Energy Trading Approach

The distribution network infrastructure is a collection of feeders (Fig. 2.1). A feeder has a fixed set of nodes, each representing a prosumer, which is a combination of load and distributed energy resources, such as rooftop solar panels and batteries. We assume that the prosumers can estimate their future production and consumption based on historical data and anticipated utilization. The prosumers submit energy offers based on their estimates via automated agents that act on behalf of residents (*i.e.*, residents do not need to trade manually). The estimates do not need to be perfect because we assume the existence of a distribution system operator (DSO), which also participates in the market and can supply residual demand not met through the local market. The DSO may use the market to incentivize timed energy production within the microgrid to aid in grid stabilization and the promotion of related ancillary services [70] through updates to the price policy. Since the trades record only the energy futures and do not control the actual exchange of energy, we include a smart meter at each prosumer to measure the prosumer's actual energy production and consumption. In practice, these smart meters must be tamper-resistant to prevent prosumers from "stealing electricity." After a smart meter has measured the net amount of energy consumed by the prosumer in some time interval, it can send the relevant information to the DSO for billing purposes to keeping the actual consumption private.

Our goal is to find an optimal match between energy production and consumption offers, which we refer to as the *energy trading problem*. Each offer is associated with an identity that belongs to the prosumer that posted the offer. We refer to these identities as *accounts*, and prosumers may generate any number of them.

Table 2.2: List of Symbols

Symbol	Description
	Microgrid
$\mathscr{F}$, $\mathscr{U}$	set of feeders and prosumers, resp.
C_f^e, C_f^i	maximum net (*external)* and total (*internal*) load *constraints*, resp., on feeder $f \in \mathscr{F}$
L_u^+, L_u^-	*production* (+) and *consumption* (-) *limits*, resp., of prosumer u
C_g^e, C_g^i	maximum net (*external)* and total (*internal*) load *constraints*, resp., on group $g \in \mathscr{G}$
EPA, ECA	asset granting permission to produce or consume, resp., a unit of energy
Δ	length of each time interval
T_{clear}	minimum number of time intervals between the finalization and notification of a trade
E_u^t	energy transferred by prosumer u in interval t
t_f	next interval to be finalized $t+1+T_{clear}$
	Offers
$\mathscr{S}_f$, $\mathscr{B}_f$	set of selling and buying offers, resp., from feeder $f \in \mathscr{F}$
$\mathscr{S}$, $\mathscr{B}$	set of all selling and buying offers, resp.
$\mathscr{S}^{(t)}$, $\mathscr{B}^{(t)}$	set of all selling and buying offers, resp., submitted by the end of time interval t
A_s, A_b	account that posted offers $s \in \mathscr{S}$ and $b \in \mathscr{B}$, resp.
E_s, E_b	amount of energy to be sold or bought, resp., by offers $s \in \mathscr{S}$ and $b \in \mathscr{B}$
I_s, I_b	time intervals in which energy could be provided or consumed by offers $s \in \mathscr{S}$ and $b \in \mathscr{B}$, resp.
R_s, R_b	reservation prices of offers $s \in \mathscr{S}$ and $b \in \mathscr{B}$, resp.
$\mathscr{M}(s)$, $\mathscr{M}(b)$	set of offers that are matchable with offers $s \in \mathscr{S}$ and $b \in \mathscr{B}$, resp.
	Solution
$\varepsilon_{s,b,t}$	amount of energy that should be provided by s to b in interval t
$\pi_{s,b,t}$	unit price for the energy provided by s to b in interval t
$Feasible(\mathscr{S},\mathscr{B})$	set of feasible solutions given sets of selling and buying offers $\mathscr{S}$ and $\mathscr{B}$
$\hat{\varepsilon}_{s,b,t}$, $\hat{\pi}_{s,b,t}$	finalized trade values
	Implementation Parameters
T_h	solve horizon; the number of intervals beyond the most recently finalized interval that are considered by the solver (offers beyond horizon $t_f + T_h$ are not considered by solver)
$\hat{\Delta}$	length of the time step used for simulating the real interval of length Δ

2.6.1 The Basic Problem Specification

Let $\mathscr{F}$ denote the set of feeders. On each feeder, there is a set of prosumers, who can make offers to buy and sell energy. We assume that time is divided into intervals of fixed length Δ, and we refer to the t-th interval simply as time interval t. For a list of symbols used in the paper, see Table 2.2.

For feeder $f \in \mathscr{F}$, we let $\mathscr{S}_f$ and $\mathscr{B}_f$ denote the set of selling and buying offers posted by prosumers in feeder f, respectively.[9] A selling offer $s \in \mathscr{S}_f$ is a tuple (A_s, E_s, I_s, R_s), where A_s is the account that posted the offer, E_s is the amount of energy to be sold, I_s is the set of time intervals in which the energy could be provided, R_s is the reservation price, *i.e.*, lowest unit price for which the prosumer is willing to sell energy. Similarly, a buying offer $b \in \mathscr{B}_f$ is a tuple (A_b, E_b, I_b, R_b), where the values pertain to consuming/buying energy instead of producing/selling, and R_b is the highest price that the prosumer is willing to pay. For convenience, we also let $\mathscr{S}$ and $\mathscr{B}$ denote the set of all buying and selling offers (*i.e.*, we let $\mathscr{S} = \cup_{f \in \mathscr{F}} \mathscr{S}_f$ and $\mathscr{B} = \cup_{f \in \mathscr{F}} \mathscr{B}_f$).

We say that a pair of selling and buying offers $s \in \mathscr{S}$ and $b \in \mathscr{B}$ is *matchable* if

$$R_s \leq R_b \quad \text{and} \quad I_s \cap I_b \neq \emptyset. \tag{2.1}$$

In other words, a pair of offers is matchable if there exists a price that both prosumers would accept and a time interval in which the seller and buyer could provide and consume energy. For a given selling offer $s \in \mathscr{S}$, we let the set of buying offers that are matchable with s be denoted by $\mathscr{M}(s)$. Similarly, we let the set of selling offers that are matchable with a buying offer b be denoted by $\mathscr{M}(b)$.

A solution to the energy trading problem is a pair of vectors (ε, π), where $\varepsilon_{s,b,t}$ is a non-negative amount of energy that should be provided by offer $s \in \mathscr{S}$ and consumed by offer $b \in \mathscr{M}(s)$ in time interval $t \in I_s \cap I_b$ [10]; and $\pi_{s,b,t}$ is the unit price for the energy provided

[9] To include the DSO in the formulation, we assign it to a "dummy" feeder.

[10] We require the both seller and buyer to produce a constant level of power during the time interval. This

by offer $s \in \mathscr{S}$ to offer $b \in \mathscr{M}(s)$ in time interval $t \in I_s \cap I_b$.

A pair of vectors (ε, π) is a feasible solution to the energy trading problem if it satisfies the following two constraints. First, the amount of energy sold or bought from each offer is at most the amount of energy offered:

$$\forall s \in \mathscr{S} : \sum_{b \in \mathscr{M}(s)} \sum_{t \in I(s,b)} \varepsilon_{s,b,t} \leq E_s \quad \text{and} \quad \forall b \in \mathscr{B} : \sum_{s \in \mathscr{M}(b)} \sum_{t \in I(s,b)} \varepsilon_{s,b,t} \leq E_b \tag{2.2}$$

Second, the unit prices are between the reservation prices of the seller and buyer:

$$\forall s \in \mathscr{S}, b \in \mathscr{M}(s), t \in I(s,b) : \; R_s \leq \pi_{s,b,t} \leq R_b \tag{2.3}$$

The objective of the energy trading problem is to maximize the amount of energy traded. The rationale behind this objective is maximizing the load reduction on the bulk power grid. Formally, an optimal solution to the energy trading problem is

$$\max_{(\varepsilon,\pi) \in \mathit{Feasible}(\mathscr{S},\mathscr{B})} \sum_{s \in \mathscr{S}} \sum_{b \in \mathscr{M}(s)} \sum_{t \in I(s,b)} \varepsilon_{s,b,t} \tag{2.4}$$

where $\mathit{Feasible}(\mathscr{S}, \mathscr{B})$ is the set of feasible solutions given selling and buying offers $\mathscr{S}$ and $\mathscr{B}$ (*i.e.*, set of solutions satisfying Equations (2.2) and (2.3) with $\mathscr{S}$ and $\mathscr{B}$).

The above formulation ensures feasibility, which takes the reservation prices into account. However, we do not address how to set the clearing prices in this paper. Clearing prices could be set using an existing approach, *e.g.*, double auction; however, this is part of our future work (Section 2.11).

2.6.2 Adding Safety Extensions to Problem Specification

To ensure the safety of the microgrid, we introduce additional constraints on the solution to the energy trading problem. Each prosumer u has independent production and

can be achieved by smart inverters.

consumption limits, which are denoted by L_u^+ and L_u^-, respectively. Further, each feeder $f \in \mathscr{F}$, has a transformer for incoming energy, which has a capacity rating. We let C_f^e denote the capacity of the transformer of feeder f. Similarly, the distribution lines and transformers within the feeder have capacity ratings as well. We let C_f^i denote the maximum amount of energy that is allowed to be consumed or produced within the feeder during an interval[11]. These constraints are physically enforced by the over-current relays of the circuit breakers and feeders.

Now we generalize and introduce the notion of groups. We note that groups can correspond to feeders and support the constraints that we introduced in the previous paragraphs. They allow us to support physical layouts other than strictly feeders, and it will be useful for privacy later. We define a *group* g to be a set of feeders (*i.e.*, $g \subseteq \mathscr{F}$). We let $\mathscr{G}$ be the set of all groups, and for each group $g \in \mathscr{G}$, we introduce group safety limits C_g^i and C_g^e, which are analogous to feeder limits. A solution is safe if it satisfies the following three constraints. First, the amount of energy transferred out of or into a prosumer is within the production and consumption limits in all time intervals:

$$\forall u \in \mathscr{U}, t : \sum_{s \in \mathscr{S}_u} \sum_{b \in \mathscr{B}} \varepsilon_{s,b,t} \leq L_u^+ \quad \text{and} \quad \forall u \in \mathscr{U}, t : \sum_{b \in \mathscr{B}_u} \sum_{s \in \mathscr{S}} \varepsilon_{s,b,t} \leq L_u^- \tag{2.5}$$

where $\mathscr{S}_u$ and $\mathscr{B}_u$ are the sets buying and selling offers posted by accounts owned by prosumer u.

Second, the amount of energy consumed and produced within each group is below the safety limit in all time intervals:

$$\forall g \in \mathscr{G}, t : \underbrace{\max \left\{ \sum_{b \in \mathscr{B}_g} \sum_{s \in \mathscr{S}} \varepsilon_{s,b,t}, \sum_{s \in \mathscr{S}_g} \sum_{b \in \mathscr{B}} \varepsilon_{s,b,t} \right\}}_{\textit{max of energy bought or sold } (X)} \leq C_g^i \tag{2.6}$$

[11] In other words, limit C_f^e is imposed on the net production and net consumption of all prosumers in feeder f, while limit C_f^i is imposed on the total production and consumption of prosumers in feeder f.

This means that the sum of all the buying trades nor the sum of selling trades can exceed the safety limit.

Third, the amount of energy flowing into or out of each group is within the safety limit in all time intervals:

$$\forall g \in \mathscr{G}, t : -C_g^e \leq \underbrace{\left(\sum_{s \in \mathscr{S}_g} \sum_{b \in \mathscr{B}} \varepsilon_{s,b,t}\right) - \left(\sum_{b \in \mathscr{B}_g} \sum_{s \in \mathscr{S}} \varepsilon_{s,b,t}\right)}_{\textit{net} \text{ energy transfer } (N)} \leq C_g^e \tag{2.7}$$

Note that the maximum of bought or sold energy (X) in Eq. (2.6) is always greater than the net energy transferred (N) in Eq. (2.7), *i.e.*, $N < X$. This is important because it means that we need to consider only $C_g^e \leq C_g^i$. If we considered $C_g^i < C_g^e$, then $N < X \leq C_g^i < C_g^e$, which means that the internal limit will always trip ($X > C_g^i$) before the external limit, making the external limit irrelevant. This observation will be important in Section 2.8.2

2.6.3 Adding Privacy Extensions to Problem with Safety Specifications

To protect prosumers' privacy, we let them use anonymous accounts when posting offers. By generating new anonymous accounts, a prosumer can prevent others from linking the anonymous accounts to its actual identity, thereby keeping its trading activities private. However, anonymous accounts pose a threat to safety. Since the energy trading formalization with safety extension (see Equations (2.5) - (2.7)) discussed earlier requires the offers to be associated with the prosumer to enforce prosumer-level constraints and with the group from which they originated to be able to enforce group-level safety constraints. Without these associations, prosumers can generate any number of anonymous accounts. They can then post selling and buying offers for large amounts of energy without any intention of delivering and without facing any repercussions. A malicious or faulty prosumer could easily destabilize the grid with this form of reckless trading. Consequently, the amount of energy that may be traded by anonymous accounts belonging to a prosumer must be limited.

To enforce the prosumer-level constraints we introduce the concept of energy production and consumption assets, which allows us to disassociate the limiting of assets from the anonymity of offers. First, an *energy production asset* (EPA) is a tuple $(E_{EPA}, I_{EPA}, G_{EPA})$, where

- E_{EPA} is the permission to sell a specific non-negative amount of energy to be produced,
- I_{EPA} is the set of intervals for which the asset is valid, and
- G_{EPA} is the group that the asset is associated with.

Second, an *energy consumption asset* (ECA) represents permission to buy a specific amount of energy and is defined by the same fields. For this asset, however, the fields define energy consumption instead of production. Each prosumer u is only permitted to withdraw assets up to the limits L_u^+ and L_u^- into a non-anonymous account.

These assets can be moved to anonymous accounts in an untraceable way such as through an *anonymizing mixer*. The mixer ensures that accounts cannot be linked to the prosumer that owns them. However, the anonymous accounts must retain their group association and the sum of the assets remains constant. Production assets are required to post a selling offer, and consumption assets are required to post a buying offer. For the offer to be valid, the account posting the offer must have assets that cover the amount and intervals offered. When a trade is finalized the assets are exchanged. We will provide more details on how they fit into the trading approach in Section 2.7.

To enforce group-level safety we only provide group-level anonymity, meaning that an offer can be traced back to its group of origin, but not to the individual prosumer within the group. When forming a group, the safety constraints need to be set appropriately. We will discuss how they should be set and the associated energy trading capacity costs in Section 2.8.2.

2.6.4 Introducing the Notion of Clearance Windows

In our basic problem formulation, we assumed that all buying and selling offers $\mathscr{B}$ and $\mathscr{S}$ are available at once, and we cleared the market in one take. In practice, however, the market conditions and the physical state of the DSO and prosumers may change over time, making it advantageous to submit new offers[12]. As new offers are posted we need to recompute the solution. While new offers can increase the amount of energy traded, the *trade values* $\varepsilon_{s,b,t}$ and $\pi_{s,b,t}$ need to be *finalized* at some point in time. At the very latest, values for interval t need to be finalized by the end of interval $t-1$; otherwise, participants would have no chance of actually delivering the trade.

Here, we extend the energy trading problem to accommodate a time-varying offer set (where offers can be unmatched, matched and pending, or matched and finalized), and a time constraint for finalizing trades. Our approach finalizes only trades that need to be finalized, which maximizes efficiency while providing safety. We assume that all trades for time interval t' (*i.e.*, all values $p_{s,b,t'}$ and $\pi_{s,b,t'}$) must be finalized and the trading prosumers must be notified by the end of time interval $t' - T_{clear} - 1$ (see Fig. 2.3), where T_{clear} is a positive integer constant that is set by the DSO. In other words, if the current interval is t, then all intervals up to $t + T_{clear}$ have already been finalized. Preventing "last-minute" changes can be crucial for safety and fairness since it allows both the DSO and the prosumers to prepare for delivering (or consuming) the right amount of energy. In practice, the value of T_{clear} must be chosen accounting for both physical constraints (*e.g.*, how long it takes to turn on a generator) and communication delay (*e.g.*, some participants might learn of a trade with delay due to network disruptions).

We let $\hat{\varepsilon}_{s,b,t}$ and $\hat{\pi}_{s,b,t}$ denote the finalized trade values, and we let $\mathscr{B}^{(t)}$ and $\mathscr{S}^{(t)}$ denote the set of buying and selling offers that participants have submitted by the end of time interval t. Then, the system takes the following steps at the end of each time interval t.

[12] Updating or cancelling offers could also be useful; however, we do not provide this functionality in the current version and leave it for future work.

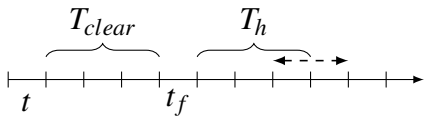


Figure 2.3: Temporal parameters (t is the current interval, t_f is the interval to be finalized).

First, find an optimal solution (ε^*, π^*) to the extended energy trading problem:

$$\max_{(\varepsilon,\pi) \in \mathit{Feasible}(\mathscr{S}^{(t)}, \mathscr{B}^{(t)})} \sum_{s \in \mathscr{S}^{(t)}} \sum_{b \in \mathscr{M}(s)} \sum_{\tau \in I(s,b)} \varepsilon_{s,b,\tau} \tag{2.8}$$

subject to

$$\forall\, \tau \leq t_f : \ \varepsilon_{s,b,\tau} = \hat{\varepsilon}_{s,b,\tau} \tag{2.9}$$

$$\pi_{s,b,\tau} = \hat{\pi}_{s,b,\tau} \tag{2.10}$$

Second, finalize trade values for time interval t_f based on the optimal solution (ε^*, π^*):

$$\hat{\varepsilon}_{s,b,t_f} := {\varepsilon^*}_{s,b,t_f} \tag{2.11}$$

$$\hat{\pi}_{s,b,t_f} := \pi^*_{s,b,t_f} \tag{2.12}$$

By taking the above steps at the end of each time interval, trades are always cleared based on as much information as possible (*i.e.*, considering as many offers as possible)[13] without violating any safety or timing constraints. Note that here $\mathit{Feasible}(\mathscr{S}, \mathscr{B})$ now also includes the safety constraints (2.5), (2.6), and (2.7).

[13]This includes offers for intervals beyond the finalization interval. Effectively, matches for an interval beyond finalization can be changed if a better solution is found; however, finalized matches are permanent and never changed.

2.6.5 Practical Considerations for Solving the Problem

To find the optimal solution efficiently, we frame the energy trading problem as a linear program. First, we create real-valued variables $\varepsilon_{s,b,t}$ and $\pi_{s,b,t}$ for each $s \in S, b \in \mathscr{M}(s), t \in I_s \cap I_b$. Then, the following reformulation of the matching problem is a linear program:

$$\max_{\varepsilon,\pi} \sum_{s \in \mathscr{S}} \sum_{b \in \mathscr{M}(s)} \sum_{t \in I(s,b)} \varepsilon_{s,b,t} \tag{2.13}$$

subject to the constraint Equations, which can all be expressed as linear inequalities (2.2), (2.3), (2.5), (2.6), (2.7), and

$$\varepsilon \geq \mathbf{0} \text{ and } \pi \geq \mathbf{0}. \tag{2.14}$$

However, we must consider that even though Equation (2.4) can be formulated as a linear program and be solved efficiently (*i.e.*, in polynomial time), the number of variables $\{\varepsilon_{s,b,t}\}$ may grow prohibitively high as the number of offers and time intervals that they span increases. In practice, this may pose a significant challenge for solving the energy trading problem for larger transactive microgrids. A key observation that helps us tackle this challenge is that even though prosumers may post offers whose latest intervals are far in the future, the optimal solution for the finalized interval typically depends on only a few intervals ahead of the finalization deadline. Indeed, we have observed that considering intervals in the far future has little effect on the optimal solution for the interval that is to be finalized next (see Fig. 2.8).

Consequently, for practical solvers, we introduce a planning horizon T_h (see Fig. 2.3) that limits the intervals that need to be considered for a solution: for any $\hat{t} > t_f + T_h$, we set $\varepsilon_{s,b,\hat{t}} = 0$, where t_f is the earliest interval that has not been finalized. By "pruning" the set of free variables, we can significantly improve the performance of the solver with a negligible effect on solution quality (see Fig. 2.8). This results in the following "pruned" objective

function:

$$\max_{(\varepsilon,\pi)\in \mathit{Feasible}(\mathscr{S},\mathscr{B})} \sum_{s\in\mathscr{S}} \sum_{b\in\mathscr{M}(s)} \sum_{\tau\in I_s\cap I_b\cap\{\tau;\tau\le t_f+T_h\}} \varepsilon_{s,b,\tau} \tag{2.15}$$

Although solving linear programs is not computationally hard, it can be challenging with many variables and constraints in resource-constrained computing environments. Since computation is relatively expensive on blockchain-based distributed platforms[14], solving even the "pruned" energy trading problem from Equation (2.15) might be infeasible using a blockchain-based smart contract. Considering this, we choose to use our hybrid-solver approach since compared to finding optimal trades, verifying the feasibility of a solution (ε,π) and computing the value of the objective function is computationally inexpensive and can easily be performed on a blockchain-based decentralized platform. Thus, the smart contract provides the following functionality:

- Solutions may be submitted to the smart contract at any time. The contract verifies the feasibility of each submitted solution, and if the solution is feasible, then the contract computes the value of the objective function. The contract always keeps track of the best feasible solution submitted so far, which we call the *candidate solution*.
- At the end of each time interval t, the contract finalizes the trade values for interval $t_f = t + T_{clear} + 1$ based on the candidate solution.[15]

2.7 TRANSAX Protocol

We implement the practical solution approach described in the previous section as a protocol of interaction between the TRANSAX components (Section 2.5). The protocol is depicted in Fig. 2.4, and the activities are described below.

[14]Further, Solidity, the preferred high-level language for Ethereum, currently lacks built-in support for certain features that would facilitate the implementation of a linear programming solver, such as floating-point arithmetic [31].

[15]If no solution has been submitted to the contract so far, which might be the case right after the trading system has been launched, $\varepsilon = \mathbf{0}$ may be used as a candidate solution.

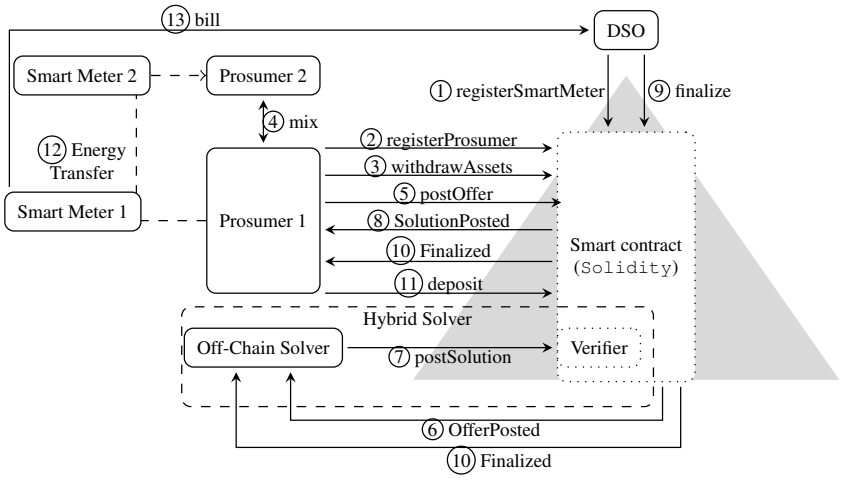


Figure 2.4: Example workflow of TRANSAX. Nodes represent entities in the platform, and edges represent interactions, such as smart-contract function calls. In this example, prosumer 1 is selling energy to prosumer 2 and the dashed line represents the energy transfer.

2.7.1 Registration

When a new customer is added to the grid, a smart meter is installed. The DSO registers the smart meter by calling ①[16] *registerSmartMeter* on the TRANSAX smart contract. This call sets the asset allocation limits for that customer and records which feeder it is located on in the grid. The customer then registers as a prosumer with TRANSAX by calling ② *registerProsumer*.

The registration information requires each prosumer to specify a smart meter, and to provide a DSO certified public address that corresponds to the specified smart meter for the DSO to use when allocating assets. Since the smart meter is associated with a specific feeder, the smart contract adds the prosumer to the group associated with that feeder. This is required to ensure that feeder-level safety constraints can be correctly applied. The registrations can happen asynchronously, allowing new prosumers to join at any time, even long after trading has commenced. The registration process occurs only once for each smart meter and prosumer. Once registered, a prosumer may participate in the following trading

[16]The circled numbers correspond to the numbered edges in Fig. 2.4

protocol repeatedly.

2.7.2 Mixing

Once a prosumer has registered, it can withdraw assets into its public address (*i.e.*, the account registered at DSO) for future intervals by calling ③ *withdrawAssets*. After withdrawing assets, a prosumer could make offers using *postOffer*. However, if it made offers using its public account, then the trades could be traced back to the prosumer as all transactions in the distributed ledger are recorded publicly, thereby violating the privacy requirements. Instead, the prosumer creates an anonymous address, which is not registered with the DSO, and transfers the assets from its public address to the anonymous addresses via ④ mixing assets with other prosumers. Mixing can be done in assigned groups by executing a decentralized mixing protocol, such as CoinShuffle [22]. The goal of the mixing protocol is to transfer funds or assets from a set of accounts to a set of anonymous accounts without directly linking any of the accounts to each other. Due to this mixing, even if an entity knows which prosumers participated in a mixing protocol (*i.e.*, based on their registered, public accounts) and what target anonymous accounts were used in the mixing, it cannot link any anonymous account to the prosumer who owns the account.

2.7.3 Trading

2.7.3.1 Posting Offers

Next, the prosumers can construct and post anonymous offers using their anonymous accounts by calling function ⑤ *postOffer*. The smart contract checks that the anonymous account used to post the offer has assets that cover the amount and intervals specified in the offer. If not, then the offer is rejected. If the offer is accepted, the smart contract emits event ⑥ *OfferPosted*, notifying the off-chain matching solvers.

2.7.3.2 Matching Offers

The matching solvers may wait for many prosumers to post many offers, but eventually, it pairs buying and selling offers and posts the solutions by calling function ⑦ *postSolution*. The smart contract checks the solution to make sure that it is feasible according to the feasibility requirements described in Section 2.6, including checking that the trades do not exceed the group capacity constraint. If the solution is valid, then the smart contract saves it and emits event ⑧ *SolutionPosted*, notifying the prosumers of the current candidate solution. Additional solutions may be submitted by any solver, and if those solutions are valid and superior (*i.e.*, they trade more energy), then the smart contract will update the candidate solution. Offers can continue to be posted until the end of the trading interval when trades will be finalized.

2.7.4 Energy Transfer and Billing

As an interval comes to a close, the DSO calls[17] function ⑨ *finalize* which means that offers for interval t_f are no longer accepted and the smart contract transfers funds from the consuming offer's account to the producing offer's account. It also exchanges the *EPA* assets of the seller for the *ECA* assets of the buyer and vice versa for each of the matched offers. The call also emits the ⑩ *Finalized* event, notifying the solvers to update their solving interval, and the prosumers that the trades for interval t_f have been finalized. If a prosumer posts offers with many anonymous accounts, it will have to aggregate all the corresponding trades to determine how much energy it is expected to produce/consume during that interval when it arrives. Once the prosumers are notified of the trades, they call function ⑪ *deposit* to transfer all assets for the finalized interval from the prosumers anonymous accounts to an anonymous account owned by their smart meter.

The smart meter checks that the total amount of assets deposited matches the amount

[17] Note that by default the DSO calls the *finalize* function to increment the current interval, but since this function is time guarded, any other entity can call it, which provides additional resilience.

withdrawn for the finalized interval. This ensures that there are no trades that have not been accounted for. The smart meter also compares the total of all production assets that were deposited against the production originally withdrawn to compute the net energy sold ($\Delta EPA = EPA_{deposit} - EPA_{withdraw}$). When interval t arrives and the energy transfer begins (12), deviations from the allocated trades are covered by the DSO, including deviations due to prosumer failures. To provide billing information for the DSO, the smart meter must measure the deviations. To this end, it measures the net energy production E_u^t (negative values represent net consumption) of prosumer u at time interval t. The smart meter then computes the difference between the net energy sold and the net energy production to get the residual production (again, negative values are residual consumption). The residual production or consumption is multiplied by the selling or buying price of the DSO, respectively, to calculate what the prosumer owes the DSO for each interval. Every (13) billing cycle, the smart meter sums the cost of the residuals and sends that to the DSO for the monthly bill. The bill B_u^t of prosumer u for timeslot t, which will be paid by the prosumer to the DSO, is

$$B_u^t = \begin{cases} (E_u^t + \Delta EPA) \cdot \pi_t^S & \text{if } E_u^t + \Delta EPA < 0 \\ (E_u^t + \Delta EPA) \cdot \pi_t^B & \text{otherwise,} \end{cases} \tag{2.16}$$

where π_t^S is how much the DSO pays to purchase energy and π_t^B is how much the DSO charges for energy. The price schedule is set for each timeslot t by the DSO. The prices could be functions of $E_u^t + \Delta EPA$ to charge higher rates as the deviation from the traded amount increases. By designing the DSO prices to vary based on the deviation from the amount traded, we can provide strong incentives to prosumers to predict energy production and consumption accurately and to post conservative offers, so that the DSO and other prosumers can adjust their production or consumption preemptively, reducing the balancing that the DSO must provide due to unanticipated demand.

Table 2.3: Summary of Component Functionality in TRANSAX

Requirement	Components	Approach
Security and Safety	Distribution System Operator and Smart Contract	DSO sets trading limits for prosumers and feeders, and the smart contract enforces them.
Resilience	Distributed Ledger and Hybrid Solver Architecture	A distributed ledger is resilient because of its distributed nature. Solvers are replicated to provide resilience.
Efficiency	Smart Contract and Solvers	Problem formulation allows temporal flexibility, a smart contract enforces choosing the best solution.
Privacy	Prosumers	Prosumers achieve privacy via a mixing protocol.

2.8 Discussion and Analysis

In this section, we first describe how the TRANSAX design ensures the security, resilience, and safety of the system. Then, we provide a discussion on the inherent trade-offs between efficiency, and privacy. Table 2.3 summarizes how each component in the architecture contributes to satisfying the system requirements.

2.8.1 Requirement Evaluation

2.8.1.1 Security and Safety

The underlying blockchain platform provides basic security features, so we are not concerned with the operations occurring on the blockchain. We are concerned with the secure and reliable operation of the solver. Similarly, the basic safety of the system is handled by the constraints described in Section 2.6.2. The safety constraints are applied correctly and reliably by the same contract. An adversary cannot force the contract to finalize trades based on an unsafe (*i.e.*, infeasible) solution since such a solution would be rejected. Similarly, an adversary cannot force the contract to choose an inferior solution instead of a superior one. In sum, the only action available to the adversary is proposing a superior feasible solution, which would improve energy trading in the microgrid.

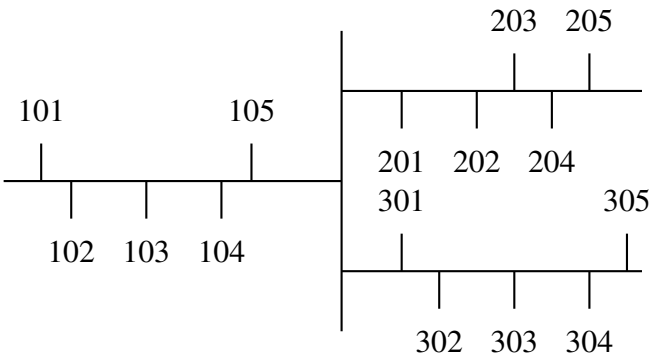


Figure 2.5: Topology of a distribution network.

2.8.1.2 Resilience

Now we show that our contract is reliable and can tolerate temporary disruptions in the DSO, solvers, or the communication network. First, since the *finalize* contract function is time guarded any entity can call it, and the system can progress without a DSO which is only required for registering new prosumers and their smart meters. Second, notice that any solution (ε, π) that is feasible for sets $\mathscr{S}$ and $\mathscr{B}$ is also feasible for supersets $\mathscr{S}' \supseteq \mathscr{S}$ and $\mathscr{B}' \supseteq \mathscr{B}$. As the sets of offers can only grow over time, the contract can use a candidate solution submitted during time interval t to finalize trades in any subsequent time interval $\tau > t$. In fact, without receiving new solutions, the difference between the amount of finalized trades and the optimum will increase only gradually: since the earlier candidate solution can specify trades for any future time interval, the difference is only due to the offers that have been posted since the solution was found and submitted. Thus, the system can continue making trades using older valid solutions

2.8.1.3 Trading Efficiency

The trading platform we have presented can support efficient trading through temporal flexibility. We show this through Example 1. As a reminder, this is due to prosumers being able to specify their production/consumption capacities and preferences (*i.e.*, reservation

prices) via offers and the linear-program finding an optimal matching. In Section 2.9.3, we show using simulation that energy trading reduces the load on the power grid.

Example 1. *Consider two prosumers (denoted by* 102*,* 103*) and one consumer (denoted by* 101*) from the community depicted in Fig. 2.5. We divide each day into 15-minute intervals. Let us assume that* 102 *can transfer* 10 *kWh into the feeder during interval 48, which translates to 12:00 pm – 12:15 pm. Assume similarly that* 103 *can also provide* 30 *kWh to the feeder in interval 48, but it has battery storage. Since* 103 *has battery storage—unlike* 102*, who must either transfer the energy or waste it—*103 *can delay the transfer until a future interval,* e.g.*, interval 49. Now suppose that* 101 *needs to consume* 30 *kWh in interval 48 and* 10 *kWh in interval 49. A possible solution would be to provide all* 30 *kWh to* 101 *from* 103 *in interval 48. However, that will lead to the waste of energy provided by* 102*. Thus, a better solution will be to consume* 10 *kWh from* 102 *in interval 48 and* 20 *kWh from* 103 *in interval 48. Then, transfer* 10 *kWh from* 103 *in interval 49, which is more efficient than the first matching as it allows more energy (summed across the intervals) to be transferred. Thus, we see that permitting temporal flexibility can significantly increase trading volume, though it does increase the size of the optimization problem, increasing computational complexity.*

2.8.1.4 Privacy

The platform provides pseudo-anonymity as the individual offers cannot be tied back to the prosumer who posted them since the offer is only affiliated with an anonymous address and contains only the energy amount and reservation price. Additionally, the DSO does not know the total amount of energy utilized by the prosumers thanks to the anonymous billing via the smart meter. However, to preserve safety, some information about the prosumers needs to be public to allow checking of the offers to ensure that they are safe or limit the resources available to them.

In our design, we assume that the consumption (L^-) and production (L^+) limits of each

prosumer are public information, as well as which feeder a prosumer is on. The group safety constraints C_g^i and C_g^e are also public. Recall that the smart contract ensures that no prosumer can withdraw more assets than the specified limits and that any offer which violates the recorded safety constraints will be rejected. As a result, the only way to violate the safety requirements is if the asset limits or safety constraints are set incorrectly, which is not allowed by our design. However, as we will show below it is possible to improve privacy by choosing a conservative safety constraint for a group or a conservative limit on the maximum assets a prosumer can withdraw, which impacts the trading efficiency. Consider the following example for illustration.

Example 2. *Consider the community depicted in Fig. 2.5. Let the prosumers denoted by* 102 *and* 103 *form a group g with an internal constraint of* $C_g^i = 40$*, where prosumers 102 and 103 have asset limits* $L^+ = 10$ *and* $L^+ = 30$*, respectively. Assume that the prosumers in this group have anonymized their assets. If the total assets traded by the group—which we denote* T_f*—is below* 10*, then there is no way to definitively say that either prosumer is trading. If the assets traded by* f *exceed* 10*, then we know that* 103 *is trading at least* $T_f - 10$ *since* 102 *can only produce* 10*. If* $T_f > 30$*, then we know that* 102 *is trading at least* $T_f - 30$*. If* $T_f = 40$ *or* 0*, then we know the full state of the feeder, either both prosumers are trading at their limit or not trading at all. To improve anonymity, the feeder as a whole should not trade more than* 10*. This however reduces trading efficiency considerably. Nonetheless, if both prosumers have* $L^+ = 20$*, then anonymity is improved until trading exceeds* 20*. Thus, it is important to select the constraints carefully. We discuss this in Section 2.8.2.*

2.8.2 Tradeoff between Privacy and Efficiency

Note that the safety of the system is a strict requirement, which we cannot compromise. Thus, the only plausible tradeoff is between privacy and efficiency. This tradeoff can be achieved by creating groups, as we discussed in Section 2.6.3. However, groups

and constraints must be created and set carefully to ensure that trading remains safe while also minimizing the loss in trading potential[18]. To better understand this problem, consider that when a group is mapped to a single physical feeder, the safety constraints are simply the feeder's constraints. However, in a group, we cannot tell which feeder the accounts belong to once the accounts are anonymous. Thus, to preserve safety, the constraints need to be adjusted. Therefore, the set of feeders are transformed into a group by treating all the prosumers in those feeders as if they were on a common feeder. Since the offers are anonymous at the group-level, the system can treat the group as a single feeder with two prosumers: one which posts production offers and one which posts consumption offers (see Fig. 2.6).

To describe the methodology for selecting group constraints and the corresponding cost of privacy, we need to consider two cases.

2.8.2.1 Case 1 - There is a set of prosumers in the group that is capable of exceeding the safety constraint of the feeder they are on:

Assume a microgrid with feeders $\mathscr{F}$ and groups $\mathscr{G}$, wherein L_u for each prosumer[19] and C_f for each feeder can have any value. Recall that we only need to consider $C_g^e \leq C_g^i$. For now, we consider when both constraints are violated simultaneously, setting $C_g^e = C_g^i$, and refer to it as the *feeder safety limit* C_g. For this system to be safe, the following condition on C_g must hold for every group g:

$$C_g \leq \min \left\{ C_f \middle| f \in g \text{ and } \sum_{u \in f} L_u \geq C_f \right\} \tag{2.17}$$

[18]The downside of grouping is that common feeder groups may result in lower energy trading limits due to modified aggregated constraints. We call this efficiency loss the cost of privacy.

[19]Note that L_u^+ and L_u^- (Table 2.2) are the same type of constraint, representing production/outgoing or consumption/incoming limits, so we will use L_u to represent both in our analysis, but in each case the equations refer to both.

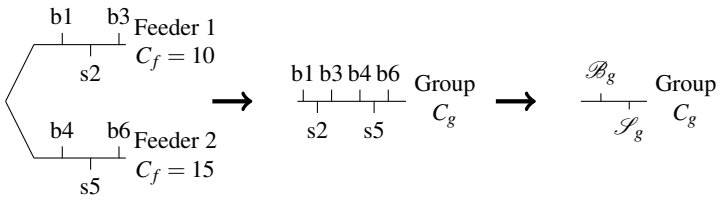


Figure 2.6: Feeder conversion diagram.

Proof. For the sake of contradiction, suppose that Equation (2.17) does not hold, but the system is safe. This means that $\exists C_f < \sum_{u \in f} L_u$ and $C_g > C_f$. Let $\sum_{u \in f} L_u = C_g$. Then, the prosumers in f can trade EL assets. However, this exceeds the feeder safety limit, so the system cannot be safe. Equation (2.17) must therefore be true. □

Thus, the best value for the group constraint is when Equation (2.17) is equality. This means that the group as a whole can at most produce the same amount as the single smallest of its internal feeders. The cost in this case is:

$$cost = \min\left\{\sum_{\forall s \in \mathscr{S}_g} E_s, \sum_{\forall b \in \mathscr{B}_g} E_b\right\} - \min\left\{\sum_{\forall s \in \mathscr{S}_g} E_s, \sum_{\forall b \in \mathscr{B}_g} E_b, C_g\right\}. \tag{2.18}$$

Thus, the cost is the amount by which the potential trades exceed the safety constraint.

2.8.2.2 Case 2 - No set of prosumers in any of the feeders in the group are capable of exceeding their feeders' safety constraint:

Given a microgrid with feeders $\mathscr{F}$ and groups $\mathscr{G}$ where C_f can have any value and

$$\forall_g \forall_{f \in g} \sum_{u \in f} L_u \leq C_f, \tag{2.19}$$

group constraint should be set as $C_g = \sum_{f \in g} C_f$ to maximize trading, and trades can be done safely.

Proof. Assume a microgrid is not safe and Equation (2.19) is true. Then, $\exists f$ such that $\sum_{u \in f} L_u > C_f$. But, Equation (2.19) says this is not allowed. So, the system is safe. □

In this case, there is no cost to group privacy. Safety is ensured by the asset withdrawal limits rather than the group constraint. Note that Case 1 can be converted to Case 2 by reducing the prosumer asset limits so that the prosumers on a feeder cannot exceed their feeder's safety constraint[20]. To compute the cost of this conversion, instead of setting $C_f^i = C_f^e$ as we did in Case 1, we let $C_f^i > C_f^e$. This means that without privacy, the amount of energy that can safely be traded within the feeder is greater than the amount of energy that can be traded with other feeders. In this case, the maximum amount of energy that could potentially be traded is C_f^i. Even if the prosumers could exceed the internal constraint, those trades would not be permitted, so they are not a loss. Therefore, we need to consider only the trades that could have been made but are no longer permitted, which is at most $C_f^i - C_f^e$.

2.8.2.3 Insights on Grouping

Based on the analysis of the effects of privacy on efficiency, the best strategy is to limit the trading assets of the prosumers such that they remain less than the feeder constraints. This means that all feeders can be safely grouped. The cost of grouping feeders is the loss of flexibility in trading due to the rigid asset limits. The cost will be at most the feeder limit minus the prosumer asset limit, if that prosumer can reach the feeder limit, and if no other prosumers in its feeder are trading. This could be mitigated by an additional mixing and trading step within the feeder, but we have not examined this possibility in detail. There is a second criterion that may influence grouping decisions. There is information leakage, and at the extremes (max load, zero load) anonymity ceases to exist. We assume that generally this will not be the case, and the odds of that occurring diminish if there are many feeders in the group. Information leakage can be reduced by setting all the asset limits to the same value for all prosumers. The maximum system cost of this is the difference between the feeder limit and the sum of the prosumer limits. To reduce information leakage, groups

[20] This can be enforced by the DSO during installation.

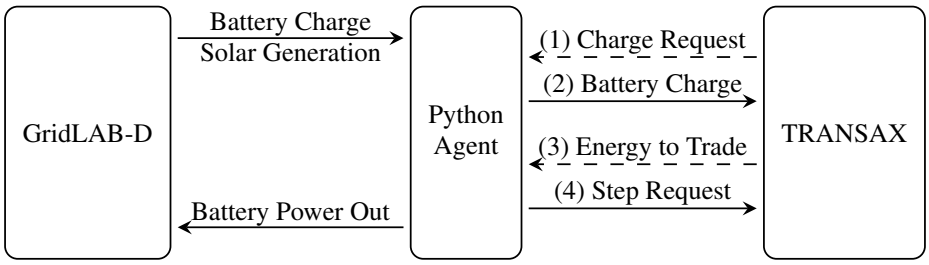


Figure 2.7: Messages exchanged between simulator and TRANSAX.

should consist of feeders with similar limits.

2.9 Experimental Evaluation

In this section, we present a simulation testbed [21] that we developed for evaluating TRANSAX, as well as our initial results illustrating the effectiveness of TRANSAX in reducing the load on the bulk power grid.

2.9.1 Testbed

The system to demonstrate the simulation platform has three major parts as shown in Fig. 2.7: the TRANSAX nodes (BeagleBone Blacks[22]) emulating the prosumers[23], the distribution system physics simulator (GridLAB-D [66], running on an x86 computer with a Core i7 processor and 24GB of RAM), and a Python agent to coordinate the hardware in the loop (emulated TRANSAX prosumers) integration with GridLAB-D. Messages and time steps between the Python agent and GridLAB-D are coordinated by the Framework for Network Co-Simulation [73].

The general message structure between GridLAB-D, the Python agent, and TRANSAX is shown in Fig. 2.7. While GridLAB-D is paused, TRANSAX agents request charge sta-

[21] The source code of the testbed is available at https://github.com/scope-lab-vu/transactive-blockchain

[22] With limited computational capability and ARM architecture, these nodes are a good representation of embedded devices that we can expect to be used in real scenarios for managing energy trading within communities.

[23] The control logic of prosumers is implemented in RIAPS.

tus for their batteries in the GridLAB-D simulation. They use this data, along with their predicted energy usage, to create a bid that is sent to TRANSAX. TRANSAX agents send the finalized trades back to the Python agent. The Python agent sets each simulated node's output for the next interval based on the finalized trades from TRANSAX by modifying GridLAB-D system parameters. In this demonstration, the Python agent meets the finalized trades only by modifying battery outputs. However, the Python agent has control over all the dynamically modifiable parameters in GridLAB-D. Consequently, future demonstrations could incorporate more control parameters, such as curtailments to solar output or curtailments to energy used by pure consumers.

The most important feature of this demonstration is its methodology for synchronizing time between GridLAB-D and TRANSAX, which is also responsible for time synchronization between GridLAB-D's variable-timestep solver and TRANSAX's matching solver. In the experiments described below, we use a solver period of 15 minutes. Thus, the Python agent forces GridLAB-D's variable-timestep solver to pause at each logical 15-minute interval. Then, the prosumer nodes post offers for each 15-minute interval of logical time, and TRANSAX clears and finalizes trades. Next, the GridLAB-D simulation is advanced with actual energy transfer, allowing the impact to be measured. This process is repeated for the duration of the simulation's logical time. The time-synchronization strategy is scalable to any desired period for the TRANSAX solver. The strategy also provides freedom to run experiments, such as assessing how the solver's period affects the amount of energy traded, the stability of the finalized trades, or computational cost. Note that all physical nodes in the setup are time-synchronized using the services provided by RIAPS [67] (Section 2.4).

2.9.2 Simulated Scenario

We run our simulations on the distribution topology described earlier in Fig. 2.5. It consists of a substation feeding three main overhead lines that are connected to prosumers. The lines below the main lines represent prosumers with batteries and solar panels, which

enables them to either consume or produce energy depending on the net output of the solar panels and batteries; and those above the main line represent prosumers with loads only (*i.e.*, they can never produce). For the demonstration, the simulation was built with 9 producer nodes and 6 consumer nodes.

The simulation was set up to run in logical time from 8 AM to 8 PM of the same day, for a total duration of 12 hours. During our experiments, we sped up the simulation by letting the real-time length of the time interval be $\hat{\Delta} < \Delta$ where $\hat{\Delta}$ is 2 minutes and Δ is 15 minutes. Note that $\hat{\Delta}$ is the amount of real time passed in the simulation before proceeding to the next interval; this allows us to speed up the experiments without compromising our results since running the system slower would be easier.

2.9.3 Results

We now discuss the results of three sets of experiments. The first experiment studies the impact of the solver horizon window T_h (Table 2.2). The second experiment demonstrates the benefit of the platform to the DSO. Finally, the third set of experiments studies how inaccurate predictions of energy consumption and productions affect the DSO.

2.9.3.1 Experiment 1 - Impact of T_h

It is expected that a longer time horizon will allow the TRANSAX solver to be more efficient and better match the producer and consumer offers. However, there is a tradeoff because a longer horizon also leads to higher computational cost. Thus, we varied the value of T_h and measured the memory usage, CPU usage, and amount of energy traded. In Fig. 2.8, we see that as the time horizon increases, so does the memory usage and energy traded until $T_h = 30$, at which point there is no additional gain to energy traded. The time horizon also impacts the CPU utilization of the solver (not shown). This demonstrates that we can select a finite time horizon and still obtain high-quality solutions.

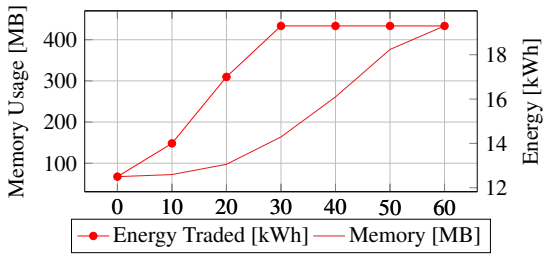


Figure 2.8: Memory consumption and energy traded during a single interval of the simulation for various values of T_h using the CPLEX solver.

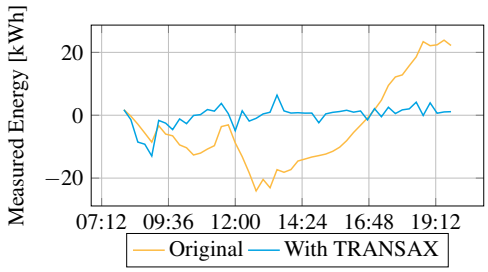


Figure 2.10: DSO load with and without TRANSAX.

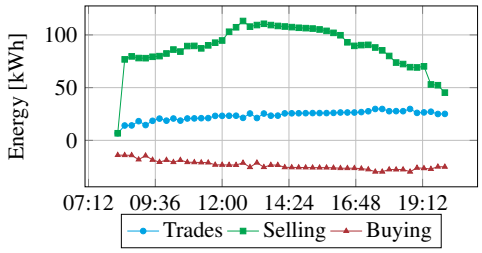


Figure 2.12: Green: sum of all production offers for each interval. Red: negative sum of all consumption offers for each interval. Blue: sum of all energy traded in each interval, whose maximum value is the minimum of the production and consumption offers.

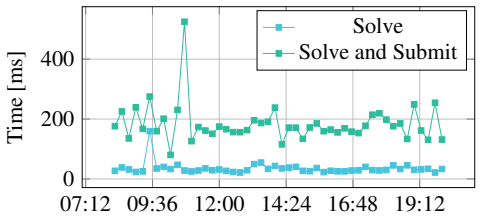


Figure 2.9: *Solve* time is how long it took the solver to find a solution to the energy trading problem. *Solve and Submit* time is how long to took to find the solution and submit it to the smart contract.

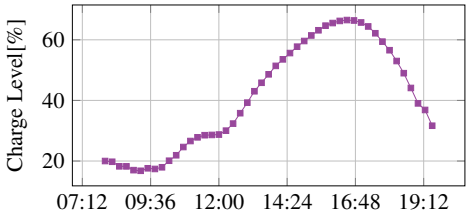


Figure 2.11: Average battery charge level.

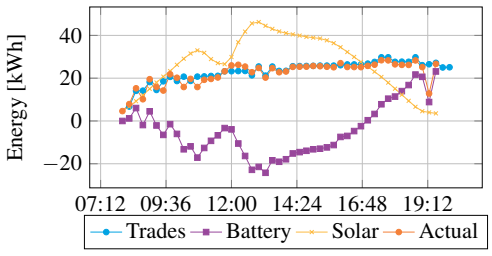


Figure 2.13: Yellow: simulated solar profile. Purple: simulated battery charge level. Orange: simulated energy traded. Blue: total energy trades recorded in the market.

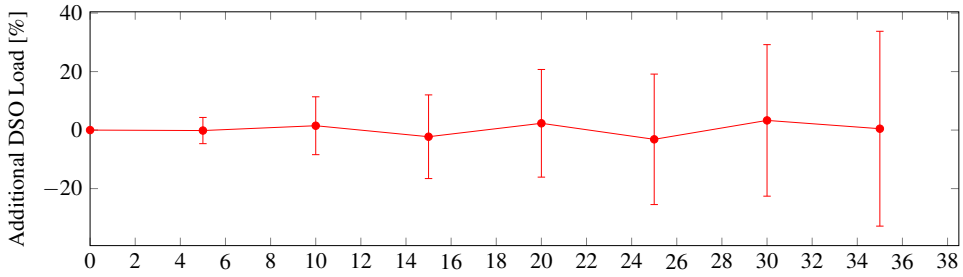


Figure 2.14: Average additional load on DSO per day (calculated across 100 days) due to the difference between actual trades and the offers.

2.9.3.2 Experiment 2 - Impact of TRANSAX on Load Serviced by DSO

To determine the impact of trading on the load supplied by the DSO, we ran several simulations. The simulation was first run without battery output and without any control by TRANSAX. This output was used to generate an energy profile for each prosumer for each interval. Then, the simulation was repeated with the prosumers submitting offers matching their energy profile to the TRANSAX system, which represents an ideal scenario with accurate bids for each 15-minute interval. This simulation demonstrates how batteries can reduce the load on the DSO. Fig. 2.10 shows the comparison of DSO (utility substation) loads with and without TRANSAX. The horizontal axis is the simulated time since the start of the simulation. The vertical axis shows the load on the substation, negative values mean that the prosumers' generation exceeds their loads. Without TRANSAX, solar generation begins to outproduce the total load within the first interval. Solar production reaches its peak around 12:45 PM. Finally, at 4:45 PM, the load exceeds solar production, and the substation load becomes positive. The inclusion of TRANSAX dramatically reduces the need for the substation backup. From 8:00 AM to 4:45 PM, the overproduction of solar meant that the batteries were charging, which mitigated the negative load on the substation. After 4:45 PM, the batteries discharged and mitigated the positive load on the system. Fig. 2.11 shows the average battery charge level across all 9 batteries. At the end of the

simulation at 8 PM, the average battery had only around 25% charge. This means that if the simulation were to go further into the night, there would not have been enough battery charge left to meet demand for the entirety of the night.

Fig. 2.12 shows the total amount of energy offered for each interval, as well as the total amount of energy recorded in trades. In Fig. 2.13, we see that the trades recorded (blue) are mostly consistent with the measured load (orange) on the system, with one notable exception at 2:15 PM. The deviations occur because the prosumers currently assume that solar output remains constant over each interval, and this constant value is used when making offers. Fig. 2.9 shows the time required by the platform to find the optimal matching of a set of offers (green), as well as that time combined with the time required to submit that solution to the smart contract (blue). Most of the time spent is due to smart contract communications.

The results of the simulation with TRANSAX are promising. TRANSAX found energy trade solutions that resulted in an overall reduction of substation load. The distribution was however not completely independent of the substation feeder, and there is still a need for a connection to the larger distribution grid through a DSO.

2.9.3.3 Experiment 3 - Impact of Imprecision of Offer Prediction on DSO

Since prosumers must estimate their future production and consumption, we are interested in assessing the impact of estimation uncertainty on the stability of the system and on the load on the DSO. In an ideal case, prosumers will provide or consume the same amount of energy as they offered, and the DSO will know in advance how much energy it must provide to compensate for system stability. Any variations from the offered amount of energy result in uncertainty for the DSO. To study the effect of this uncertainty, we created scenarios where we added normally distributed error to energy produced or consumed by each prosumer (relative to the settled offers). The standard deviation of the error was scaled as a percentage of the prosumer's largest expected trade in a day, ranging from 0% to 35%

of the energy traded. We chose this value because there are models for short-term cloud forecasting that have estimation errors of 20-30% for 15 to 45 minutes in the future [74]. We chose 35% as the upper bound. Fig. 2.14 shows the average daily difference between the energy that the DSO anticipated to provide from finalized trades and the energy it actually provided as a function of prosumer uncertainty. The averages were calculated over 100 simulated days. The uncertainty for the DSO increases with uncertainty in prosumer energy production and consumption. The standard deviation of the uncertainty for the DSO is 33% of the anticipated DSO load when prosumer trades are uncertain by a standard deviation of 35% of the offers; however, the average additional load remains near zero. The experiment demonstrates that while uncertainty in the offers will result in errors in the amount traded and eventually cause some uncertainty for the DSO, the net difference will remain small if the error is normally distributed.

2.10 Related Work

Transactive strategies manage generators and loads based on market dynamics while ensuring system reliability. The earliest example of transactive control was demonstrated in the Olympic Peninsula Project [75]. An extension of these controls is seen in the management of building energy consumption [76]. Recently, with well-known grid failures, including the 2012 Sandy Storm and 2017 hurricane Maria in Puerto Rico there have been concerted efforts to build decentralized energy systems with transactive components [77, 78]. However, the existing platforms are not fully operational and, in most cases, cannot satisfy the three conflicting requirements of resilience, privacy, and safety.

Existing energy trading markets, such as the European Energy Exchange [79] and project NOBEL in Spain, involve centralized database architectures that constitute single points of failure. The closest to a decentralized implementation that is required for resilience is Wörner et al. [52], who have developed an implementation of their peer-to-peer energy market and deployed it in a town in Switzerland. Their goal is to gather empiri-

cal evidence to answer the question of what the benefits of a blockchain system are in the electricity use case. However, the results have not yet been published. Similarly, the next phase of the LO3 project in Brooklyn [51] is working on extending the blockchain-based energy market. However, to the best of our knowledge, their focus has primarily been using blockchain as the resilient information store, and they are not using the decentralized architecture to implement a market. The blockchain there is simply a medium to store renewable energy attributes.

Since a decentralized transactive energy system consists of various components including the markets, the controller, and privacy mechanisms, we discuss them in detail below.

2.10.1 Markets

After prosumers share their energy availability and demands in the form of offers, these offers need to be matched. Researchers have proposed two approaches to this problem.

2.10.1.1 Stable Matching

Stable matching refers to a matching of all possible buy and sell offers in a bipartite graph. Yucel *et al.* proposed a homomorphic encryption-based position hiding method [80] which protects users' privacy. Nunna *et al.* [81] proposed the symmetrical allocation problem based on a native auction algorithm to match buyers and sellers. This algorithm is run periodically, and only one selling offer is matched in each round. PowerLedger [82] uses another mechanism to match offers. Offers are broken into equal portions and matched together e.g., when a new consumer arrives, it receives the equal allocation from the energy pool in the area.

2.10.1.2 Auction

Another approach to match buyers and seller offers is to use auctioning approaches. Majumder *et al.* [83] proposed a double auction mechanism before the era of blockchains

where the controller doesn't need the users' preferences, but instead, they use an incentive-compatible auction mechanism to extract that information in the form of bids. In the era of blockchains, Kang *et al.* [84] and Guerrero [85] used double auctions to match parties and not goods in blockchains. To ensure the integrity of the results of matching, Wang [86] proposed a multi-signed digital certificate. Khorasany *et al.* [87] designed a greedy algorithm with the averaging auction mechanism to match buyers with higher prices to sellers with lower prices. Zhao *et al.* [88] created a two-phase auctioning algorithm to find the optimal pricing for bids. Finally, Zhang *et al.* [89] developed a non-cooperative auctioning game and used it to find the optimal solution for the matching problem using the Nash equilibrium.

2.10.2 Grid Control and Stability

Microgrid controllers are an integral part of smart grids. They ensure the stability and resiliency of the microgrid. They enable the transition of the microgrid from grid-connected to islanded [90, 91] so that the failures in the grid do not cascade to other areas similar to the outage event back in 1999 in Sao Paulo, Brazil [92]. Currently, most microgrid controllers are centralized [47] which are vulnerable to cyber-threats and privacy issues. A large spectrum of cyber-threats are applicable on centralized microgrid controllers with single-point-of-failure ranging from attackers eavesdropping on channels between the controllable resource and the centralized controller to steal critical information of the users or network infrastructure, performing DDOS attacks on the centralized controller, or manipulation of demand via IoT (MadIoT) attacks[93] to injecting malware into the market operation system and manipulate settings, such as DLMP limits or clearing time interval similar to the notable cyber-attack against Ukrainian power systems in December 2015 [94, 95].

Due to these drawbacks of centralized grid controls, the industry is transforming from centralized to decentralized [96, 97]. TRANSAX aims to create a decentralized transactive energy market that ensures the privacy and security of users while maintaining the stability

and resiliency of the grid.

2.10.3 Security and Privacy

2.10.3.1 Communication Security

The first step to preserve users' privacy and anonymity in a distributed system is to provide communication privacy. Without this, an adversary can discern who is making a function call or sending a message over the network based on the sender's MAC address, IP address, or route to destination. Existing protocols for low-latency communication anonymity include onion routing [98], the similar garlic routing [99], STAC [100], and the decentralized Matrix protocol [24]. However, Murdoch and Danezis [101] show that a low-cost traffic analysis is possible of the Tor-network, theoretically and experimentally. Communication security is an orthogonal research problem to TRANSAX.

2.10.3.2 Address Anonymity

Communication anonymity is necessary but not sufficient for anonymous trading, as the cryptographic objectives of authentication and legitimacy are not fulfilled. We suggest using cryptographic techniques from distributed ledgers, *blockchains*, and cryptocurrencies. The most adopted one, Bitcoin allows for very simple digital cash spending but has serious privacy and anonymity flaws [102, 103, 104]. Additionally, Biryukov and Pustogarov, 2015, show that using Bitcoin over the Tor network opens a new attack surface [105]. Solutions to the tracing and identification problems identified by these researchers have been proposed and implemented in alternative cryptocurrency protocols: mixing using ring signatures and zero-knowledge proofs [106, 107].

A proposed improvement to standard ring signatures is the CryptoNote protocol, which prevents tracing assets back to their original owners by mixing incoming transactions and outgoing transactions. This service hides the connections between the prosumers and the

[24] https://matrix.org/docs/spec/

addresses. Mixing requires the possibility to create new wallets at will and the existence of enough participants in the network. Monero is an example of a cryptocurrency that provides built-in mixing services by implementing the CryptoNote protocol [21]. There are however alternative implementations of mixing protocols such as CoinShuffle [22] or Xim [23]. A variant of ring signatures, group signatures, were first introduced by Chaum and van Heyst, 1991, [108] and then built upon by Rivest *et al.*, 2001 [109]. The basis for anonymity in the CryptoNote protocol, however, is a slightly modified version of the *traceable ring signature* algorithm by Fujisaki and Suzuki, 2007 [110]. This allows a member of a group to send a transaction so that it is impossible for a receiver to know any more about the sender than that it came from a group member without the use of a central authority.

Some newer cryptocurrencies, such as Zerocoin [107], provide built-in mixing services, which are often based on cryptographic principles and proofs.

2.10.3.3 Smart Meters' Privacy

Most works discussing privacy look at it from the context of smart meters. McDaniel and McLaughlin discuss privacy concerns due to energy-usage profiling, which smart grids could potentially enable [111]. Efthymiou and Kalogridis describe a method for securely anonymizing frequent electrical metering data sent by a smart meter by using a third-party escrow mechanism [112]. Tan et al. study privacy in a smart metering system from an information-theoretic perspective in the presence of energy harvesting and storage units [113]. They show that energy harvesting provides increased privacy by diversifying the energy source, while a storage device can be used to increase both energy efficiency and privacy. However, transaction data from energy trading may provide more fine-grained information than smart meter-based usage patterns [114].

2.11 Conclusions and Future Work

In this paper, we described TRANSAX, a decentralized platform for implementing energy exchange mechanisms in a microgrid setting. Building on top of blockchains, we obtained decentralized trust and consensus capabilities, which prevent malicious actors from tampering with the shared system state. We found that satisfying the seemingly conflicting goals of safety and privacy can be reconciled using anonymity within a grid, though this may result in a loss of flexibility and trading volume if the prosumers within a feeder could exceed the feeder's limit. Using our hybrid-solver approach, which combines a smart-contract based validator with an open set of external solvers, we showed that we can clear offers securely, efficiently, and resiliently, submitting solutions to the contract within approximately 200ms. We also demonstrated using TRANSAX that private blockchain-based transactive energy is feasible for communities on the scale of microgrids and smaller, though we have not determined the upper limit for scalability. We can ensure that trades are balanced and that energy trading can the load on the DSO.

In the current implementation, we have not chosen a specific approach for *setting the clearing prices* for the prosumers' trades since the economics of setting the clearing prices is an orthogonal problem. Friedman and Rust [115] provide a survey of these mechanisms for governing trade, to which they refer as market institutions. One of the most commonly used mechanisms is the double auction. Note that we cannot apply the double auction directly because of the different time-interval attributes that the offers may specify. Prior work has extended the double auction to allow for multiple attributes; however, they typically (*e.g.*, [116]) require a function to combine the attributes into a single value, which is then used to order the offers. The difficulty of this approach is in identifying a meaningful function. A more straightforward approach is to perform the feasibility matching as we have presented, and then for each interval, use a double auction to set the clearing price for the matched offers. This approach provides a straightforward solution to the problem of setting clearing prices; however, it is not obvious whether it will preserve the properties that

a simple double auction has, such as incentive compatibility. We leave the investigation of these mechanisms and how they are impacted by privacy to future work.

Further, we need to allow prosumers to *update or cancel offers*. The current formulation can support updating offers as long as the updates do not invalidate previous solutions; for example, a selling offer can increase the amount of energy to be sold or augment the set of intervals in which energy could be produced. To support restrictive changes or canceling offers, we would need to introduce a deadline for when offers could no longer be updated or canceled. Solvers could then wait for this deadline and start working only after the deadline. Lastly, in the current implementation, the DSO provides the missing energy when a prosumer fails; however, we may also consider the case when the DSO is not available. In this case, a potential solution is to maintain backup energy reserves to satisfy demand that was unmet due to a failure.

Chapter 3

Managing Cheating Through Blockchains – An Edge Computing Case-Study

3.1 Overview

In addition to the standard challenges associated with CPS and other distributed systems, multi-stakeholder cyber-physical systems (MSCPS) present significant additional concerns with respect to *trust*. Because, by definition, MSCPS are composed of multiple agents which may malfunction or have conflicting objectives. Efficient system operation is achieved only when agents are able to cooperate, but agents cannot inherently trust each other. They must therefore employ mechanisms that reduce overall efficiency to ensure that the system functions correctly. Otherwise, agents will not trust the system as a whole and will not use it.

For many distributed systems, participants can reach a *consensus* about system actions based on stored data. However, in MSCPS, consensus alone is inadequate since the system additionally interacts with the physical world. In the case of edge-cloud computing, agents need to not only agree on what kind of computational task is to be run (its requirements), but the task must also actually be executed. Ensuring this requires a method of *verification* to confirm that agreed-upon actions successfully occurred. An important consideration in addition to consensus and verification is that of *fairness*, which entails ensuring a fair outcome for involved parties, This ideally means that participants obtain the correct results, but fairness may be preserved if participants are compensated for incorrect results.

Distributed Ledger Technology (DLT) has enabled decentralized currency markets and has the potential to enable other markets as well. A *distributed ledger* (DL) is an append-only distributed database with the key feature being that the database replicas are managed by independent stakeholders. This allows mutually distrustful parties to achieve consensus

on the state of information recorded in the absence of a centralized authority because each agent is responsible for correctly updating its copy of the database. In these protocols, each participant can independently verify the validity of the transactions and determine their order.

A major criticism of proof-of-work and other related consensus protocols is that they are computationally wasteful and inefficient. The desire to harness the benefits of a decentralized consensus mechanism for an edge-computing environment, and MSCPS generally, must therefore be balanced against concerns regarding computational efficiency.

My work in this area has been primarily focused on identifying less resource-intensive ways to convince only the essential system participants of the validity of results. This is valuable when attempting to harness the surplus compute that is available at the edge. Existing solutions for harnessing this power, such as volunteer computing (e.g., BOINC), are centralized platforms in which a single organization or company can control participation and pricing. By contrast, an open market of computational resources, where resource owners and resource users trade directly with each other, could lead to greater participation and more competitive pricing. To enable trusted computations between mistrusting parties in the edge-cloud environment while minimizing the additional computation overhead, I, with collaborators developed a platform for outsourcing computations. The existing efforts to construct such a platform, particularly those using blockchain, focus on ensuring global consensus on the results of the computation, but there are many cases where this is not required.

This platform, called MODiCuM, minimizes computational overhead since it does not require global trust in mediation results. Further, none of the outsourced computations are computed using the smart contract, but instead only uses the contract to hold the participants accountable. MODICUM deters participants from misbehaving—which is a key problem in decentralized systems—by resolving disputes via dedicated mediators and by imposing enforceable fines. This effectively replaces the *trusted third party* required for

general trusted two-party computation with the distributed ledger and smart contract. Analytical results prove that MODICUM can deter misbehavior, and the overhead of MODICUM is demonstrated using experimental results based on an implementation. This work has been published in DEBS 2020.

The work comprising this chapter has been published in the Proceedings of the 14TH ACM International Conference on Distributed and Event-Based Systems (DEBS) [32].

- S. Eisele, T. Eghtesad, N. Troutman, A. Laszka, and A. Dubey, "Mechanisms for Outsourcing Computation via a Decentralized Market," in Proceedings of the 14TH ACM International Conference on Distributed and Event-Based Systems (DEBS), Montreal, Quebec, Canada, July 2020.

3.2 Introduction

The number of computing devices—and thus computational power—available at the edge is growing rapidly; this trend is projected to continue in the future [117]. Many of these are end-user or IoT devices that are often idle since they were installed for a specific purpose, which they can serve without using their full computational power. Our goal is to harness these untapped computational resources by creating an open market for outsourcing computation to idle devices. Such a market would benefit device owners since they would receive payments for computation while incurring negligible costs. To illustrate, running an AWS Lambda instance with 512MB of memory for 1-hour costs $0.03, while the electrical cost of operating a BeagleBone Black[1] single-board computer with 512MB of memory for an hour is 100 times less. Thus, it is feasible that a computation service could be provided economically.

Prior efforts to leverage these underutilized resources include *volunteer computing* projects, such as BOINC [118] and CMS@Home [119], in which users donate the computational resources of their personal devices to be used for scientific computation. Volunteer

[1] https://github.com/beagleboard/beaglebone-black/wiki/System-Reference-Manual

computing suffers from two limitations that prevent it from broader utility. First, the resources made available by volunteer computing participants are only accessible to specific users and projects. Second, it relies on systems "volunteering" their time as it does not include incentives to provide reliable access to computational resources, leading to the problem of low participation [120].

Participation can be incentivized through the implementation of a competitive market, which facilitates the discovery and allocation of supply and demand for computational resources and tasks. The market must provide mechanisms to address misbehavior and resolve any disputes[2] between the participants. Such a market could be managed by a central organization, as many in the sharing economy are (e.g., Uber, Airbnb). A central organization could mediate disputes. However, a centralized system presents a clear target for attackers, can be a single point of failure, and without competition may charge exorbitant fees. An alternative is to create an open and decentralized market, where resource owners and resource users trade directly with each other, which could lead to greater participation, more competitive pricing, and improved reliability.

In distributed computing systems, faults and misbehavior are typically addressed using consensus algorithms. Recently, distributed ledgers have emerged as a novel mechanism to provide consensus in decentralized public systems [121]. Smart contracts extend the capabilities of a distributed ledger by enabling "trustless" computation on the stored data. In theory, smart contract implementations, e.g. the one used in Ethereum [122, 31], could be used to outsource complex computations. However, since the computation is replicated on thousands of nodes, it is costly. To reduce costs, complex computations must be executed off-chain and only result aggregation, validation, and record-keeping should be kept on the chain (e.g. [34]).

Prior efforts to construct outsourced computation markets using distributed ledgers include TrueBit [123], and iExec [124]. Unfortunately, these existing solutions have varying

[2]Disputes are disagreement between the parties about the correctness of the job execution. They may arise due to a fault or malicious behavior.

degrees of inefficiency due to extensive verification of the computation performed. There have been some efforts to ameliorate this situation. For example, TrueBit performs computation using a typical computer and relies on the distributed ledger only to complete disputed instructions; however, this approach is still quite inefficient. We discuss these existing solutions and their drawbacks in detail in Section 3.3.

Contributions: The key problem in implementing a decentralized market is the *efficient* resolution of disputes, which includes determining if the results of outsourced jobs are correct. This paper introduces *Mechanisms for Outsourcing via a Decentralized Computation Market* (MODICUM), a distributed-ledger based platform for decentralized computation outsourcing. In contrast to other distributed-ledger based solutions (mentioned above), our approach retains computational efficiency by minimizing the amount of resources spent on verification through three ideas. First, it relies on *partially trusted mediators* for settling disputes instead of trying to establish a global consensus on the computation results. Second, it *verifies random subsets* of results, which keeps verification costs low while supporting a wide range of jobs. Third, it deters misbehavior through *rewards and fines*, which are enforced by a distributed-ledger-based smart contract. Thus, MODICUM does not prevent cheating and misuse, but it deters rational agents from misbehavior. The specific contributions of this paper are as follows:

1. We introduce a smart contract-based protocol and a platform architecture for incentivizing the participation of job creators, resource providers, and mediators.
2. We present an analysis of the protocol by modeling the exchange of resources as a game, and show how we can select the values of various parameters (fines, deposits, and rewards), ensuring that honest participants will not lose, and the advantage of dishonest participants is bounded.
3. We provide a proof-of-concept implementation of the protocol, built on top of Ethereum. Through comparison against AWS lambda we determine that due to the transaction costs currently associated with the main Ethereum blockchain, our implementation is cur-

rently suited only for very long-running jobs. However, any improvements to Ethereum will benefit our platform. Additionally, our protocol is not limited to the use of a specific platform. Any platform that supports smart contract functionality can be utilized.

Outline: We begin by discussing related work in Section 3.3. Then, we introduce MODICUM in Sections 3.4 and 3.5. We analyze the protocol in Section 3.6. In Section 3.7, we describe an implementation of our platform and provide experimental results on its performance. Finally, in Section 3.8, we present concluding remarks. Implementation is available at [125], and proofs and detailed specification can be found in our full paper [126].

3.3 Related Work

In [127], the authors determine that for verifying outsourced computation the cryptographic approach is not practical and should instead use repeated executions. The computations however should not be duplicated more than twice. Their strategy is simple: outsource to two providers and compare results. The key challenge then is to prevent collusion. They propose *sabotaging collusion with smart contracts*. Essentially, they hold the providers accountable using security deposits and a smart contract. They also assume that if two providers intend to collude, the providers also use a smart contract to hold each other accountable. To counter this, the authors propose a third contract that states that the first provider who betrays the other, showing the colluder's contract as proof, is granted immunity and will receive a reward. In their work, they have not yet considered the scenario when the client may be an adversary. They also do not address the case when the contractors do not need a contract to trust each other.

The authors of [128] consider a case where there is a trusted third party that would be responsible for verifying a critical computation, except that it becomes a bottleneck for the rest of the system. It instead becomes a boss and outsources the verification task. To incentivize participation, the boss offers a reward, and to discourage misbehavior the boss

requires a security deposit to enable it to enforce fines. The two verification strategies they consider are random double-checking by the boss and hiring multiple contractors to perform a job and comparing results. The authors do not consider the case when the boss is malicious or attempts to avoid paying out the rewards promised. They also do not discuss practical issues such as what if the contractors never return a result.

In iExec [129], the number of repeated executions required for verification is determined by a confidence threshold. After a result is computed, the pool scheduler checks if the results submitted achieve the desired level of confidence using Sarmenta's voting [130]. The workers register themselves to a scheduler that they choose to trust. If the scheduler breaks trust, the worker can leave to a competing pool. iExec checks tasks for non-determinism before allowing them to be deployed in the network. It does this by executing the task many times. This has the drawback that the task must take no inputs; otherwise, all execution paths would need to be tested.

In TrueBit [123], verification is provided via Verifiers, which duplicate the computation based on an incentive structure that rewards them for finding errors, and errors are intentionally injected into the system occasionally to guarantee benefits for verification. When a Verifier finds an error, it initiates a protocol where they compare the machine state at various points during the execution, like binary search. Once the step where the two solutions deviate is found, the machine state is submitted to an Ethereum smart contract, which implements a virtual machine that acts as a mediator. The TrueBit virtual machine executes that specific step and determines which agent is at fault. The authors state that though TrueBit can theoretically process arbitrarily complex tasks, in practice the mediation is inefficient for complex tasks.

In each of these prior works, the authors assume that results must be globally accepted or have universal validity. In [123], the authors mention that using a trusted mediator is an option for resolving disputes regarding whether a task was done correctly, but such a solution is unacceptable because it does not provide universal validity. We argue however

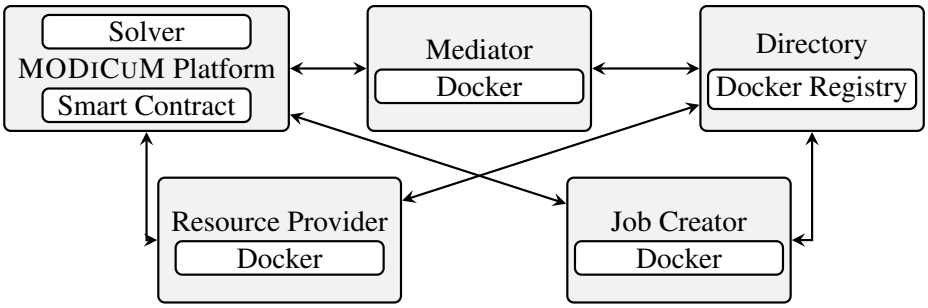


Figure 3.1: MODiCuM architecture. For ease of presentation, only one instance of Resource Provider, Job Creator, Mediator, Directory, and Solver is shown.

that many tasks—perhaps even the majority—do not require universal validity. For such tasks, only the agent who requested the execution of the task must be convinced of the validity of the result, which means that the systems proposed by prior work incur significant and unnecessary computational overhead for such tasks. Prior works also do not address the possibility that the job creator itself could manipulate the system by providing tasks that have non-deterministic or environment-dependent results.

3.4 MODiCuM Architecture

MODiCuM enables the allocation and execution of computational tasks on distributed resources that may be dishonest and may enter or exit the platform at any time. Conceptually, this requires resource management, i.e., the allocation of resources to maximize utility. We also need a service for managing job requests, which include both the job specification and the data products related to a job. Finally, we need a service that manages the market, which includes matching jobs to available resources, tracking the provenance of job products, accounting, and handling failures. Due to the decentralization of the system, *job creators* (JC) and *resource providers* (RP) can have their own strategies; however, the market must operate as a singleton.

To satisfy all of these requirements, we develop an architecture that consists of a distributed-ledger-based *Smart Contract*, storage services (*Directory*), allocation services (*Solver*), *Re-*

source Providers, *Job Creators*, and *Mediators*. These components and actors are shown in Figure 3.1 and described in the following subsections. We start by describing first what a job is in MODICUM.

3.4.1 MODICUM Jobs

A job is a computing task that takes inputs and produces outputs. The jobs MODICUM is designed to support are limited to deterministic, batch jobs that are not time-critical, and are isolated, meaning they receive no information that allows them to discern which agent (RP, M) is executing them. We discuss the importance of our limitations in Section 3.5.4. Further, the confidentiality of the job cannot be guaranteed since the computation is outsourced to any capable participant. It is essential that jobs can be executed independently of the host configuration. The state-of-the-art approach for achieving this is to use a container technology [131]. Consequently, we use Docker [132] images to package the jobs in our implementation. Docker containers provide an easy-to-use way of measuring and limiting resource consumption and securely running a process by separating the job from the underlying infrastructure. To avoid downloading a full image for every job execution, we support Docker layers. To execute an instance of a job, a base layer (i.e., OS or framework), an execution layer with job-specific code, and a data layer are required. The base layer can be downloaded a priori, and the other layers are downloaded after a match. If the execution layer has already been downloaded, then only the new data layer is required. The job requirements, including the required resources, are specified in the offers that the agents make. These requirements are used by the resource provider to set the Docker container's resource constraints at runtime.

The cost of running a job depends on the amount of resources used. The resources we consider for feasibility are instruction count (computes from CPU speed and time on CPU), disk storage, memory, and bandwidth for downloading the job. The resources we use to compute the price are the instruction count and bandwidth used as reported by the RP

where each is multiplied by the RP's asking price. Thus, the price of the job is calculated as follows:

$$\begin{aligned}\pi_c = &result.instructionCount \cdot resourceOffer.instructionPrice \\ &+ result.bandwidthUsage \cdot resourceOffer.bandwidthPrice\end{aligned} \tag{3.1}$$

Note that it is often difficult to estimate resource requirements precisely. Repeated executions of the same job result in approximately the same resource usage, but not exactly the same. Therefore, when constructing an offer, the JC must add sufficient margins to the estimated resource requirements to account for the expected variance. If the JC did not provide this leeway, jobs would often exceed their resource allotments, forcing the JC to often pay for the execution of failed jobs.

3.4.2 MODICUM Actors

3.4.2.1 Resource Provider and Job Creator

Job Creators (JC) have jobs to outsource and are willing to pay for computational resources. *Resource Providers* (RP) have available resources and are willing to let Job Creators use their hardware and electricity in exchange for monetary compensation. JCs post offers to the market specifying the jobs, quantities of resources required, deadlines for execution, required Docker base layers, computation architecture, and unit bid prices for resources; while RPs post offers specifying available resources, Docker base layers, computation architecture, and unit ask prices for resources. To prevent the JC and RP from cheating, they are required to include security deposits (see Eq. (3.2)) in the offers submitted to the platform. We provide more details on the costs and deposits later in Section 3.6. JCs and RPs also specify trusted Directories and Mediators, which we describe below.

A JC has multiple strategies for verifying the correctness of the results returned by an

RP. Any reasonable verification strategy must cost less than what it would cost to simply execute the job. A straightforward verification strategy is to re-execute a random subset of jobs and compare the results with those provided by the RP. As long as the number of verified jobs is low enough, this strategy is viable. If the JC cannot execute the job itself, then it can post duplicate job offers instead and compare the results provided by different RPs. For some jobs, there exist verification algorithms that cost significantly less than execution, and so the JC may be able to verify every job. However, this requires the JC to implement an efficient algorithm for verification, which might be challenging.

3.4.2.2 Directory

Directories are network storage services that are available to both JCs and RPs for transferring jobs and job results. Directories are partially trusted by the actors, which means that actors (RPs and JCs) choose to trust certain Directories, but these are not necessarily trusted by the platform or by other RPs or JCs. Directories are paid by the JC and RP for making its services available for the duration of a job.

3.4.2.3 Solver

Matching a JC offer, and an RP offer requires computations that cannot be executed on a smart contract which has limited computation capabilities. Therefore, we extend the concept of hybrid solvers introduced in [34] and include *Solvers* in MODICUM. A Solver can be a standalone service, or it may be implemented by another actor (e.g., RPs and JCs may act as Solvers for their own offers). Unlike the smart contract, Solvers are not running on a trustworthy platform; hence, the contract has to check the feasibility of matches that the Solvers provide which is significantly easier computationally than finding matches. Solvers receive a fixed payment, set by the platform and paid by the JC and RP, for finding a match that is accepted by the platform.

3.4.2.4 Mediator

When there is a disagreement between a JC and an RP on the correctness of a job description or a job result (e.g., the RP claims that a job result is correct, while the JC claims that it is incorrect), a partially trusted *Mediator* decides who is at fault or "cheating." A Mediator is capable of executing the job in the same way as the RP, but it can be more expensive since it is expected to provide a more reliable service and to maintain its reputation in the ecosystem. Each Mediator sets a price which it is paid by the JC and RP for making its services available for the duration of a job. In the case of mediation, it is additionally compensated for the computations it executes.

3.4.2.5 Smart Contract

The *Smart Contract* (SC) is the cornerstone of our framework. Most communication in MODICUM is effectuated through function calls to the SC and events emitted [3] by the SC. The SC is deployed and executed on a trustworthy decentralized platform, like the Ethereum blockchain [31], which enables it to enforce the rules of the MODICUM protocol described in the next section. It also enables actors to make financial deposits and to withdraw funds on conditions set by the SC. The functions provided by the smart contract can be found in our full paper [126].

3.5 MODICUM Protocol

In this section, we discuss the operation protocol, possible misbehavior, the concept of mediation, and various faults that can occur and how they can be handled. Fig. 3.2 shows a possible activity sequence from registration to completion of a job. Fig. 3.3 represents the state of a job from the perspective of the smart contract.

[3] *Emitted* events in platforms such as Ethereum are recorded to the transaction logs of the ledger, which can be accessed by interested agents via polling. We use the word *emit* because that is word used for this functionality in Solidity, which we use for our proof-of-concept implementation.

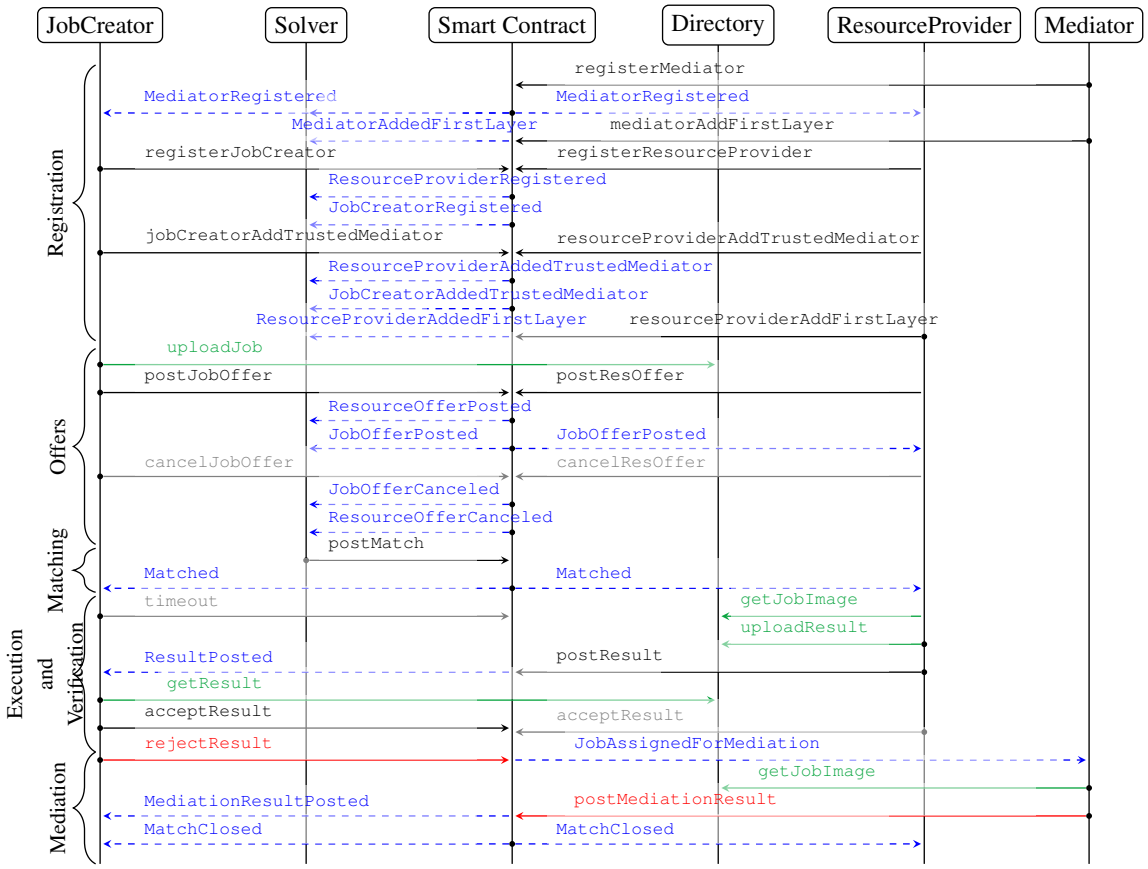


Figure 3.2: Sequence diagram showing the outsourcing of a single job. Black arrows are function calls to the smart contract. Blue dashed lines are events emitted by the smart contract. Gray lines are optional function calls. Red lines are optional calls that are required in case of disagreement between RP and JC. Green lines are off-chain communication. Note that events are broadcast, and visible to all agents that can interact with the contract.

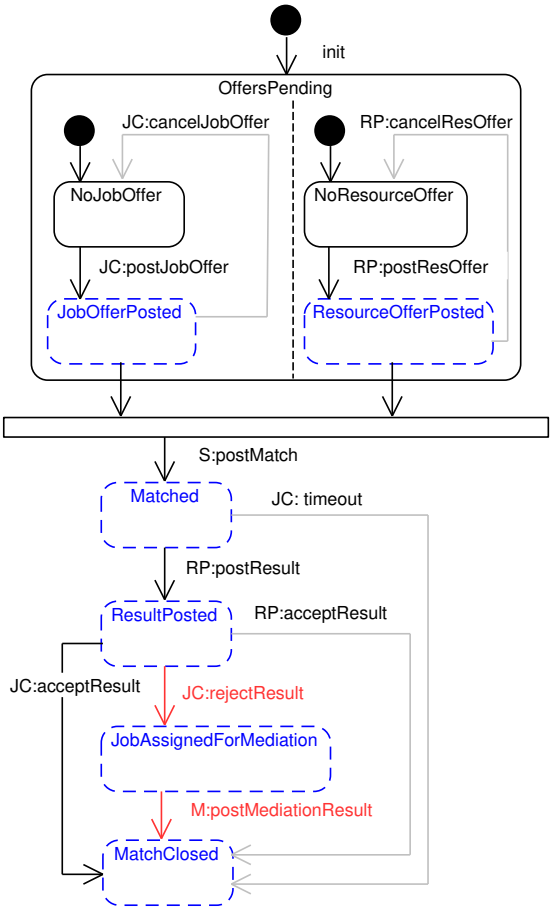


Figure 3.3: States of job in MODICUM. Function calls to the smart contract are prefixed by the actors that make the calls.

3.5.1 Registration and Posting Offers

First, RPs and JCs register themselves with MODICUM (Fig. 3.2: `registerResourceProvider`, `registerJobCreator`). Note that RPs and JCs need to register only once, and then they can make any number of offers. If an RP is interested in accepting jobs, it will send a resource offer to the platform (Figs. 3.2 and 3.3: `postResOffer`). On the other side, a JC first creates a job and uploads the job to a Directory that it trusts and can use (Fig. 3.2: `uploadJob`). Then, it posts a job offer (Figs. 3.2 and 3.3: `postJobOffer`).

Note that any time before the offers are matched, RPs and JCs can cancel their offers. Cancellation can be due to unscheduled maintenance, or because their offers have not been matched for a long time and they wish to adjust their offer (e.g., increase maximum price) by cancelling the previous offer (Figs. 3.2 and 3.3: `cancelJobOffer` and `cancelResOffer`) and posting a new one. Once matched, cancellation is no longer permitted.

3.5.2 Matching Offers

After receiving offers, the smart contract notifies the Solvers by emitting events (Figs. 3.2 and 3.3: `JobOfferPosted`, `ResourceOfferPosted`). The Solvers add these to their list of unmatched offers and attempt to find a match among them. After finding a match (which consists of the resource offer, job offer, and the Mediator) a Solver posts it to the smart contract (Figs. 3.2 and 3.3: `postMatch`). The smart contract checks that the submitted match is feasible, if it is the match is recorded and the JC and RP are notified that their offers have been matched (Figs. 3.2 and 3.3: `Matched`). Note that for a match to be feasible the resources specified in the RP offer must satisfy the requirements of the job specified by the JC offer. Additionally, they should have a common architecture, trusted mediator, and directory. The full feasibility specification can be found in our full paper [126]. With the `Matched` event, the contract pays the Solver the amount RP and JC

specified as the matching incentive.

3.5.3 Execution by RP and Result Verification by the JC

After receiving notification of a match, the RP downloads the job from the Directory (Fig. 3.2: `getJobImage`) and runs the job. While running the job, RP measures resource usage. Finally, when the job is done, it uploads results to the Directory (Fig. 3.2: `uploadResult`) and reports the status of the job and resource usage measurements to the smart contract (Figs. 3.2 and 3.3: `postResult`). The status of the job is the state in which the job execution is terminated. Some possible termination states include `Completed` or `MemoryExceeded`. The full list of status codes can be found in our full paper [126] and we will discuss some of them in Section 3.5.4.

After receiving the notification that the result has been posted (Figs. 3.2 and 3.3: `ResultPosted`) the JC downloads the result (Fig. 3.2: `getResult`) and decides whether to verify it or not. It then accepts, ignores, or rejects the result. If the JC accepts the result (Figs. 3.2 and 3.3: `acceptResult`) the contract returns deposits, pays the RP and Mediator, and closes the match (Fig. 3.3: `MatchClosed`).

If the JC ignores the result, then after some time, the window for the JC to react closes, and the RP is permitted to accept the result (Fig. 3.3: `RP:acceptResult`) resulting in the match closing (Fig. 3.3: `MatchClosed`). If the JC disagrees with the result, it rejects the result (Figs. 3.2 and 3.3: `rejectResult`) with a reason code such as `WrongResults` or `ResultNotFound`; then, mediation follows (Figs. 3.2 and 3.3: `JobAssignedForMediation`).

3.5.4 Faults and Mediation

There are essentially two types of faults that can occur in the system: 1. Connectivity: this can occur for the JC, RP, and the Mediator when they try to communicate with the Directory or the smart contract. Note that the JC, RP, and Mediator do not talk directly

(see Fig. 3.1). Solver connectivity faults are not a concern since it only interacts with the smart contract and the failure of a Solver implies that Solver does not submit a solution, but others may. Hence, we only worry about the connectivity to the smart contract and the directory. 2. Data (job input and results of execution): it can be malformed, return an exception when executed, not be available on the Directory, be verifiably incorrect, etc.

First, we discuss the connectivity with the smart contract. If the RP cannot communicate with the smart contract, it may not receive a notification that it has been matched and it is also unable to call `postResult`. These are both addressed by the JC having a timeout. If the JC calls `timeout` (see Figs. 3.2 and 3.3) and the required waiting period has elapsed, then the smart contract pays the JC the estimated value of the job from the RPs deposit and returns the remainder. The timeout also addresses when the JC misses the `ResultPosted` message sent by the RP, since if the misses the message, it will attempt to timeout, which will fail because the result was posted, and then it will fetch the result. Smart contract connectivity failure also means that the JC cannot send `acceptResult` and that the RP never receives a notification that the JC accepted. This is addressed by allowing the RP to bypass the JC and call `acceptResult` if the platform specified duration for the JC to respond has elapsed. Connectivity failure also means that the JC and the RP may not receive the `MediationResultPosted` message. In this case, they may respond by removing that Mediator from their trusted list; and if the result eventually arrives, they can re-add the Mediator. This also addresses when the Mediator does not get the mediation request or cannot post the mediation result. In this case, mediation is considered to have failed; to release the security deposits, we enforce a timeout via the smart contract. In the event of this timeout, we instead pay the RP half of the JC's job estimate and return the remainder of the deposits. Obviously, the Mediator does not get paid since mediation failed.

Now, we discuss connectivity with the Directory. If a JC cannot connect to the Directory to submit a job, it simply tries to upload to another Directory it trusts. If it cannot

connect to retrieve a result then the JC must either pay the RP or request mediation with the `DirectoryUnavailable` status. To verify the JC claim, the Mediator queries the Directory for its uptime. If the Directory reports that it was available for the entire job duration, then the Mediator assigns fault to the JC; otherwise, it assigns fault to the Directory. If the JC does not agree, it may remove the Directory and/or the Mediator from its trusted lists. Similarly, if the RP cannot connect to the Directory, either to fetch the job or upload the results, then it posts a result with the `DirectoryUnavailable` status. If the JC does not agree that the Directory is unavailable, then it requests mediation which proceeds as above with the RP rather than the JC. If the Mediator sends a mediation result of `DirectoryUnavailable` then the RP and JC may choose to remove the Directory, Mediator, or both from their trusted list.

Finally, we discuss the data faults. The data faults that the RP can detect and the corresponding result status are: 1) no job on the Directory (`JobNotFound`), 2) a job description error (`JobDescriptionError`), 3) excessive resource consumption (`ResourceExceeded`), and 4) an execution exception during execution (`ExceptionOccurred`). The JC can request mediation if it disagrees with a claim of any of these faults, in addition to detecting no result on the Directory, or that the result is incorrect. Finally, the JC can request mediation if, after verification, the JC suspects that the resource usage claimed by the RP is too high. This could occur since the resource variation should be small, but the JC may have set a high resource limit to ensure the job completes.

If the JC requests mediation claiming there is a data fault, then the Mediator attempts to replicate the steps taken by the RP, with the distinction being that it re-executes the job n times (see Section 3.6.1, it is defined as a parameter of the smart contract), and compares its results with the RP's results. In two cases, the JC will be at fault: 1. All of the Mediator's results and RP's results are the same, which means that the RP has executed the job correctly. 2. The Mediator gets two different results when running the job, which means that JC has submitted a non-deterministic job. Otherwise, the Mediator assumes that

the RP has submitted a wrong result. Another case is when the JC claims that the result is not on the Directory. In that instance, the Mediator attempts to retrieve the result from the Directory. If it cannot it faults the RP, if it can it faults the JC. If either agent disagrees it may remove that Directory, Mediator, or both from its trusted list.

The Mediator submits the verdict to the smart contract (Figs. 3.2 and 3.3: `postMediationResult`), and the smart contract claims the security deposit. Of the deposit, *the actual job price* is used to compensate the damaged party for its losses, and π_m (which is the job price times the number of repeated executions n) goes to the Mediator to cover its mediation costs. In addition, Mediators always receive π_a as payment for making their service available. They receive this when a job is closed (Figs. 3.2 and 3.3: `MatchClosed`).

For this mediation approach to work, the RP must not allow jobs to access any extra information (e.g., physical location, time) beyond what is in its description and Docker image. Otherwise, a JC could create a job that could determine where it is running (e.g., via connecting to a remote server) and produce different results on the RP than on the Mediator. Thus, the Mediator would always incorrectly blame and punish the RP. This is why the platform requires that jobs be 1. deterministic and punishes the JC if non-determinism is detected (otherwise, we could not use repeated executions to verify) and 2. batch (otherwise, the jobs could not be isolated).

3.5.4.1 Collusion

A part of the challenge in designing a fair system is the problem of collusion. We enumerate all possible two-party collusions and discuss their objectives and how MODICUM addresses them. We do not consider more than two-party collusions explicitly because they are indistinguishable from two-party collusions for the non-colluding agents in MODICUM.

- *Job Creator and Solver*: Since offers are public and any participant can act as a solver, the collusion between JCs and solvers is inevitable. The goal of this collusion is to match a

JC's offers to the resource offers with the lowest unit price. Thus, the RP with the lowest-price offer will be matched first. This is harmless because every resource offer includes a minimum reservation price; thus, a JC cannot force an RP to perform computation for less than what the RP voluntarily accepts.

- *Job Creator and Mediator*: Both JC and Mediator can benefit from this collusion by taking the RP's security deposit and splitting it between them, while the JC can also benefit by having its jobs executed without paying. This can be achieved by the JC requesting mediation on a correct result and the Mediator ruling in favor of the JC. To an honest RP, this collusion will appear as a faulty Directory, faulty Mediator, or non-deterministic job, though it can eliminate the last by repeating the job execution. The RP removes the Mediator and Directory from its trusted list. Thus, a mediator can launch this attack only once per RP.
- *Job Creator and Directory*: This collusion is similar to the one between the JC and Mediator. JC and Directory can collude in multiple ways. For example, the Directory manipulates the job so that the RP will return `JobDescriptionError` result status. Then, JC will request mediation, the Directory will provide the correct job to the Mediator, and the Mediator will rule in favor of JC. Since the RP cannot distinguish this collusion from the one between a JC and mediator, as a response, the RP removes the Mediator and Directory from its trusted list. Thus, a Directory can launch this attack only once per RP.
- *Job Creator and Resource Provider*: There is no possible benefit from this collusion.
- *Resource Provider and Solver*: The goal of this collusion is the same as for the collusion between JC and Solver, and its impact and mitigation are also the same. This collusion is desirable for the same reason as well.
- *Resource Provider and Mediator:* RP and Mediator can both benefit from this collusion by taking the JC's security deposit and splitting it between them; while the RP can also benefit by receiving payment for a job without actually executing it. This can be achieved

by the RP returning any job result, which the JC might verify and request to be mediated, upon which Mediator will rule in favor of the RP. This collusion is mitigated in the same way as the collusion between JC and Mediator, except that the roles of JC and RP are reversed.

- *Resource Provider and Directory*: The goal of this collusion is the same as the RP and Mediator collusion, and it can be achieved and handled in a manner similar to the JC and Directory collusion. To an honest JC, collusion will appear as a faulty or colluding Directory. As a response, the JC removes the Mediator and Directory from its trusted list. Thus, a Directory can launch this attack only once per JC.
- *Directory and Solver*: There is no benefit from this collusion.
- *Directory and Mediator*: This collusion aims to ensure that the JC will request Mediation by manipulating data or availability, and then splitting the payment for Mediation. Depending on the Directory's manipulation and the Mediator's ruling, either the JC or RP will respond in the same way as if the RP or JC were colluding with the Directory or Mediator.
- *Mediator and Solver*: The goal in this case is for the solver to prioritize trades that include the Mediator, and further prioritize JCs and RPs with a history of requiring mediation. This could result in unfairness, i.e., some jobs may never get matched. However, this is not a real concern since the JC and RP can act as solvers and match their own offers.

From exploring the possible scenarios, we conclude that the JC and RP could be cheated once by a Mediator or a Directory; however, the faulty agent will be removed from the trusted list afterward. Since the business model for Mediators and Directories is attracting RPs and JCs who trust and pay them for their services, they have strong incentives to build a positive reputation in the ecosystem. Thus, we assume they will behave honestly. The formal proof for this conjecture will be provided in future work.

3.6 Analyzing Participant Behavior and Utilities

Here we formulate the agents' actions as a game and solve for strategies that result in a Nash equilibrium. We also show that platform parameters can be set so that a rational JC will follow the protocol with, at least, some minimum probability. The extensive-form representation of the game (shown in Fig. 3.4) is explained below.

3.6.1 Game-Theoretic Model

To understand the game, we first introduce a set of parameters in Table 3.1. Parameters denoted by p are probabilities, π are payouts, c are costs incurred by agents, and g are the costs for interacting with the platform. Many parameters have constraints on the valid values that they can take and on their relationships with other parameters.

Now, we consider the JC's choices. The JC has a job to outsource, whose execution provides a constant benefit b upon receiving the correct job result. The JC is willing to pay a job-specific price, up to $\hat{\pi}_c$, which is appropriate for the resources required to have the result computed. The JC is able to verify if the job result is correct, but it incurs verification cost c_v by doing so. In order to mitigate this cost, the JC may choose to verify the result with some probability p_v, trading confidence for lower costs.

However, a dishonest JC can design non-deterministic jobs and we assume that the JC can always recover the correct result from any output[4]. The goal of the JC in designing a non-deterministic job is to get the correct output without having to pay the RP. It can accomplish this if it requests mediation, and when it does, the mediator concludes that the result returned by the RP is incorrect. Thus, if a JC designs a job to look "normal" to the mediator with probability p_a and "incorrect" with probability $1 - p_a$, then the JC will accept correct results with probability p_a and request mediation with probability $1 - p_a$. A simple illustration of such a job is one which returns a natural number as its solution, and

[4]Note that the RP and M cannot be expected to analyze the code, and hence cannot know that the job contains non-determinism.

changes the sign of that value (i.e., multiply by -1) with a fixed probability, creating a set of positive results and a set of negative results.

The game starts with the JC choosing a probability value for p_a. A probability of $p_a = 0$ means that the JC is completely dishonest, and all results will be considered incorrect. A probability of $p_a = 1$, means the JC is honest and all correct outputs are accepted.

The RP makes the next move, choosing between honestly executing the job or forging a result to deceive the JC. The RP may be motivated to return a false result if the job execution cost c_e is higher than the deception cost c_d and the JC does not verify the result with some probability. The RP executes the job with probability p_e, where $p_e = 0$ means that the RP is completely dishonest and always attempts to deceive the JC, while $p_e = 1$ means that the RP is honest and executes the job correctly. Note that the correct result can be a fault code if the computation fails.

The JC makes the next move and selects its strategy, choosing between verifying the result or passing on the verification. The JC verifies the with probability p_v. If verification finds the result to be incorrect, the JC requests mediation to dispute the result.

To resolve the dispute, the Mediator must determine which agent is at fault. The Mediator does this by performing the steps that an RP would take to execute a job, repeating several times to detect non-determinism. When initialized the smart contract specifies a *verification count n*, which is the number of times the Mediator will execute a job checking for anomalies. Since the job has a probability p_a of returning a normal result and the Mediator executes the job n times, the probability that the job returns a normal result in every execution is p_a^n. Thus, the Mediator detects a non-deterministic job with probability $1 - p_a^n$ and fines the JC for being dishonest.

As stated in Section 3.5.1, to deter cheating through fines, we require JCs and RPs to provide a *security deposit d* when submitting offers. We define the deposit to be dependent on the JC's estimate of the job price $\hat{\pi}_c$ and scaled by a penalty rate θ, which is set by the smart contract. The job price $\hat{\pi}_c$ estimated by the JC is the same as π_c except it uses the JC's

bid prices and requested resources. The penalty rate must be set to a sufficiently high value to deter misbehavior. The security deposit must also cover the cost of potential mediation π_m, which we estimate as $\hat{\pi}_c \cdot n$ since the JC is willing to pay $\hat{\pi}_c$ and the Mediator must run n times. The deposit must also cover the availability costs of the Mediator and Directory as well as the Solver costs; we let π_a denote the sum of these costs. Thus, we define the minimum security deposit d_{min} as:

$$d_{min} = \underbrace{\hat{\pi}_c \cdot \theta}_{\text{penalty}} + \underbrace{(\hat{\pi}_c \cdot n)}_{\pi_m} + \pi_a \tag{3.2}$$

The game induced by the interactions of the actors described above has 7 possible outcomes. Each outcome has payouts for the agents as described in Fig. 3.5. To illustrate how the payouts are calculated, consider the following sequence. The JC pays g_j to submit a non-deterministic job with a probability p_a of returning a normal result. The RP honestly executes the job incurring cost c_e and pays g_r to submit the result. Since the RP executed honestly, the JC receives the benefit b. The JC verifies the result, incurring cost c_v, and detects that the non-beneficial part of the result is anomalous. It then attempts to avoid paying π_c to the RP by requesting mediation, paying g_m. The Mediator executes the job n times and if in one or more of those executions it encounters an anomalous result, which occurs with probability $1 - p_a^n$, then it submits to the smart contract that the JC is at fault and the JC loses its security deposit d for submitting non-deterministic jobs resulting in outcome o_6. Otherwise, if all of the results from the repeated executions are normal then the JC successfully cheats and receives π_d as reparations for being "faulted" resulting in o_7. The payouts of the other outcomes are calculated similarly. The platform interaction costs are fixed properties of a given smart contract and its underlying platform.

Since we assume that Directories and Mediators always act honestly, we do not consider their strategic decisions in our game analysis. The expected utilities of both the RP and the JC are summarized in Tables 3.2 and 3.3, respectively. The table is constructed by

considering the possible actions of the RP and JC. There are two possible actions for RP (execute and deceive) and two for JC (verify and pass) as illustrated by the tree in Fig. 3.4. Hence, the utilities of RP and JC depend on the four action combinations and their possible outcomes $o_1 \cdots o_7$. To understand how the utilities are calculated, consider the example of the case when RP chooses to execute, and JC chooses to verify. This is node 7 in Fig. 3.4, and there are three possible outcomes o_5, o_6, o_7. Outcome o_5 occurs with probability p_a, o_6 with probability $(1-p_a)(1-p_a^n)$, and o_7 with probability $(1-p_a)p_a^n$. Thus,

$$U_{EV}^{JC} = p_a \cdot t_{o_5} + (1-p_a)\big((1-p_a^n)\cdot t_{o_6} + p_a^n \cdot t_{o_7}\big) \tag{3.3}$$

The utility for each combination of actions is denoted by utility U with a superscript of the agent (i.e., RP and JC) and subscript of the action combination of RP and JC (i.e., EV is $< execute, verify >$, DP is $< deceive, pass >$, EP is $< execute, pass >$, and DV is $< deceive, verify >$). Note that we replace the total outcomes payoffs using Fig. 3.5 in Tables 3.2 and 3.3.

3.6.2 Equilibrium Analysis

Here we analyze the utility functions explained in the previous section. For lack of space, we describe only the key results. Detailed proofs for these statements can be found in the appendices of the arXiv version of this paper [126].

The ideal operating conditions for the platform would be if the RP always *executed* ($p_e = 1$), the JC never had to *verify* ($p_v = 0$) and only submitted jobs which returned deterministic results ($p_a = 1$). However, if the agents are rational, these parameters do not constitute a Nash equilibrium. This is because if the JC does not verify, then the RP will choose to deceive rather than execute since $U_{EP}^{RP} < U_{DP}^{RP}$. The JCs utility in this case is always negative and so it is better off not participating in the platform. This proves the following theorem:

Theorem 1 (JC should not always pass)**.** *If the JC always* passes *(i.e., $p_v = 0$), then the RP's best response is to always* deceive *(i.e., $p_e = 0$).*

If the RP always chose to deceive, the platform would serve no purpose. Therefore, we must ensure that if the JC chooses to verify, the RP prefers to execute. This occurs when $U_{DV}^{RP} < U_{EV}^{RP}$. We show in [126] that this is true if $p_a^{n+1} > \frac{1}{2}$. When these conditions are true, we can prove the following theorem:

Theorem 2 ($p_e > 0$)**.** *If $p_v > 0$ and $p_a^{n+1} > \frac{1}{2}$, then a rational RP must execute the jobs with non-zero probability.*

Recall p_a is a parameter set by the JC, so to satisfy the condition on p_a in Theorem 2 we must show that the platform can set parameters to force the JC to choose a value for p_a that is greater than some lower bound. We assume that the JC is rational and chooses p_a to optimize its utility U^{JC}. To find the bound we take its derivative with respect to p_a, $\frac{\partial U^{JC}}{\partial p_a}$, and assess how each parameter shifts the optimal value for p_a. The trends are as n, θ, g_m increase p_a also increases, meanwhile as π_c and p_e increase p_a decreases. Knowing how varying each parameter shifts the optimal value for p_a we can select the worst-case values for each parameter, i.e. those that minimize optimal p_a, maximizing dishonesty. Specifically, if the parameter and p_a are inversely related, set the parameter to its maximum allowed value, and if they are directly related set the parameter to its minimum value. Thus, the worst-case values for each of the parameters are: $g_m = 0$, $p_e = 1$, $n = 1$, $\theta = 0$. The plot in Fig. 3.6 uses those parameters and shows that increasing n does cause the optimal value for p_a to increase. We see that when $n = 1$, the optimal $p_a = 0.5$; and when $n = 4$, the optimal $p_a = 0.943$. This can be summarized as:

Theorem 3 (Bounded p_a)**.** *Setting the JC utility function parameters to minimize the optimal value for p_a (maximizing a rational JC's dishonesty) ensures that any deviation will increase p_a. The platform controls n and θ, and so controls the minimum optimal value of p_a.*

We want to minimize the number of times the Mediator has to replicate the computation, so we set $n = 2$ and set $\theta = 50$ which yields a minimum $p_a = 0.99$.

So far we have shown that we can ensure that a rational RP will prefer to execute when a JC verifies, and deceive when the JC passes, and we can limit the amount of cheating the JC can achieve through non-deterministic jobs. Next, we analyze the JC utilities to determine the Nash equilibria of the system.

Analyzing JC types: The preferences of the JC depend on the parameters in its utility function. We refer to each combination of preferences as a "type" of JC. We will call a JC that prefers to always verify as type 1. A JC that prefers to always pass is type 4; we have already covered this type and determined that a JC of this type will not participate. A JC that prefers to verify when the RP executes and pass when the RP deceives is type 2. A JC that prefers to pass when the RP executes, and verify when the RP deceives, is type 3. The JC has a preference because its utility is better in that case. These preferences are summarized in Table 3.4 with the $*$ symbol followed by the type that prefers that choice. This table is Table 3.3 refactored to remove terms that do not impact the JC's preference and to highlight the relationship between the preference and the cost of verification c_v. We consider the equilibrium for each type assuming the RP has been restricted as we discussed previously. The theorems below summarize these observations. Proofs are available in our full paper [126].

Theorem 4 (JC type 1)**.** *If the JC is type 1, it will always verify ($p_v = 1$) since c_v is sufficiently low. This results in a pure strategy equilibrium $< execute, verify >$.*

Theorem 5 (JC type 2)**.** *If the JC is type 2, it results in two pure strategy equilibria $< execute, verify >$, $< deceive, pass >$, and one mixed strategy Nash equilibrium where the JC randomly mixes between verifying and passing.*

Theorem 6 (JC type 3)**.** *If the JC is type 3, it will result in a Nash equilibrium where the JC randomly mixes between verifying and passing, and the RP mixes between executing and deceiving.*

Strategies: Based on these preferences, a type 1 JC will always verify $p_v = 1$. Type 2 JCs may also choose to always verify, or choose a mixed strategy, setting p_v such that the RP receives the same utility regardless of whether it executes or deceives resulting in a Nash equilibrium. It achieves this by solving Eq. (3.4) for p_v, setting $c_e = \pi_c$ and $c_d = 0$. Type 3 JCs only have the option of solving Eq. (3.4) for p_v.

$$\begin{gathered} p_v \cdot U_{EV}^{RP} + (1-p_v) \cdot U_{EP}^{RP} = p_v \cdot U_{DV}^{RP} + (1-p_v) \cdot U_{DP}^{RP} \\ \text{Solve for } p_v; \quad p_v = \frac{c_e - c_d}{p_a^{n+1} \pi_c (\theta + n + 1)} \end{gathered} \tag{3.4}$$

The RP's strategy changes depending on which type of JC it is working with. If the JC is type 1, it is simple: the RP must execute. However, the other two types can mix, so the RP must also mix. It does this by solving Eq. (3.5) for p_e. The challenge with this is that the RP does not know the value of c_v. However, all other parameters are known once a match is made except p_a which from our work earlier we know that $p_a \geq .99$. Thus, the RP can sample c_v from a uniform distribution where $0 \leq c_v \leq P_{EV}^{JC}$ for type 2 and $0 \leq c_v \leq P_{DV}^{JC}$ for type 3 (P_{EV}^{JC} is the JCs preference value for $< execute, verify >$ from Table 3.4). However, since the RP does not know which type of JC it is working with, it further mixes between the 3 strategies according to its belief on the distribution of the types of JC in the system.

$$\begin{gathered} p_e \cdot U_{EV}^{JC} + (1-p_e) \cdot U_{DV}^{JC} = p_e \cdot U_{EP}^{JC} + (1-p_e) \cdot U_{DP}^{JC} \\ \text{Solve for } p_e; \quad p_e = \frac{2\pi_c - c_v - g_m - \pi_c (1-p_a^n)(n+\theta+1)}{p_a (2\pi_c - g_m - \pi_c (1-p_a^n)(n+\theta+1))} \end{gathered} \tag{3.5}$$

This analysis shows that we can limit the dishonesty of the JC and that the JC and RP can compute strategies that will result in a mixed-strategy Nash equilibrium. Further, we can show that the JC will not have to verify frequently by examining Eq. (3.4) and showing that the maximum verification rate is low. First, we know that if $c_e < \pi_c$ (which should hold since otherwise the RP will always deceive), then $\frac{c_e - c_d}{\pi_c} \leq 1$. In this case, the

verification rate is at its maximum when $c_e = \pi_c$ and $c_d = 0$. Simplifying Eq. (3.4), we find that $\frac{1}{p_a^{n+1}(\theta+n+1)}$. In establishing Theorem 3, we showed that setting θ and n enforces a minimum value for p_a. Again to minimize computation replication we choose $n = 2$ and $\theta = 50$ and recall that this results in $p_a \geq 0.99$. Substituting these values into our simplified equation and find that $p_v = 0.02$. This means that the JC will verify 2% of the results. Since $p_a \geq 0.99$, mediation will occur at most 0.02% of the time.

3.7 MODICUM Implementation

In this section, we describe an implementation of MODICUM. The code is available on GitHub [125]. We use a private Ethereum network to provide smart contract functionality. MODICUM actors, including Job Creators, Resource Providers, Solvers, and Directory services are implemented as Python services. The matching solver uses a greedy approach to match offers as they become available using a maximum bipartite matching algorithm. These actors use the JSON-RPC interface to connect to the Ethereum Geth client [133]. Note that each actor can be configured with any number of Geth clients, and the ledger can be implemented either as a private blockchain or we can use the main Ethereum chain as the ledger. We use Docker [132] images to package jobs. Jobs can be run securely by separating the job from the underlying infrastructure through isolation. This can be achieved using a hardening solution such as AppArmor [134] or seccomp [135] in conjunction with Docker. This protects computational nodes from erroneous or malicious jobs if properly configured. Proper configuration has been discussed in other papers, for example [136], and we do not go into detail here. To determine job requirements, jobs are profiled using cAdvisor [137].

As part of the offer and matching specification, we require the JC to include the hash of the base of the Docker image. Additionally, during setup, Mediators and RPs specify a set of supported base Docker images (Fig. 3.2: *mediatorAddFirstLayer* and *resourceProviderAddFirstLayer*). This permits some optimization since RPs are able to specify which base

images they have installed, thus by matching them accordingly, we can reduce the bandwidth required to transfer the job by the size of the base image. Common base images vary between about 2MB - 200MB [138].

3.7.1 Experimental Evaluation

JCs, RPs, and a Mediator were deployed on a 32 node BeagleBone cluster with Ubuntu 18.04. We set up a private Ethereum network. The Solver and Directory were deployed on an Intel i7 laptop with 24GB RAM. The actors connect to the Geth client [133] each using a unique Ethereum account.

Measuring Gas Costs and Function Times: To measure the minimum cost of executing a job via MODICUM, we had a single JC submit 100 jobs and measured the function gas costs and call times independent of the job that was being executed. These can be found in our full paper [126]. The JC's average gas cost of a nominal execution is 592,000 gas. At current Ethereum prices, this converts to $0.168 per job for the JC [139]. Comparing this to Amazon Lambda pricing [140] on a machine with 512MB RAM (which a BeagleBone has), a job would have to last ˜6 hours to incur the same cost. However, the electrical costs to run a BeagleBone (210-460mA @5V) at maximum load for that long, assuming $0.12/kWh electricity price, is only $0.0016. This illustrates that there is potential for such a transaction system to be a viable option compared to AWS. However, using Ethereum as the underlying mechanism is currently only viable for long-running jobs; but work is underway to improve the efficiency of Ethereum [141]. We also measured the gas cost of a mediated execution: the JC's average gas cost in this case is 991,000, and the Mediator's cost to post the mediation result is 187,000.

During these tests, the duration of the function calls was also measured (see Fig. 3.8). We note that the mean time for a block with transactions to be mined (block time in Fig. 3.8) is about every 10 seconds, and that function call delay is consistently about 5 to 10 seconds longer. This may be attributable to calls missing a recent block. The close time is the

time measured between `MatchClosed` events. Since the jobs were run sequentially it is a measure of the cumulative time added to the execution of a job, in this test running a job through MODICUM added approximately 52 seconds.

Measuring the Overhead of Platform: To measure the overhead, we compared the execution of jobs run with Docker containers natively against jobs run in MODICUM. The job we used was the bodytrack computer vision application drawn from the PARSEC benchmarking suite [142]. PARSEC has been used to benchmark resource allocation platforms [143] as well as platforms for high-performance computing [144] among others. We again ran 100 jobs, which took a total of 221 minutes, averaging 2 minutes per job. The mean time for a block to be mined was 31 seconds, meaning it took about 4 blocks to complete a job. The block time was likely longer in this experiment because as the blockchain grew block mining times appeared to increase, though we did not study this explicitly. This application tracks the 3D pose of a human body through a sequence of images. In Figure 3.9a, we see that MODICUM increases the runtime by about 1 second or 4%. In Figure 3.9b, the average memory consumption increases by 0.1MB, or 3%. To check mediation, we ran the jobs again, but rejected the results for all 100 jobs and requested mediation.

Resource consumption while running the benchmark on MODICUM can be seen in Fig. 3.7. The nodes are idle during the valley from 16:04:30 to 16:05:30 at which point the platform is started. From 16:05:30 onward, MODICUM is running, and the bursts that can be seen, for example at 14:09:30-14:10:45 in Figure 3.7c, are when jobs are being executed. MODICUM introduces about 20-25MB RAM overhead for each agent type. It introduces 80% CPU overhead on the Mediator, 30% on the Job Creator, and no apparent change to the Resource Provider. This is acceptable since the BBB devices are resource-constrained and so the overhead will be less significant on more powerful compute nodes.

3.8 Conclusions

An open market of computational resources, where resource owners and resource users trade directly with each other has the potential for greater participation than volunteer computing and more competitive pricing than cloud computing. The key challenges associated with implementing such a market stem from the fact that any agent can participate and behave maliciously. Thus, mechanisms for detecting misbehavior and for efficiently resolving disputes are required. In this paper, we propose a smart contract-based solution to enable such a market. Our design deters participants from misbehaving by resolving disputes via dedicated Mediators and by imposing enforceable fines through the smart contract. This is possible because we recognized that the results do not need to be globally accepted, convincing the JC will often suffice. We learned that due to the limitations of Ethereum our platform is only suitable for long-running tasks, but there is space between the cost of electricity and AWS for a platform of this nature. Future work is looking into other platforms that support smart contracts, as well as leveraging improvements to Ethereum.

Table 3.1: MODICUM Parameters and System Constraints

MODICUM Smart Contract (SC)	
θ	penalty rate set by the contract
d	deposit by JC and RP for collateral prior to transaction
d_{min}	minimum security deposit
g_m	cost of requesting mediation
g_r	cost for an RP to participate; includes the costs of submitting the offer and the results, as well as partial payment to the Solver for a match accepted by the smart contract
g_j	cost for a JC to participate; includes the costs of submitting the job offer, as well as partial payment to the Solver for a match accepted by the smart contract
π_d	payout to wronged party when *deception* is detected
n	number of times Mediator executes a disputed job
Mediator (M) and Directory (D)	
π_m	payout to the *Mediator* when it is invoked
π_a	payout to the Mediator and Directory for being *available* for the duration of the job, which also covers the Solver match payment
Job Creator (JC)	
p_v	probability that JC verifies a job result (verification rate)
p_a	probability that a correct execution of a non-deterministic job returns a "normal" result for which the JC will not request mediation (functionally, this is an indicator of how honest the JC is)
π_c	payment from JC to RP for successfully *completing* a computation
b	JC's benefit for finished job minus cost of submitting job
c_v	JC's cost to verify a job result
Resource Provider (RP)	
π_r	payout that *resource* provider asks for completing the job
p_e	probability that RP intentionally *executes* the job correctly ($1-p_e$ is the probability of cheating)
c_e	*execution* cost for RP to compute job
c_d	computational cost for RP to *deceive* and create wrong answer
System Constraints	
1	$b > \pi_c + \pi_a + g_j$ for honest JC
2	$\theta \geq 0$ the penalty rate cannot be negative
3	$n > 0$ else the mediator does not re-execute the job
4	$c_e > c_d > 0$ else the RP never has incentive to cheat
5	$\hat{\pi}_c \geq \pi_c \geq \pi_r > c_e$; if $c_e \geq \pi_r$, the RP will abort the job; and if $\pi_r > \hat{\pi}_c$, then that job and resource offer match is disallowed by the contract
6	$d \geq d_{min}$

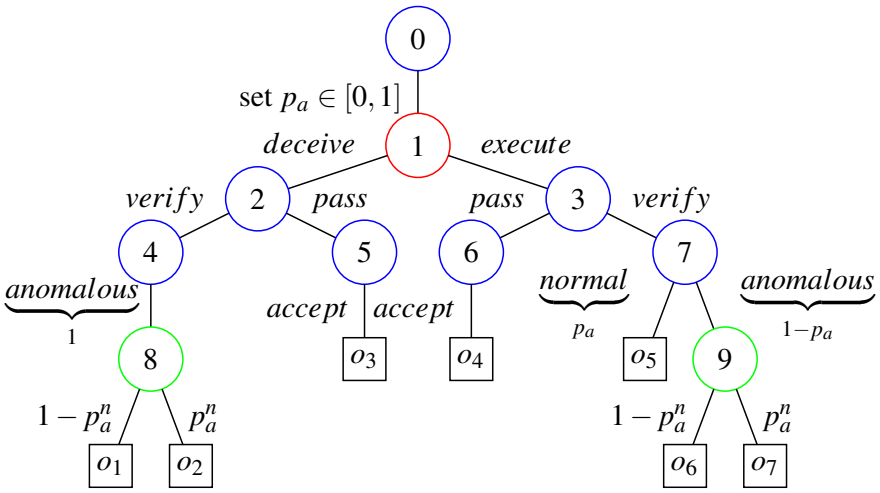


Figure 3.4: Extensive-form game produced by the MODICUM protocol. Blue nodes indicate JC moves, red nodes indicate RP moves, green nodes indicate Mediator's probabilistic outcome. The game is sequential, but the decisions are hidden, so we treat it as a simultaneous move game. Each outcome has payouts for the agents, which are found in Fig. 3.5

Outcome	Party	Contract Payoff	Self Benefit	Reward(r)
o_1	RP	$\pi_d - g_r - \pi_a$	$-c_d$	$\pi_d - c_d - g_r - \pi_a$
	JC	$-g_j - d - g_m - \pi_a$	$-c_v$	$-g_j - d - g_m - c_v - \pi_a$
o_2	RP	$-d - g_r - \pi_a$	$-c_d$	$-d - g_r - c_d - \pi_a$
	JC	$\pi_d - g_j - g_m - \pi_a$	$-c_v$	$\pi_d - g_j - g_m - c_v - \pi_a$
o_3	RP	$\pi_c - g_r - \pi_a$	$-c_d$	$\pi_c - g_r - c_d - \pi_a$
	JC	$-g_j - \pi_c - \pi_a$	0	$-g_j - \pi_c - \pi_a$
o_4	RP	$\pi_c - g_r - \pi_a$	$-c_e$	$\pi_c - g_r - c_e - \pi_a$
	JC	$-g_j - \pi_c - \pi_a$	b	$b - g_j - \pi_c - \pi_a$
o_5	RP	$\pi_c - g_r - \pi_a$	$-c_e$	$\pi_c - g_r - c_e - \pi_a$
	JC	$-g_j - \pi_c - \pi_a$	$b - c_v$	$b - g_j - \pi_c - c_v - \pi_a$
o_6	RP	$\pi_d - g_r - \pi_a$	$-c_e$	$\pi_d - g_r - c_e - \pi_a$
	JC	$-g_j - d - g_m - \pi_a$	$b - c_v$	$b - g_j - d - g_m - c_v - \pi_a$
o_7	RP	$-d - g_r - \pi_a$	$-c_e$	$-d - g_r - c_e - \pi_a$
	JC	$\pi_d - g_j - g_m - \pi_a$	$b - c_v$	$b + \pi_d - g_j - g_m - c_v - \pi_a$

Figure 3.5: Game outcomes and payments in Fig. 3.4. For example, o_1 is the outcome when the RP deceives, the JC verifies, and the Mediator finds non-determinism and faults the JC. The reward for the agents is the sum of contract payoff and self-benefit and is denoted r_{o_i}. The Mediator is not included in the table: in every outcome, it receives π_a; when the JC requests mediation, the M also receives π_m, which is drawn from the faulty party's deposit d.

Table 3.2: RP payoffs by decision

	verify	pass
execute	$\overbrace{-c_e - g_r - \pi_a + \pi_c\left(np_a^n(p_a-1)+p_a+p_a^n\theta(p_a-1)\right)+\pi_d(1-p_a)(1-p_a^n)}^{U_{EV}^{RP}}$	$\overbrace{\pi_c - c_e - g_r - \pi_a}^{U_{EP}^{RP}}$
deceive	$\underbrace{-c_d - g_r - \pi_a + p_a^n\pi_c(-n-\theta)+\pi_d(1-p_a^n)}_{U_{DV}^{RP}}$	$\underbrace{\pi_c - c_d - g_r - \pi_a}_{U_{DP}^{RP}}$

Table 3.3: JC payoffs by decision

	verify	pass
execute	$\overbrace{b - g_j - \pi_c(n+\theta)(1-p_a)(1-p_a^n)+(1-p_a)(-g_m+p_a^n\pi_d)-c_v-p_a\pi_c-\pi_a}^{U_{EV}^{JC}}$	$\overbrace{b-g_j-\pi_a-\pi_c}^{U_{EP}^{JC}}$
deceive	$\underbrace{-c_v-g_j-g_m+p_a^n\pi_d-\pi_a-\pi_c(n-p_a^n(n+\theta)+\theta)}_{U_{DV}^{JC}}$	$\underbrace{-g_j-\pi_a-\pi_c}_{U_{DP}^{JC}}$

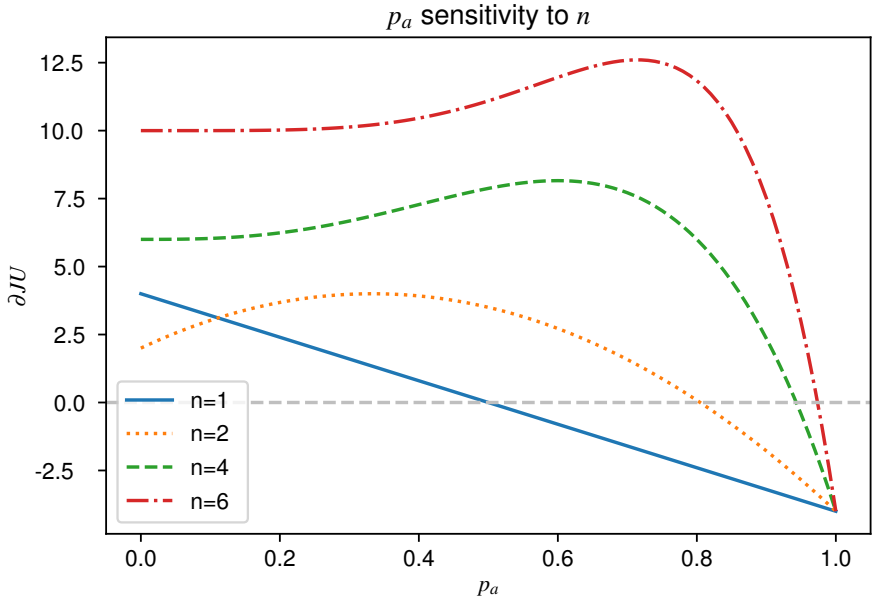
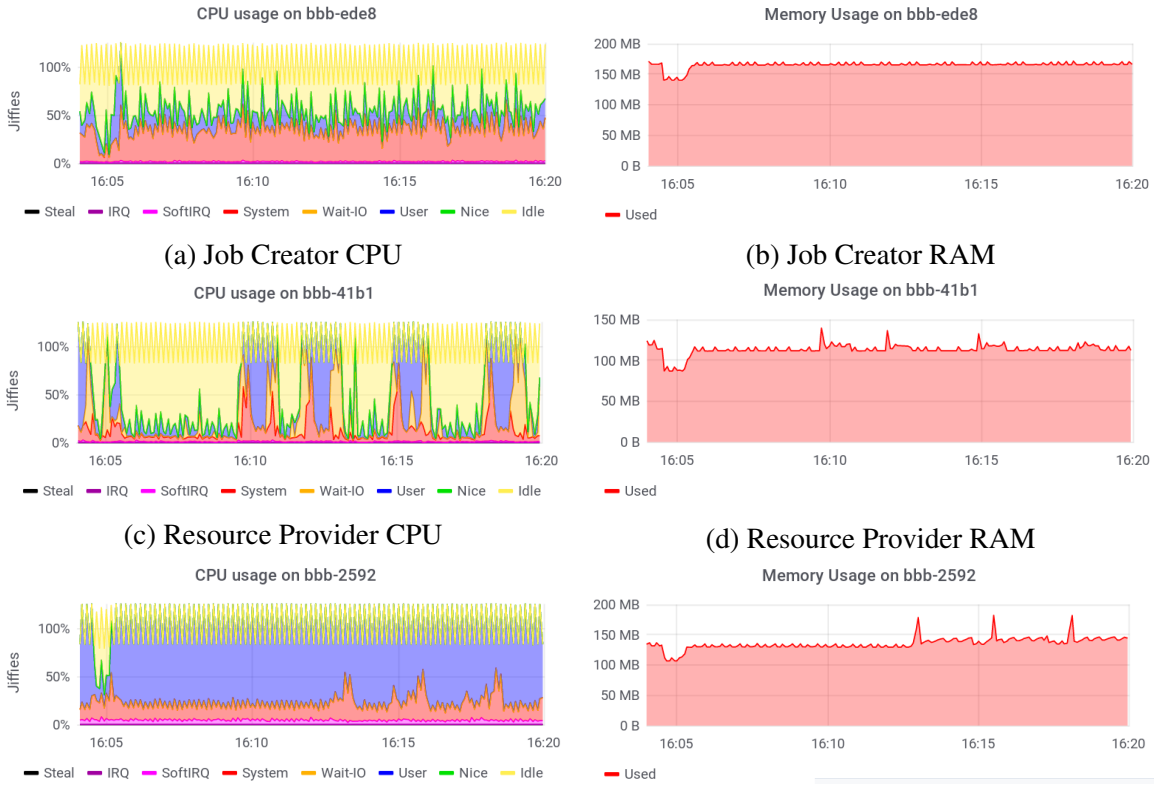


Figure 3.6: We vary the value of n and plot p_a against $\frac{\partial U^{JC}}{\partial p_a}$. This shows that as n increases, so does the optimal value of p_a (zero-crossing of derivative curve). Parameter values are $\pi_c = 2$, $g_m = 0$, $\theta = 0$, $c_v = 1$, $b = 4$, $p_e = 1$

Table 3.4: Simplified JC payoffs to assess dominant strategy with $\pi_d = \pi_c$

	verify	pass
execute	$-\pi_c(1-p_a)(1-p_a^n)(n+\theta+1)+$ $(1-p_a)(-g_m+2\pi_c)^{*1,2}$	$c_v^{*3,4}$
deceive	$2\pi_c - g_m - \pi_c(1-p_a^n)(n+\theta+1)^{*1,3}$	$c_v^{*2,4}$



(a) Job Creator CPU

(b) Job Creator RAM

(c) Resource Provider CPU

(d) Resource Provider RAM

(e) Mediator CPU

(f) Mediator RAM

Figure 3.7: MODICUM Total Resource usage. Each plot shows the resource consumption on a node.

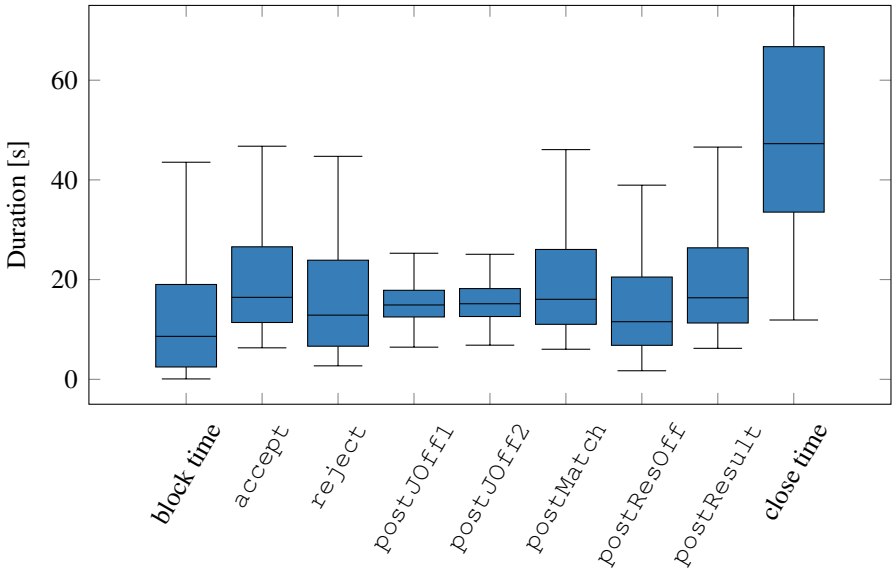
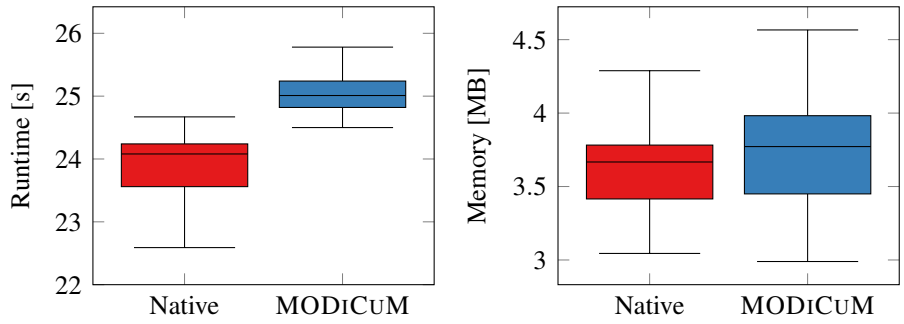


Figure 3.8: Duration of MODICUM function calls during nominal operation.



(a) Runtime (b) Average memory

Figure 3.9: Running time and memory usage on MODICUM and native execution.

Chapter 4

Efficiency is a Challenge – Using Distributed Ledgers with Blockchains – A Stream Computing Case Study

4.1 Overview

To address the challenges present in MSCPS, a platform was developed for outsourcing computations to edge devices (MODiCuM) that seeks to minimize the costs associated with reaching consensus through the use of a Mediator. The initial implementation of MODiCuM was limited to *batch* jobs, which were isolated from outside systems. Ultimately, the capability to outsource *streaming* computations, which make it possible to extract the time-value of data, is required. This is because many critical real-time industrial internet applications fall in that category and are becoming more common with the growth of the industrial internet-of-things.

To support stream computation, inputs need to be processed faster than they arrive, but since the computation is outsourced, assurance that the results are correct is also required. Existing streaming solutions do not provide that assurance. On the other hand, trusted computation solutions incur monetary and computational costs that are not compatible with the volume and velocity of streaming data. The question then is: is there a solution that provides both?

The solution presented here relies on the recognition that only key operations require correctness assurances, and through careful partitioning of market functions we can provide both support for stream processing at low cost, and trust when required. The two primary design features of this architecture are 1) an append-only data structure, in which new events are added to a distributed ledger. The append sequence mimics the temporal nature of incoming data and indexes it spatially for quick spatial-temporal queries. The

primary implementation mechanism is Apache Pulsar [145] using Apache BookKeeper [146] to implement the ledger. While similar systems such as Kafka [147] implement a data store per topic, BookKeeper can distribute the topic content[1] across nodes and mix multiple topics within a node. 2) A universally trusted computing mechanism to manage those aspects which would compromise the market if manipulated. This is implemented using the Ethereum blockchain via "smart contract". This allows us to reap the benefits which come from using blockchain (namely, trust among participants) while facilitating efficient execution of streamed computations. The challenge then is integrating these two systems. Specifically, how to use the trusted computing mechanism to ensure that rational participants will do what they claim they will do. This is accomplished through the construction of a protocol that enforces a game that verifies participant outputs to ensure that rational participants will behave in expected ways. However, we must also ensure that the verification mechanism used has minimal impact on the stream processing performance. To minimize the impact, the trusted computing mechanism is only required to compare two hashes, and disputes are resolved by *Mediator* that is trusted by the relevant participants. Within this solution participants can make certain trade-offs in the form of performing speculative work, by which we mean a Supplier can start processing service inputs before the allocation is confirmed on the blockchain, trusting that it will eventually be added and that it will eventually be paid. Or it can wait for the allocation and signatures to be confirmed before starting work.

Integrating with Pulsar provides additional benefits such as runtime monitoring. To account for participant failures timeouts are also included to detect when agents are not functioning. As part of this work, initial experiments were run to set a baseline for the platform's operation and visualize its normal execution. As this solution is further developed, its performance can be compared against this baseline to detect other abnormal behaviors that may arise.

[1] A topic is a named tag associated with a type of of message.

The work presented in this chapter has been submitted, and at the time of this writing is awaiting review.

- S. Eisele, M. Wilbur, T. Eghtesad, F. Eisele, A. Mukhopadhyay, A. Laszka, and A. Dubey, "Protocol, strategies and analysis for enabling a distributed computation market for stream processing," October 2020, Pending Review.

4.2 Introduction

Emerging Trends Online, or stream processing, allows Internet of Things (IoT) and Smart City services to extract useful information from real-time data. Examples include real-time availability maps for dockless scooters [148], improved emergency response procedures [149], energy usage estimation and subsequent optimization for transit vehicles [150], real-time occupancy maps [151], and traffic density and pedestrian density estimation from intersections [152]. These applications are time-sensitive, necessitating rapid processing of the inputs. This is the purpose of modern stream processing engines such as Spark, Flink, Kafka, Pulsar, StreamQRE, and Heron [153, 154, 155, 156].

Such applications are commonly hosted on cloud computing platforms. This can be expensive for cities[2], especially for real-time high-velocity and high-volume data analysis. Recently, an alternative paradigm known as dispersed edge computing has been conceptualized [157, 158], where the compute power of nearby resources is leveraged. This paradigm is still in nascent stages and does not necessarily address the problem of where the resources to perform dispersed computing come from, for example by utilizing the slack computing capacity at the edge. These approaches lack principled mechanisms that incentivize owners to share slack resources with other stakeholders. This *surplus computing capacity* is the portion of computing resources that remains after they have been utilized to satisfy the requirements of their intended use. It is estimated that there are hundreds of exaFLOPs

[2]In a personal communication with a large cloud vendor, a southern U.S. metropolis had to undertake an expense of $100K dollars per year for setting up real-time data processing applications for managing scooters.

of surplus compute capacity available [159, 160]. The advantage of using these surplus resources is in reduced costs [3] and in the distributed nature of devices, which promotes resilience and diffuses network congestion. This surplus capacity is not generally accessible currently. This presents an opportunity to design a market that can effectively trade this latent surplus computing capacity.

State of the art Work on aggregating surplus resources for outsourcing computational tasks via a market exists. For example, [161, 162] aggregate surplus compute resources, but only from trusted entities like internet service providers. This sidesteps the problem of validating results in the same way that cloud providers do but leaves the bulk of the surplus resources untapped. Other works such as [163, 123] allow mistrusted entities, and rely on blockchain-based distributed ledgers, where there is no central trusted entity; instead, trust is distributed among the participants, to provide trusted compute. Recognizing that blockchain-only systems are inefficient, slow, and have limited throughput these design protocols that perform the computation and verification on standard compute nodes and only rely on the blockchain to control market operation. However, they do not account for fundamental requirements of stream processing and delay providing outputs until after verification, making them only suitable for batch processing.

Challenges To support stream computation with surplus computation several challenges must be addressed. Obviously, for stream processing, the inputs need to be processed faster than they are generated, and since the computation is outsourced, some assurance that the results are correct is required. Existing solutions (like those mentioned above) do not provide that assurance or are not equipped to validate results in a manner amenable to stream processing, *i.e.*, near-instantaneous and inexpensive due to the volume of data processed. Additionally, streaming processes are often long-running; surplus capacity, on the other hand, is transient and subject to the demands of the primary application. Thus, it is likely that agents who provide surplus compute will not be able to host a service for

[3] The space, hardware, cooling, and operations costs are sunk costs, which are already paid for to support the devices' primary applications.

its entire life-cycle. The time sensitivity of streaming processes means that the interactions between services must be accounted for and they are especially sensitive to failures. A failure that requires restarting the service results in loss of value from the data that should have been processed.

Our Contributions This paper presents a solution that addresses the issues of resource availability, trust, reliability, and time-sensitivity posed by outsourcing stream processing. To gain access to surplus compute capacity, we construct a market that incentivizes *Suppliers* (agents who provide surplus compute) to participate by offering compensation for their otherwise wasted compute cycles. To handle the volatility inherent in surplus resources, *Customers* (stakeholders who have services to outsource) specify a minimum service time, and their offers can be broken up into sub-offers with a duration no shorter than the minimum service time. To address the potential for Suppliers to fail accidentally, Customers are able to request that their service be hosted by multiple Suppliers. *Allocators* are employed to construct allocations, matching Customer and Supplier offers according to feasibility constraints, including replicas and minimum service times.

To address the issue of trust, this solution relies on the recognition that only key operations require correctness assurances, and through careful partitioning of market functions we can provide both support for stream processing at low cost, and trust when required. The two primary design features of this architecture are 1) an append-only data structure, in which new events are added to a distributed ledger. The primary implementation mechanism is Apache Pulsar [145]. 2) A universally trusted computing mechanism to manage those aspects which would compromise the market if manipulated. This is implemented using the Ethereum blockchain via "smart contract". This is similar to [123, 163] in that we use blockchain for limited aspects of the protocol, but the integration with Pulsar allows us to reap the benefits which come from using blockchain (namely, trust among participants) while facilitating efficient execution of streamed computations. The challenge then is integrating these two systems. Specifically, how to use the trusted computing mechanism to

ensure that rational participants will do what they claim they will do.

This is accomplished through the construction of a protocol that enforces a game that verifies participant outputs to ensure that rational participants will behave in expected ways. However, we must also ensure that the verification mechanism used has minimal impact on the stream processing performance. To minimize the impact, the trusted computing mechanism acts as a *Verifier* but is only required to compare two hashes, and disputes are resolved by *Mediator* that is trusted by the relevant participants. Within this solution participants are able to make certain trade-offs in the form of performing speculative work, by which we mean a Supplier can start processing service inputs before the allocation is confirmed on the blockchain, trusting that it will eventually be added and that it will eventually be paid. Or it can wait for the allocation and signatures to be confirmed before starting work.

This solution can provide the substrate for multi-stakeholder applications that rely on a dispersed edge computing paradigm. However, it requires careful consideration of the protocol and parameters of the system because individual participants may be selfish and choose their strategy to maximize their own utility. We show how our mechanism ensures that strategies that would undermine trust in the system are not viable. Thus, we emphasize the following aspects of our solution in the paper.

1. We formally describe the design of the architecture and protocol of the system.
2. We show that our solution enables trust. For this, we provide the design and game-theoretic analysis of an incentive-compatible and individually-rational market mechanism. We prove that if the participants are rational, the system operators can set parameters to discourage cheating.
3. Lastly, we describe real-world application scenarios and examples obtained from our partners and show how the system will work in practice.

Our results show that with this design, rational participants will follow the protocol and benefit from participating in the system, while participants that deviate from the protocol

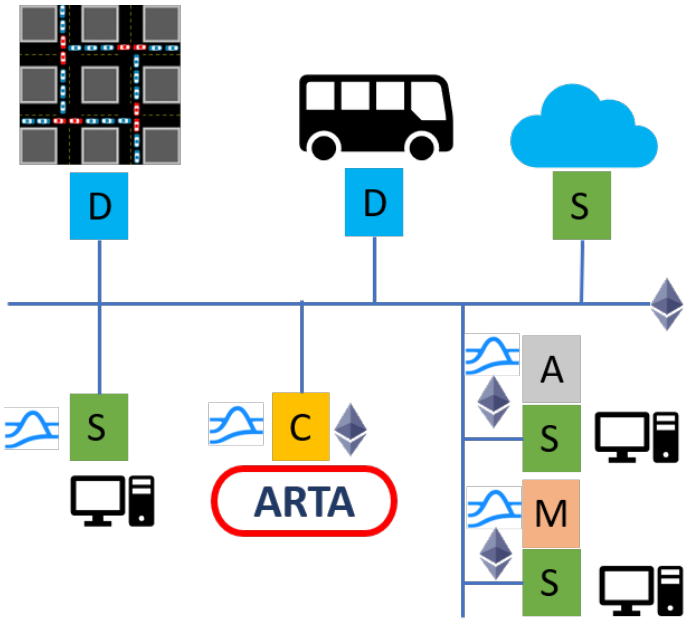


Figure 4.1: Example: participants in the market. Blue lines represent the city network (Both wired and wireless).

incur fines.

Outline We describe the problem formulation and our assumptions in Section 4.3. This is followed by a description of our system and protocol in Section 4.4. We then show the analysis of our system using a game-theoretic formulation Section 4.5. We present our implementation and experimental results (Sections 4.6 and 4.7). We conclude with related research and discussion (Sections 4.8 and 4.9).

4.3 Problem Formulation

The problem that we seek to address is the construction of a robust and trusted market for outsourcing streaming computation.

4.3.1 Basic Problem Formulation

Let μ denote a market for outsourcing of services that rely on stream computations. The market is formed by various agents (Figure 4.1 provides an illustrative deployment).

There are Customers, denoted by C, who have computational services to outsource and can make offers to rent compute capacity. For example, in a community where a regional transit authority (ARTA) requires the ability to process real-time data from its vehicles, the authority can participate in a market as a Customer where there is a set of Suppliers, denoted S, who can make offers to sell surplus compute capacity and host the services. Examples of Suppliers include businesses, universities, end-users, ISPs, and cloud providers. In the figure, D are the streaming data sources of C who wants to process the sensor data. A and M are Allocators and Mediators and will be explained later in the paper.

Services We denote a streaming service by ψ, which has a data input rate of λ and requires R_c MB of memory and I_c CPU instructions (in millions) to process each input. The service also has a finite lifecycle defined as $\Delta_c = c_{end} - c_{start}$ where c_{start} and c_{end} denote the start and end clock times of the service, respectively. We assume that for a specific service, Customers can estimate I_c, R_c, λ, and Δ_c. An example of such a service is the occupancy detection application that several transit authorities deploy [164, 165].

Resources The surplus RAM R_s and CPU resources I_s of Suppliers are available only for a limited duration defined as $\Delta_s = s_{end} - s_{start}$, subject to the demands of the owner of the resource. We assume Suppliers can estimate I_s, R_s, and Δ_s.

Assumption 1. *We note that spare CPU cycles and RAM are not the only resources that may be required to provide a service; network bandwidth, disk space, GPU cycles, etc. may also be required. However, these resources do not significantly change the problem of outsourcing online computation. As a result, we do not include them in our system model. In particular, we point out that the surplus compute market we design is meant for urban communities that have already begun implementing smart city and IoT stream-computing applications. To support these applications, cities must have access to a robust high-speed wide area network which is crucial to facilitate data transfer. Since such a network is required regardless of whether the computation occurs on the edge or in the cloud, we make the reasonable assumption that the market has access to a network that is reliable.*

We primarily focus on computation aggregation and not communication constraints.

Offers We denote the sets of Customer and Supplier offers as O_c and O_s, respectively. A Supplier offer $os \in O_s$ is a tuple that includes: $Supplieraccount$, the account that posted the offer; I_s, the amount of surplus instructions (in millions) per second available; R_s, the amount of RAM available; s_{start}; s_{end}; and π_{xmin}, the minimum price the Supplier is willing to be paid per million instructions. A customer offer $oc \in O_c$ is a similar tuple that includes: $Customeraccount$, the Customer account that posted the offer; I_c, CPU instructions (in millions) to process each input; R_c, the amount of RAM required; c_{start}; c_{end}; π_{xmax}, the maximum price the Customer is willing to pay per million instructions; λ, data input rate of the service; *name*, the name of the service; r, the number of Supplier replicas that the Customer want to host the service. Accounts are identifiers for the Suppliers and Customers.

A Supplier offer and a Customer offer can be matched to form an *allocation* Ω if:

$$I_c\lambda \leq I_s \text{ and } R_c \leq R_s \text{ and } \pi_{xmin} \leq \pi_{xmax} \tag{4.1}$$

$$[s_{start}, s_{end}] \cap [c_{start}, c_{end}] \neq \emptyset \tag{4.2}$$

In other words, a pair of offers is matchable if the Supplier has sufficient resources, there exists a price that both Customer and Supplier would accept, and the times overlap. We assume access to an algorithm that, given a set of Supplier and Customer offers, can find a match if one exists. This is a classic example of a resource allocation problem for which there are many different algorithms [166].

Allocation An allocation is a service contract between the participants according to their offers. We define an allocation as a tuple which includes: $customer$, the allocated Customer; $\{suppliers\}$, a set of allocated Suppliers; a_{start}, the allocation start time; a_{end}, the allocation end time; *name*, the service name; π_x, the transaction price per million instructions such that $\pi_{xmin} \leq \pi_x \leq \pi_{xmax}$; and r, the number of Supplier replicas. For the

transaction price π_x any value between π_{xmin} and π_{xmax} is feasible, it is determined by the Allocator based on its allocation mechanism (*e.g.*, double auction, fixed price, etc.). Effectively, an allocation declares which offers were matched, the service price, the start and end times, the transaction price.

The duration of the allocation is defined as $\Delta = a_{end} - a_{start}$. Recall: I_c is the number of CPU instructions (in millions) to process each input and π_x is the transaction price per million instructions. For convenience, we define π_s to be the price to process a service input: $\pi_s = I_c \pi_x$. Therefore, the total value of an allocation is $\pi_{total} = \pi_s \lambda \Delta$. Note the basic problem is to find the allocations given a set of offers.

Processing A Supplier correctly processing an allocation means that the Supplier is able to process the incoming data at least as fast as the incoming data rate λ, otherwise data would eventually be lost. Formally: $I_c \lambda \leq I_s$.

4.3.2 Additional Considerations

To enable trust and robustness in the market, we must consider the resource availability, trust, reliability, and time sensitivity problems posed by outsourcing stream computations, while keeping overhead costs low.

Resource Availability Consider a Customer needs to run a service continuously from c_{start} to c_{end} in order to receive the expected benefit. Suppliers are likely unable to host a service for its entire life-cycle due to the demands of the Supplier's primary applications. To address this the total Customer offer must be "chunked" into sub-offers that correspond with the offers by the Suppliers in the System.

Trust Given a particular market mechanism, each agent i (a Supplier or Consumer) has a utility function U_i, and a set of actions A_i to choose from. We assume that the agents in the system are rational and choose to maximize their utility. For example, agent i chooses action $a^* \in A_i$ such that $a^* = \arg\max_{a \in A_i} U_i$. This means that Suppliers have an incentive to neglect processing service inputs, because of the electricity costs $\pi_{s\varepsilon}$, if the benefits are

greater than the consequences. We therefore face the challenge of designing a mechanism that needs to achieve two crucial goals: first, it should incentivize rational agents to participate in the market, and second, it should incentivize truthful behavior. To aid this endeavor we specify a security deposit π_d the Customer and Suppliers need to provide in order to participate. π_d is computed as $\pi_d = \pi_s \times \rho$, where ρ is a penalty rate defined by the market. We show how the desired behavior is enforced by considering game-theoretic models based on rational actors. We do not consider malicious agents which might have incentives that are external to the market. We also require access to a verification mechanism that can detect cheating.

Reliability In any large distributed system failures are inevitable so platforms must be designed to account for them. The Customer would like to minimize the costs that it pays to have a service hosted. However, if correct results are not provided on time then it receives no benefit. As a result, the Customer may need to hire multiple Suppliers to ensure reliability. The number of suppliers needed for a specific job is exogenous to our market and is typically a function of the specific job and the risk profile of the Customer. We assume that the time to failures can be approximated by an exponential distribution, a widely used model in reliability engineering [167]. Let the average rate of failure (per unit time) be denoted by q. The probability of failure can then be denoted by $p_f = 1 - e^{-qt}$. We assume that the probability of failure is common knowledge to all participating agents. Conditional on p_f, the Customer must hire *r replicas* to ensure $P(b)$ probability of success such that:

$$r = \frac{1 - P(b)}{p_f} \tag{4.3}$$

Verification As mentioned we require a verification mechanism, however, introducing verification may also introduce delays in the output stream. Thus, any verification that

occurs must be essentially instantaneous or performed after the output has been released. Verification must also be inexpensive due to the volume of data processed. To this end we assume that the services are deterministic, meaning that repeated processing of a specific output provides the same output every time. This assumption forms the basis for verifying Supplier outputs. Specifically, verification compares outputs to detect errors, because this is fast. Naturally, the assumption also means that the market we design cannot be used for applications that involve random sampling, or for training machine learning models that can involve non-deterministic estimators. However, it can be used for inference using trained machine learning models.

Assumption 2. *We require the stream applications to be stateless. Stateful services cannot be split across streams, do not scale well, are not robust to failure since the state is lost, and introduce time delays to recover. The assumption of statelessness does not mean that we cannot handle state at all. If the state is stored external to the service then it can be read to perform the processing. Alternatively, if the service operates on windowed time-series data then it can be transformed into a data stream that merges the inputs into a single sample, thereby allowing the service to be stateless. Specifically, "non-re-entrant functions" with implicit states stored in the service itself are not supported.*

Mediation Verification by comparing result, though fast, may not be able to determine which inputs were actually correct. For this reason, we introduce the notion of mediation. Mediation is performed by a trusted agent that duplicates the contended output and compares it against the previous results. Mediators are assumed to be few or have limited resources and for this reason, cannot be used to host all services.

4.4 The System

We now describe the system components, protocols, and the strategy analysis for rational actors.

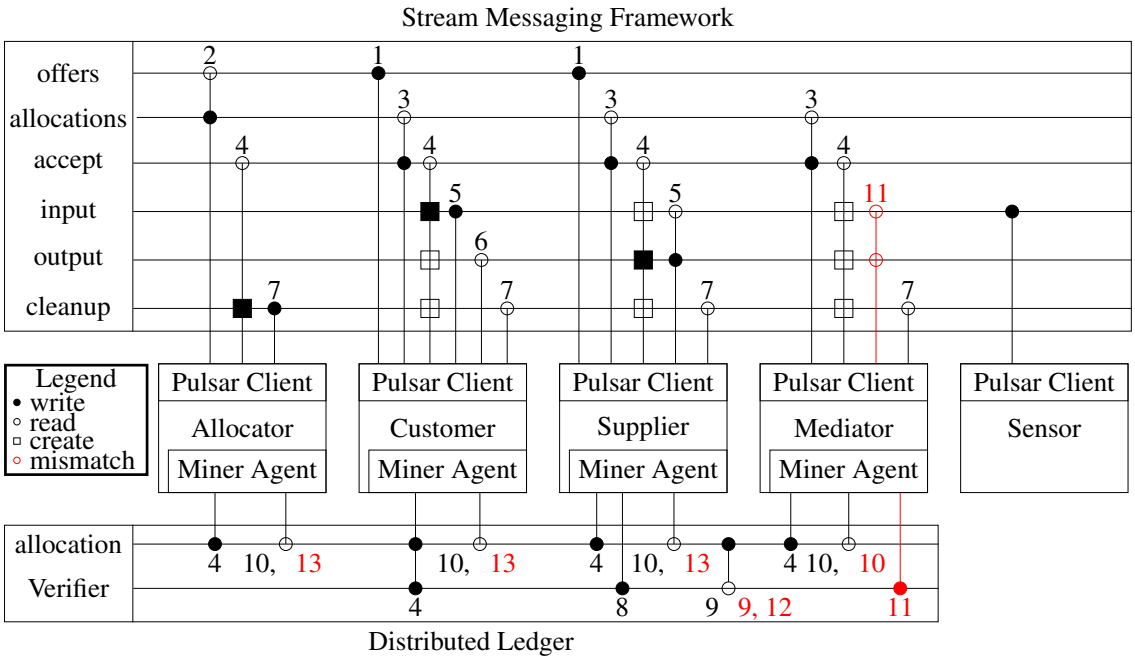


Figure 4.2: Horizontal lines represent communication channels between participants. Vertical lines represent functions that write to (filled circle) and/or read from (open circle) channels. For example, the Supplier reads that an allocation was accepted on the accept channel which causes it to create (denoted by a square) a reader on the input channel, a writer on the output channel, and a reader on the cleanup channel. The functions occur in the numbered sequence. Red numbers only occur if the outputs checked by the Verifier do not match.

4.4.1 Enabling Market Capabilities

Allocators As discussed, we require the capability to match offers in our system to form allocations. Suppliers are offered an incentive to implement an allocation algorithm and provide this service. We refer to such suppliers as *Allocators*. The set of Allocators is denoted A. Each Allocator in the system is free to implement an arbitrary (and correct) matching algorithm. We assume each Allocator is fair, *i.e.*, does not target offers to leave unmatched.

Blockchain Blockchain-based distributed ledgers have been of interest in recent years for systems where there is no central trusted entity, instead trust is decentralized among the participants. However blockchain-only systems are inefficient, slow, and have limited

throughput making them unsuitable for streaming applications that are time-sensitive and long-running. For example on Ethereum, blocks are committed approximately once every 15 seconds [168], and confirming a particular transaction takes at least 2 blocks, and can take up hundreds depending on the amount paid in transaction fees [169]. Stream Processing, on the other hand, needs to be comparatively faster. The challenge we face is to enable speed but preserve trust.

Messaging and Streaming platform With Blockchain To address this challenge we supplement the blockchain (box on the bottom of Fig. 4.2) with a distributed messaging and streaming platform (box on the top of Fig. 4.2). Participants communicate through both of these ledgers, and both ledgers record state.

To understand how the participants in the market communicate we reference Fig. 4.2. Communication happens using *channels*. A channel is a topic in a pub/sub system with authentication and authorization controlled by the stakeholders that communicate on that channel. For example, the offers channel is public so any stakeholder can read it, and any stakeholder can write their own offers[4]. In contrast, the input channel is private and only the Customer and its sensors can write to it, while only the allocated Suppliers and Mediator can read it. Using a streaming platform we can efficiently communicate the bulk of information. Then, deploy a smart contract on a blockchain to implement elements that are critical to preserving trust in the system such as tracking the state of allocations, providing verification, and transferring payments.

4.4.2 Protocol

The protocol is subdivided into stages of interactions. The sequence of events is denoted with the numbered labels in Fig. 4.2, which we use to assist in describing the market protocol. We refer to these events with circled numbers in the text (*e.g.*, ①). We also refer to the state machine shown in Fig. 4.3 as we describe the protocol, which depicts how the

[4] they cannot modify the offers written by other stakeholders

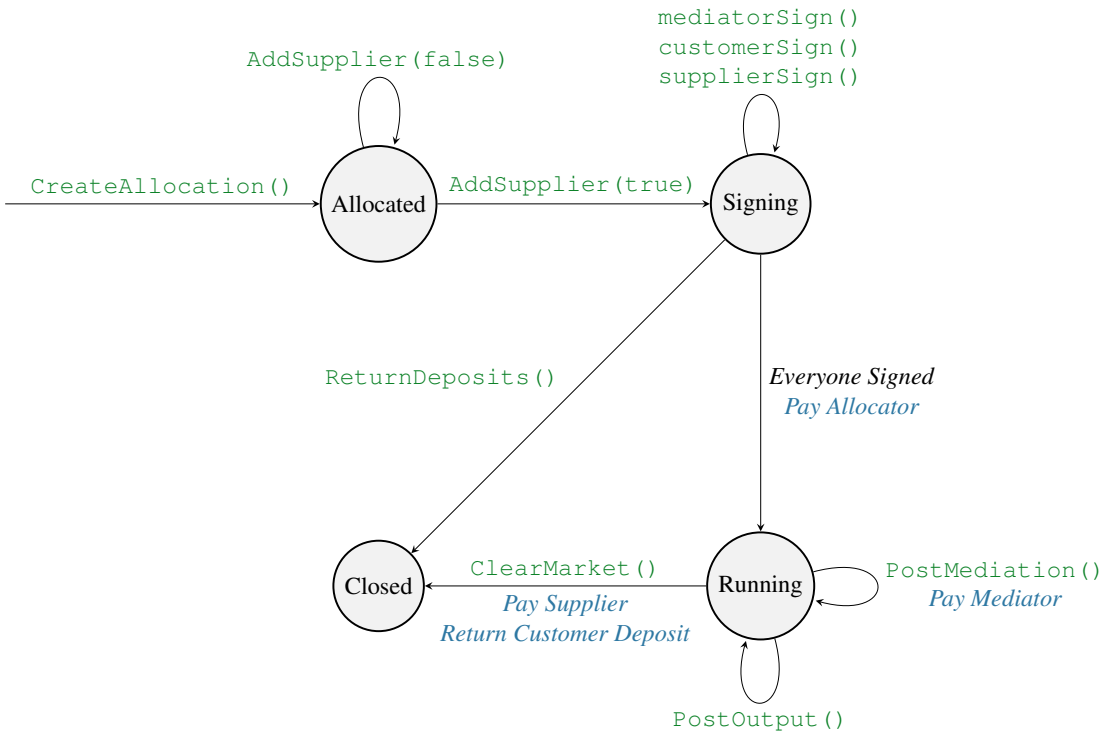


Figure 4.3: State of allocation on the smart contract represented as a State Diagram. Our smart contract code can be found online [170] and functions are described in Section 4.4.2.

smart contract tracks the state of each allocation. We also explain the participants and the cost structure we use. As a convention, text in `teletype` font represents smart contract functions and text that is *italicized* are state machine states.

Making Offers The protocol begins with the Customers and Suppliers constructing their offers as described in Section 4.3. These offers are then sent on the offers channel ①.

Creating Allocation The Allocator then reads the offers channel ②. This causes it to execute a matching algorithm and construct an allocation, if one exists. It then sends the allocation on the allocations channel.

Accepting Allocation The Customers and Suppliers read the allocations channel ③ and send a message on the accept channel to specify if they accept the allocation or not. The participants read the accept channel ④, and if all the allocated participants accept, then the Allocator calls the blockchain smart contract function `createAllocation`. The state of the allocation is initiated to *Allocated*. Afterward, for each Supplier of this allocation,

the Allocator adds the Supplier and its offer hash (`AddSupplier`) to the allocation. When all Suppliers are added, the state of the allocation transitions to *Signing*. This function call incurs gas cost π_{ac}. Also in response to all of the allocated participants accepting, the other participants must construct the necessary channels for the service to operate. In conjunction with this, the participants read the blockchain for the *Signing* state change event from the smart contract. When it appears, the participants check the allocation to make sure that it matches the specific allocation that the Allocator sent on the allocations channel and, if it does, sign (*e.g.*, `customerSign`) the allocation on the blockchain by submitting their security deposits. This varies somewhat between participants. As part of signing the allocation, the Customer commits to n test input/output pairs[5]. This causes the Customer to incur the cost $n\pi_{cg}$ to generate a set of hashed outputs and writing that hash to the smart contract π_{cc}. When all of the participants sign the contract, the state of the allocation is automatically changed to *Running*.

The Customer, Suppliers, and Mediator also incur a cost for signing the allocation and pay the Allocator for its service. These costs are lumped as π_a. Once all participants have signed, the Allocator receives π_A as payment for its service.

Service Execution After the channels are constructed the Supplier can begin reading ⑤ on the service input channel and sending outputs on the service output channel[6]. For each correctly processed input, the Customer reads ⑥ it receives a benefit b and the Supplier incurs electricity cost $\pi_{s\varepsilon}$.

Verifying Outputs At the end of the allocation, the Allocator sends a message on the cleanup channel ⑦ notifying the participants to end the allocation. The Customer informs the Supplier which n outputs are to be verified. The Supplier identifies the corresponding outputs and sends them to the Verifier on the blockchain ⑧, calling `PostOutput`. The Supplier incurs a cost of π_v performing this operation. The Verifier on the blockchain stores the output of the Supplier and compares it against the output provided by the Customer

[5]We describe how this is done and why it is necessary in Section 4.5

[6]Note: this can start before signing is complete

during allocation.

Mediation and Closing the Allocation If the outputs do not match then the Verifier emits a MediationRequested event (red ⑨) to the blockchain, where the Mediator reads it (red ⑩). The Mediator then reads ⑪ from the input and output channels and re-processes the n inputs and compares its output against the Suppliers and Customer outputs to determine which participants are at fault. This incurs costs of $n(\pi_{s\varepsilon} + \pi_{cg})$. The Mediator then writes the result to the blockchain, calling `PostMediation`, which incurs a cost of π_{mc}. Then, the Verifier transfers payments and closes the allocation ⑫ by calling `ClearMarket`. The call causes the state of the allocation to transition to *Closed*, which is read (red ⑬) by the participants. If a Customer or Supplier output did not match the Mediator output they are fined π_{cd} or π_{sd} respectively. From these fines, the Mediator is paid $n(\pi_s)$. If the outputs match, the Verifier transfers payments; the Supplier receives $\lambda_\Delta \pi_s$, the Customer pays $\lambda_\Delta \pi_s$, and the Mediator receives π_m for being available. The Verifier also closes the allocation (black ⑨), calling `ClearMarket` causing the state of the allocation to transition to *Closed*, which is read (black ⑩) by the participants.

Timeouts At each point in the protocol where a participant is waiting on the output from another (contract signing, service setup, service cleanup, service verification) we include timeouts in the implementation. If the output is not received within a specified time window the sender is considered failed, does not get paid, and is fined a portion of its deposit rather than the full deposit as it would have if it had submitted an incorrect result.

4.4.3 Allocation Duration

A Customer may need to run a service for a duration that is longer than the availability of any of the Suppliers. In that case, the offer must be ”chunked” into sub-offers with durations that correspond with the offers made by the Suppliers in the system. Some care must be taken when chunking since each offer that is allocated incurs monetary and time costs. Specifically, there is a time cost associated with constructing an allocation,

denoted by δ_{alloc}, and with setting up the service after the allocation is signed, denoted by δ_{setup}. Allocation setup includes the Allocator writing the allocation to the allocations channel, allocated participants reading the allocation, writing to the accept channel, and optionally waiting for the allocation to be signed on the blockchain. Service setup includes transferring the service and starting the service. To be profitable and avoid thrashing, each service has a minimum service time δ_{min}. For an allocation to be valid $t_{end} > \delta_{min} + \delta_{alloc} + \delta_{setup}$ must be true. The time at which this becomes false is the expiry time is $t_e = t_{end} - \delta_{min} - \delta_{setup} - \delta_{alloc}$.

The Customer can read the active Supplier offers to identify a reasonable service duration, greater than the minimum service time, and construct offers using this value. As part of the allocation algorithm, the Allocator checks if the Customer offer has expired. If the offer has not expired, it checks to see how many replicas the Customer has requested. It then attempts to find that many Suppliers that are available for the duration. If the Allocator can not satisfy an offer it produces a void allocation that serves to notify the Customer that the specific offer is not allocated.

4.5 Analysis: Mitigating Cheating, Collusion and Failures

To analyze the strategic interaction between the Customers and Suppliers, we consider our protocol as a game. We derive the utility functions for the agents by using the cost structure that the protocol imposes. Before we discuss the game we explain why the protocol enforces the following: 1. have the Customer check the Supplier outputs, 2. why the Customer must pay even if the Supplies do no work, 3. implement the Verifier on the blockchain, 4. include a Mediator, and 5. have the Customer commit to n outputs during the allocation,

Customer Verifying Supplier Outputs The Customer checks the outputs generated by the Supplier in order to prevent collusion among the Suppliers. Consider a situation where the check is not enforced. In that case, the Suppliers can collude and agree on a

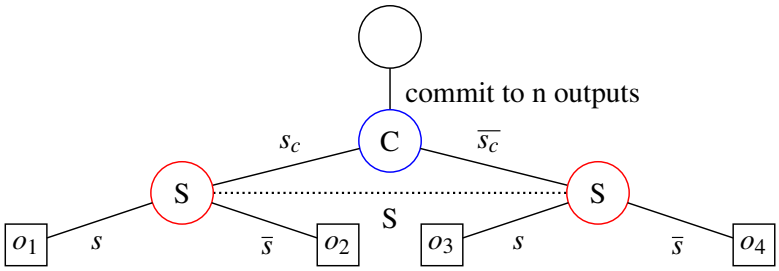


Figure 4.4: Extensive-form game produced by the protocol. Blue nodes indicate Customer moves, red nodes indicate Supplier moves. The game is sequential, but the decisions are hidden, so we treat it as a simultaneous move game. Each outcome has payouts for the agents, which are found in Table 4.4

common output such that their outputs match. Recall that the Verifier only compares the outputs, so a common (albeit incorrect) set of outputs would be accepted. Such behavior is possible, since presumably, colluding can be significantly cheaper in practice than running the service. One way to prevent such behavior is to ensure the existence of an additional Supplier that does not participate in collusion. Such an idea is motivated by the role of "trusted agents" in multi-agent systems [171]. There are two relatively straightforward ways to achieve this. First, the Customer itself can occasionally act as a Supplier, and second, the Customer can occasionally hire a Supplier that it trusts to process an input. The presence of such a Supplier does not negate the benefits of the outsourcing market because the service workload can (and usually will) exceed the trusted resources available to the Customer. Since the Supplier in consideration is trusted by the Customer it does not collude with the other Suppliers. In such a scenario, the Verifier detects the collusion since the outputs agreed upon by the colluding Suppliers do not match with that of the trusted Supplier.

Customer pays regardless A small but key design decision to eliminate undesired behavior is to have the Customer pay $r\lambda_\Delta\pi_s$ at the end of the allocation regardless of the outcome. This decision avoids complications that appear in prior work [163]. If the Customer is refunded when a job fails, it naturally incentivizes Customers to construct jobs that can manipulate the system (since Customers can get work done for free by collecting

the refund), or collude. We understand that this design choice may seem unfair — there is a possibility that the Customer pays for a service that is not delivered. However, in practice, we merely shift the overall cost incurred by the Customer. This shift is due to the fact that if Customers had the incentive to cheat, the system would have incurred costs to build infrastructure to regulate such behavior. Naturally, such cost would have been borne by all agents, including the Customer. Also, we explain shortly how this payment does not surface in the optimal strategy profiles for the agents.

Blockchain Implementation of the Verifier To prevent the Verifier from being able to collude we implement it on the blockchain. The Verifier is only capable of detecting errors, not ascertaining which entities are at fault.

Inclusion of a Mediator Including a Mediator to reprocess inputs allows us to only penalize participants who are at fault. Participants allowlist Mediators and include this information in their offers for the Allocator to construct valid allocations. If a participant chooses to no longer trust a particular Mediator then it no longer includes it in its offers.

Verifying n outputs We now explain the need for the Customer to commit to checking n inputs. Consider the situation where the Customer can check the Supplier outputs but can choose not to.

The Game Model We model the interaction between the Supplier(s) and the Customer as a game. The Supplier's actions are to either process an input (denoted by s) or not (denoted by $\bar{s}$). Similarly, the Customer can choose to process the input (s_c) or not ($\overline{s_c}$) (note that the Customer can hire a trusted supplier to do the processing or do it by itself; this choice is orthogonal to the strategic interaction we consider). The Customer pays π_a to accept the allocation and π_s for the Supplier to process the input. Recall that the Customer pays this regardless of whether the Supplier provides the output. The Customer receives benefit b when it receives the processed output and pays $\pi_{s\varepsilon}$ (the electricity cost of processing the input as defined in section 4.4.2) to process the input itself if it decided to check.

The Supplier also pays π_a to accept an allocation and π_v to have its output verified. To process an input the Verifier incurs $\pi_{s\varepsilon}$ in electrical costs. If the outputs differ, the Verifier records this to the blockchain, which triggers mediation to determine which Customers and Suppliers to penalize. The Mediator posts the outcome back to the blockchain, finalizing the allocation and. triggering payments. If a Customer is at fault it is fined π_{cd}. If a Supplier is at fault it is fined π_{sd}. We show the game matrix for this scenario in Table 4.2.

Theorem 7. *There is no pure strategy Nash equilibrium in the game between the Supplier and the Customer*

Proof. Assume the Customer chooses to process all inputs (say, through a trusted Supplier), then: 1. the Supplier's utility for (s, s_c) is higher than for $(\bar{s}, s_c)$, and as a result, the Supplier's optimal strategy is to truthfully process all inputs. 2. However, the Customer then has a profitable deviation by changing its strategy to process no inputs. 3. If the Customer processes no inputs, then the Supplier also has a profitable deviation and processes no inputs, causing the Customer to deviate and process all inputs as we began. □

Notice, that if the Customer makes a credible threat that it checks some (n) of the inputs, then the game changes such that processing all inputs is the Supplier's dominant strategy. Naturally, the choice of n depends on a specific game and the payment structure. However, for the Supplier to be convinced the Customer's threat is credible, the Customer must have no profitable deviations from processing n inputs. We identify two ways to accomplish this. We describe one of the approaches here (the one that we implement as part of our experiments) and discuss the other approach in the online appendix.

Prior to the allocation, the Customer processes n inputs. Then, as part of accepting the allocation, it hashes those n outputs and commits them to the blockchain. It then mixes those n inputs in with the regular workload inputs.[7] At the end of the allocation, the Customer notifies the Supplier which outputs it must submit. The Supplier submits its output

[7] This assumes that the Supplier cannot distinguish test inputs from workload inputs

hash to the Verifier which compares them against the Customer's committed output hash. If the Customer hash did not represent the correct number of correct outputs, then it is detected and the Customer is penalized. Thus, there is no profitable deviation from processing n inputs for the Customer.

The game that is derived from this protocol can be seen in Fig. 4.4 with payouts in Table 4.4. In this game, (s, s_c) is the dominant strategy for the Customer as long as the utility for $U(s, s_c) > U(s, \overline{s_c})$, which is true when $n\pi_{cg} < \pi_{cd}$. Setting variables as in Eq. (4.4) the Customer will always process n inputs as long as $n < \rho$.

$$\begin{aligned} n\pi_{cg} &< \pi_{cd} \\ \text{Note: } \pi_{cg} = e_c\pi_s,\ e_c < 1,\ &\text{set } \pi_{cd} = \rho\pi_s \\ ne_c\pi_s &< \rho\pi_s \\ ne_c &< n < \rho \end{aligned} \tag{4.4}$$

Similarly (s, s_c) is the dominant strategy for the Supplier as long as the utility for $U(s, s_c) > U(\overline{s}, s_c)$ which is true when $\lambda_\Delta(\pi_s - \pi_{s\varepsilon}) > -\pi_{sd}$. This is true unless the Customer severely underestimates the resources required, in which case the Supplier's output states that the resources allocated were exceeded, or the Supplier underestimated its power consumption in its initial offer. Note that if the Customer sends too few, or bad outputs to the Supplier, the Supplier can call for Mediation.

This formulation presumes that the Supplier processed every input or none of them. However, it is possible that since the Customer is only checking n inputs the Supplier can risk skipping the processing of some inputs to reduce its electricity costs. The utility for the Supplier then depends on if it processed all the inputs whose outputs would be checked. The utility of the Supplier in that case is:

$$
\begin{aligned}
U[S] =& P(s)^n(\lambda_\Delta \pi_s - \lambda_\Delta \pi_{s\varepsilon} P(s)) + \\
& (1 - P(s)^n)(-\pi_{sd}) - \pi_v - \pi_a
\end{aligned}
\tag{4.5}
$$

The Supplier can then solve for $P(s)$ to maximize its utility.

$$
\begin{aligned}
\frac{\partial U[S]}{\partial P(s)} &= nP(s)^{n-1}\lambda_\Delta \pi_s - (n+1)\lambda_\Delta \pi_{s\varepsilon} P(s)^n + nP(s)^{n-1}\pi_{sd} \\
&< \text{set equal to 0, solve for P(s)} > \\
P(s) &= \frac{n(\lambda_\Delta \pi_s + \pi_{sd})}{(n+1)\lambda_\Delta \pi_{s\varepsilon}} \\
&< \pi_{sd} = \rho \pi_s,\ \pi_{s\varepsilon} = e_s \pi_s, \text{ simplify} > \\
P(s) &= \frac{1}{e_s}\left(\frac{n}{n+1} + \frac{n\rho}{\lambda_\Delta(n+1)}\right) \\
&< \text{set } \rho = \frac{\lambda_\Delta}{n} > \\
P(s) &= \frac{1}{e_s}
\end{aligned}
\tag{4.6}
$$

In Eq. (4.6) we see that by requiring the penalty rate ρ equal to $\frac{\lambda_\Delta}{n}$ the maximum utility of the Supplier occurs when $P(s) > 1$ (since $e < 1$) which is not possible. This means this requirement ensures that the Supplier will always process the inputs. ρ is a parameter that is set at the system level on the blockchain, while λ_Δ is specified in the Customer offer and n and π_{sd} are computed by the Allocator.

Keeping threats credible While describing the protocol, we make the following three statements: 1) the Customer "commits to n test input/output pairs", 2) "The Customer informs the Supplier which n outputs are to be verified", and 3) "The Supplier identifies the corresponding outputs and sends them to the Verifier on the blockchain.". If these outputs are sent in the clear, our game breaks or devolves back to a mixed strategy Nash equilibrium. For example, if the Customer sends its n test outputs raw to the blockchain, the

Supplier can read them and copy those outputs. Then, it can simply provide those outputs at the end of the allocation and neglect processing any of the inputs. For this reason, we need to make sure that the outputs are shared in a way that does not break the game. To do this we have the Customer use a hash function to mask the output it sends to the blockchain. Then since the Verifier is comparing outputs the Supplier's output must also be hashed so the results can be compared. To prevent the Supplier from simply copying the Customer's hash it must provide its outputs so that the Verifier itself can hash them to match the Customer outputs.

To do this we first introduce some notation. Let $O_c = \{o_1, o_2, \ldots o_n\}$ represent the set of the Customers n committed outputs where o_i is a particular Customer output. Let $O_s = \{o_{s1}, o_{s2}, \ldots o_{s\lambda_\Delta}\}$ represent the set of λ_Δ outputs produced by the Supplier, where o_{si} is a particular Supplier output. Then to mask the outputs we use the following hash functions:

$$\begin{aligned} \alpha(K) &= \{hash(k_i) : \forall k \in K\} \\ \gamma(K) &= hash(K) \\ \Gamma(K) &= hash(hash(K)) \end{aligned} \tag{4.7}$$

For the hash function we need to use a hash function that is supported by the chosen blockchain implementation - for example, since we use Ethereum we can use keccak256, sha256, and ripemd160[172].

When the Customer sends its outputs to the Supplier so the Supplier can identify which outputs it needs to send to the Verifier, the Customer sends $\alpha(O_c)$. Then to determine which inputs are the test inputs the Supplier hashes all of its outputs and find which hashes are common to the two sets. We call that subset of Supplier outputs O_v where $O_v = \{o_i : hash(o_i) \in \alpha(O_c) \cap \alpha(O_s)\}$. This is because if the Customer sent $\{o_i : o_i \in O_c\}$ then the Supplier could skip processing the inputs and simply use the outputs provided to produce O_v. Or if the Customer sent the index i of each test output the Supplier could

neglect processing the inputs until it received the indices to produce O_v.

When the Customer commits O_c to the blockchain it sends it as $\Gamma(O_c)$. The output set is double hashed because if the Customer sent $\gamma(O_c)$ then the Verifier on the blockchain would have to hash all of the Supplier's n outputs which is expensive. Instead, the Customer sends $\Gamma(O_c)$, requiring the Supplier to send $\gamma(O_v)$ to the Verifier. Then the Verifier only has to hash a single element to compare against the Customer's output hash: $\Gamma(O_c) == hash(\gamma(O_v))$.

4.6 Implementation

Since interacting with a public blockchain like Ethereum is slow (minimum of 15 seconds to record a transaction) and costly ($0.80 base fee with fastest transaction) we minimize the number of interactions with it. While using side-chains is an option, implementations of side-chains are still relatively new and immature [173]. Instead, we choose to use Apache Pulsar, which was developed for robust stream computing applications. These characteristics have been demonstrated in industrial applications [174]. Pulsar is a distributed messaging framework and ledger which supports multiple clusters and multi-tenancy [175]. A tenant can be likened to a user in an operating system. Users operate on processes and files, while tenants operate on topics and clusters of hardware resources. By representing a stakeholder as a tenant the stakeholder is able to manage which other stakeholders have access to its clusters while isolating cluster hardware. [8] Thus clusters and tenants make Pulsar a viable option for implementing and supporting a multi-stakeholder market. [9] [10]

To describe our implementation we refer to Table 4.5. Each stakeholder is represented by a tenant $(t_1, t_2, ..., t_n)$ and their compute resources are placed into clusters $(c_1, c_2, ..., c_n)$.

[8] Pulsar tenants are applied to machine clusters. These clusters contain and isolate the resources owned by the market participants.

[9] This is not the only way to implement the multi-stakeholder functionality we desire, but it is sufficient for our purposes.

[10] There are some weaknesses in the current Pulsar implementation. The management of tenants is not itself distributed.

To participate in the Market, a stakeholder must first add its clusters to the Pulsar instance. Then joining the Market is done by giving the *marketplace* tenant read-access to its *public* namespace, which is on the stakeholder's Marketplace Cluster (MC) [176]. The set of clusters that allow the *marketplace* tenant constitutes the marketplace. After joining the Market if the new stakeholder writes an offer to the *market/public/offers* topic the other stakeholders are able to read it. Likewise, the new stakeholder can read the offers and allocations of other stakeholders. Sharing of topics, and by implication messages, is handled with the Pulsar administrative mechanisms. All stakeholders are implicitly granted read access to the marketplace tenant.

4.6.1 Resilience

The Market is robust since many stakeholders are present and maintain their own clusters. A failure of the blockchain means the miner network has been compromised this undermines the trust in the system and the Market fails. However, on a proof-of-work blockchain, this means an adversary controls more than 50% of the mining power, thus the security of a blockchain is based on having a large number of miners in the network. We assume this failure does not occur. The failure of other participants does not cause the market to fail but may interrupt some allocations.

Essentially, our market design ensures that the consequences of failures are localized to the specific stakeholder that experiences the failure. In the proposed system, as long as there is more than one stakeholder that sends offers to an Allocator and stakeholders have a mutually trusted Mediator, the Market is considered to be operational. Pulsar can detect failures and can automatically fail-over to continue operating if the stakeholder has configured its Pulsar cluster appropriately.

4.7 Experiments and Results

These experiments are performed on a computing cluster that is used to create a representative set of actors. We model the sensors and customers based on applications that we have from our work with a representative Area Regional Transit Authority (ARTA).

4.7.1 Blockchain Analysis

Gas measures the amount of computation required to execute a transaction or make a smart contract function call in an Ethereum network. The total gas cost of a transaction depends on *transaction costs* and *execution costs*. While the transaction costs count for the amount of data written on the blockchain, execution costs count for CPU operations required. The exact code that is executed while making a function call to the smart contract can vary based on many factors. For example, the number of Suppliers affects the number of CPU operations required for making a `ClearMarket` function call since the smart contract must compute the hash of the output hash for each Supplier and compare it to the Customer's committed hash of output hash.

Further, the agent (*i.e.*, Supplier, etc.) making calls to the smart contract decides on the price of gas based on how busy the Ethereum network is. Generally, more price on gas means that the transaction will be executed faster on the network. Table 4.6 shows the approximate gas cost required for making each function call to the smart contract based on the sum of their *transaction costs* and *execution costs*, and their equivalent of USD price given a typical gas cost of 20 Gwei.

In [163] the gas cost for a nominal execution of the platform for the equivalent of the Customer was $592,000$. The cost to the Customer in this work is the cost of signing the allocation which is about 4 times less at $137,281$. This reduction is possible due to the use of Pulsar to handle the majority of market interactions, minimizing contract calls. The smart contract functions could be optimized in the future to further reduce costs.

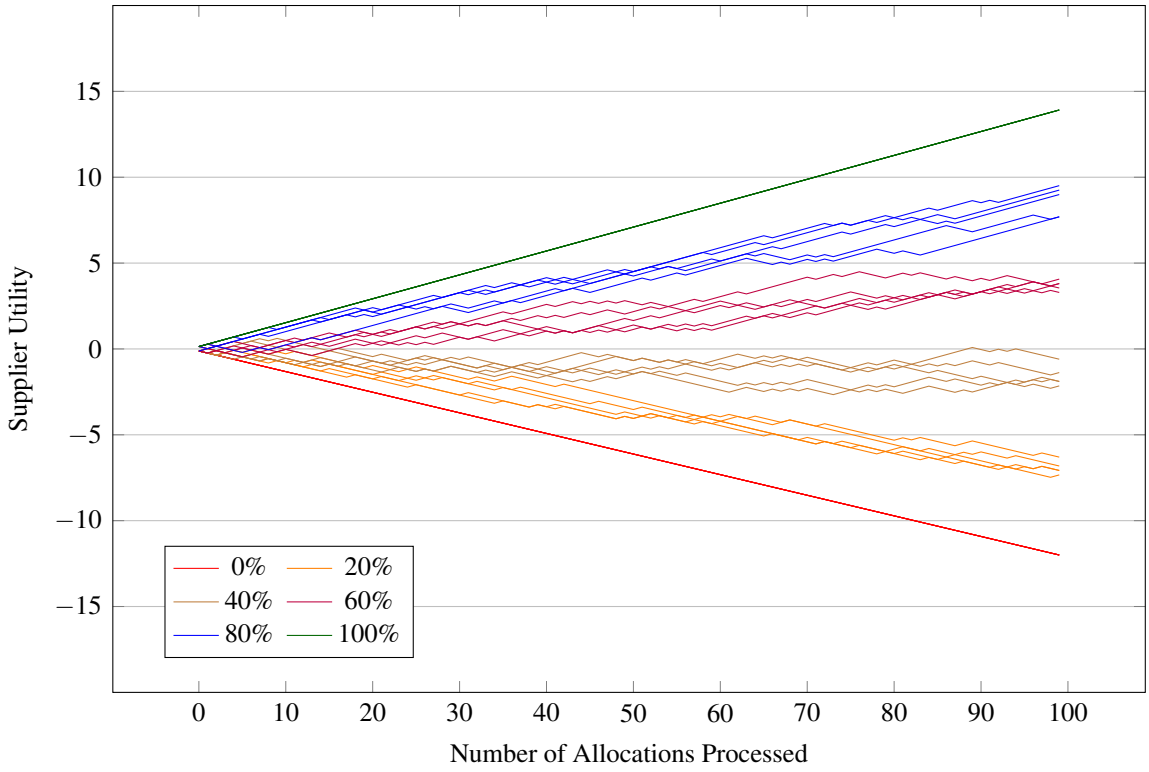


Figure 4.5: Cumulative supplier utility compared to the number of allocations completed by that supplier. Suppliers are broken into six groups representing the probability of the supplier successfully processing the inputs of a given allocation $P(s)$ (0% in the figure represents cheating 100% of the time). Each group consists of 5 suppliers. As shown, processing all allocations maximizes profit over time.

4.7.2 Strategy Experiments

The goal of this experiment was to validate the conclusions determined from the analytical analysis of the game by empirically checking that correct behavior resulted in maximum utility. To perform this we chose a set of parameters to compute values for the payments. These can be seen in Section 4.7.2. We set up the Supplier and the Customer so that we could configure their strategies. Specifically, the Suppliers were given a parameter representing the probability that the Supplier would properly process all the service inputs (i.e. did not cheat), $P(s)$. The Suppliers were grouped by $P(s)$ into 5 groups with $P(s)$ set to 0, 20%, 40%, 60%, 80%, and 100% respectively. Similarly, the Customers were given a parameter representing the probability that the Customer would supply the required outputs

$P(c)$. For the 30 Customers, we similarly broke them into five groups with $p(c)$ set to 0, 20%, 40%, 60%, 80%, and 100% respectively.

We ran the experiment with 30 Suppliers and 30 Customers. Each of the Customers and Suppliers sent 100 offers on the offers channel, where they were read by the Allocator and matched. The Allocator wrote the resulting allocation to the allocation channel. Once the allocation was sent the participants executed their strategies by writing to the output channel what their strategy was. This was read by a script that was emulating the Verifier and Mediator. Based on the strategies taken by the Supplier and Customers, the Verifier/Mediator would calculate the corresponding payments according to the game outlined in Section 4.5 and write to a channel representing the blockchain that the Customers and Suppliers would read to update their balance.

As shown in Fig. 4.5, the Supplier utility was maximized when the Suppliers processed all inputs and did not cheat. The utility was least when the Suppliers cheated every time and improved as the probability of not cheating increased. There was minimal variance between Suppliers with the same $P(s)$, showing that the behavior of the Customers had a negligible impact on the Supplier's utility. Therefore, it is always in the Supplier's best interest to process the service inputs.

The Customer Utility is dependent on the behavior of the Suppliers, so to investigate the impacts of Supplier behavior on Customers, we ran nine experiments. Each of these experiments involved a set of Suppliers with a fixed $P(s)$. The first experiment therefore had all Suppliers with a $P(s) = 60\%$ and the last experiment had a $P(s) = 100\%$. For each of the experiments, we recorded the final utility of the customer that submitted n outputs for verification and a customer that did not submit n outputs for verification.

As shown in Fig. 4.6, it is always in the best interest of the Customer to submit the required n outputs for verification regardless of the system level $P(s)$. In the case of a perfect system with $P(s) = 100\%$, there was a 24% greater utility for submitting outputs for verification. Additionally, there is a clear linear relationship between the $P(s)$ of the Sup-

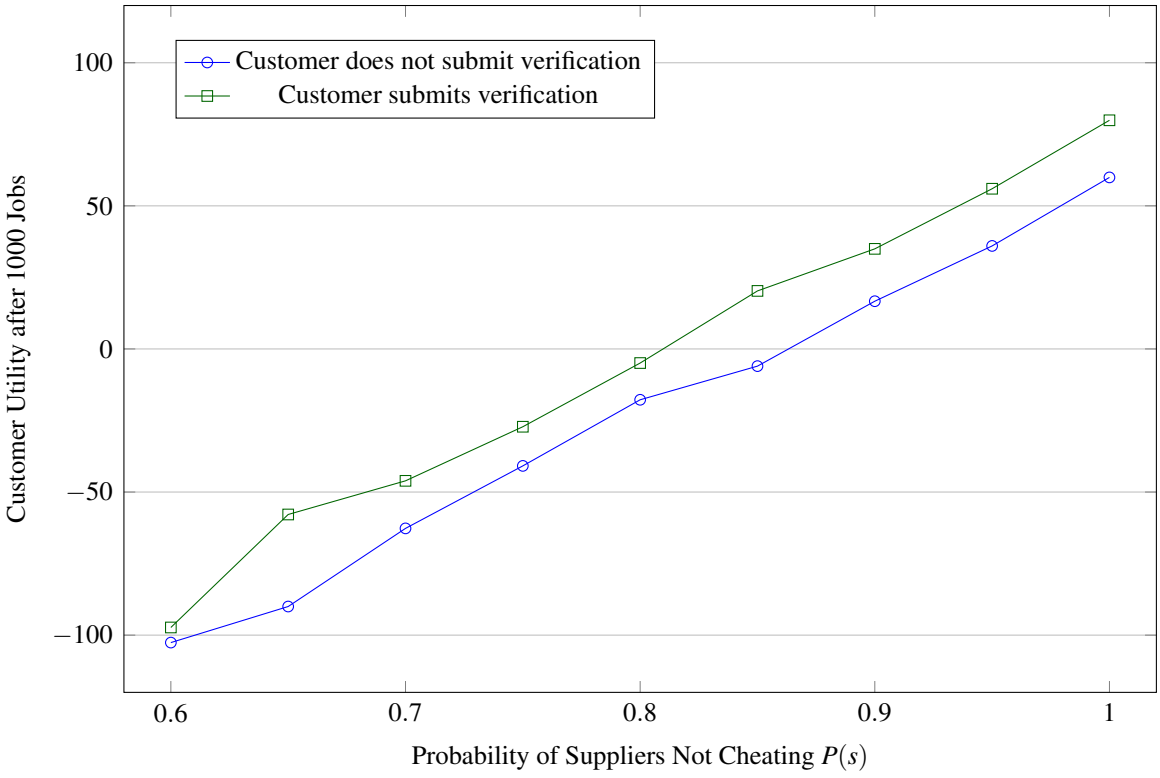


Figure 4.6: Customer utility after 1000 allocations compared to the probability of suppliers in the market servicing a given allocation ($P(s)$). The customer gets the maximum benefit when submitting the required number of verification values. As shown, there is a linear relationship between the global $P(s)$ of the suppliers and customer utility.

pliers in the system and utility for the Customer. For the ARTA application as formulated in Section 4.5, the Customer had a break-even point when Suppliers in the system had a $P(s)$ of 82%.

From this experiment, we see that the rational strategy for the Customer is to verify n outputs and the Supplier to process all inputs. Allowing the participants to trust the system. This was possible because of the protocol and incentive design decisions.

4.7.3 End to End Execution Experiments

The goal of this experiment was to monitor the performance of the system under load. We ran an experiment with 20 customers and 30 suppliers and monitored the time between

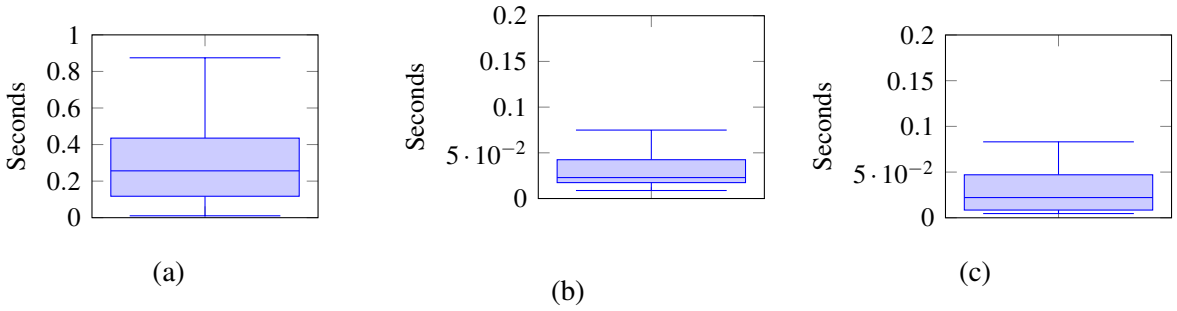


Figure 4.7: Message delay (seconds) at runtime for experiments with 20 Customers and 30 Suppliers. Fig. 4.7a: distribution of time between a Supplier submitting an offer and receiving an allocation (Supplier-1 to Supplier-3 in Fig. 4.2). Fig. 4.7b: distribution of time between customer writing to input channel and receiving a processed result from the output channel (Customer-1 to Customer-6 in Fig. 4.2). Fig. 4.7c: distribution of time between the Mediator reading the verification outcome and mediator writing to verifier (Mediator-10 to Mediator-11). A total of 76,990 messages were processed during the experiment.

messages exchanged within the system. All customers ran the ARTA application with parameters as outlined in Section 4.7.2 and the results are provided in Fig. 4.7. In total there were 76,990 messages exchanged throughout the experiment. Using Fig. 4.2 to visualize the data flow; Fig. 4.7b shows the delay between a Customer writing to an input channel ① and reading the processed result from the output channel ⑥. The median time for this process was 0.029 seconds, showing that customers can expect a reasonably fast processing time for the ARTA application.

Fig. 4.7a shows the delay between a supplier submitting an offer on the offers topic and receiving an allocation. This process can be visualized in Fig. 4.2 as the time between a Supplier writing to the offer channel ① and reading the allocation channel ③. The median time for this process was 0.38 seconds showing there was minimal delay in matching offers. Fig. 4.7c shows the delay between the mediator reading the verification outcome and the mediator writing to the verifier. The median time for this process was 0.033 seconds, showing that mediation is a reasonably fast process and does not add significant overhead to the system.

4.8 Related Work

With the proliferation of large-scale IoT systems, publish-subscribe and stream processing software have rapidly advanced in recent years. In this context, a data stream is a large unbounded collection of time-stamped messages that are published and consumed on a message topic. Modern low-latency publish-subscribe middleware such as Apache Kafka [179] and Apache Pulsar [145] use brokers and partitioned logs used to reliably process large scale data in near real-time. Kafka is used at LinkedIn to process peak loads of 200,000 messages a second and deliver over 55 million messages a day to users [180]. Many of the applications built on these architectures rely on processing raw data streams, including event detection systems, video analysis applications and, analytics. Stream processing engines such as Heron [181], Apache Storm [182], and Apache Spark Streaming [183] provide unified APIs and protocols for building stream processing applications.

Modern pub-sub and streaming frameworks have made it more accessible to design robust streaming applications. Typically these systems are deployed on the cloud, and their architectures assume that the servers running these applications are trustworthy. This presents unique challenges in offloading computation to mistrusted entities. Market-driven approaches have been studied in the context of residual cloud computing [184] and batch processing [163]. However, despite the potential benefits of creating an open decentralized market for outsourcing stream processing, there has been limited research in this area. Cherniack et. al. [185] outlined a federated market for which producers and consumers derive value from streaming data, however they do not address trust in their design. TrueBit [123] is a platform designed to extend the computation capabilities of blockchain-based consensus computers (such as Ethereum) which provide strong guarantees that small computations are performed correctly. They do this by constructing a market and outsourcing computational tasks to Solvers, employing game-theoretic techniques. However, this platform is not suitable for stream computing because it relies on the Solvers sending their results to the blockchain before they are released. Mutable [186] is a cloud platform whose

resources are aggregated from the surplus compute capacity of trusted entities like internet service providers, telecommunications operators, or urban data centers. Though similar in nature to this paper, aggregating surplus resources, they do not consider all the potential resources and the associated potential for misbehavior.

4.9 Conclusion and Future Work

In this paper, we developed an environment that can be used to implement decentralized market mechanisms. Using this environment we designed a market for outsourcing online computations. This market solution can provide the substrate for multi-stakeholder applications that rely on a dispersed computing paradigm, of which ARTA and multi-modal traffic monitoring are examples. This architecture allows participants to make trade-offs between speed and trust. By this, we mean that as soon as an allocation has been accepted the participants may begin speculatively working in order to work quickly. However, this work may be wasted because that allocation is never added to the blockchain. Alternatively, participants may be cautious and avoid beginning the service until the allocation has been recorded in the blockchain.

We demonstrated that rational participants will follow the protocol and benefit from participating in the system, while participants that deviate from the protocol incur fines. We do not prevent agents from operating maliciously and returning erroneous results, however, we do make it so that the costs exceed the benefits within the system. We do not handle scenarios when there are benefits that are external to the system that make it worthwhile to misbehave. Considering these scenarios is part of our future work.

Since there are many Allocators in the system, each with its own allocation algorithm, a multitude of Markets within the system can be formed. Exploring how these various markets interact is also an interesting avenue for future work.

Table 4.1: Key Symbols

Smart Contract (SC)	
ρ	penalty rate set by the contract
π_v	cost of Supplier submitting outputs to the Verifier.
π_{cc}	cost of Customer committing outputs to the Verifier
π_{mc}	cost of Mediator committing mediation results to the blockchain
Mediator (M)	
π_m	payout to the Mediator for being *available* for the duration of the service
$\pi_{v\varepsilon}$	Mediator's electricity cost to verify outputs
Customer	
π_{xmax}	amount the Customer is willing to pay per million instructions
I	number of instructions (in millions) required to process a service input
b	benefit that the Customer obtains from service output
π_{cg}	Customer's cost of generating an output
e_c	$= \frac{\pi_{cg}}{\pi_s}$ Customer's efficiency of processing vs. the price paid to outsource
λ	rate at which sensor data is generated (these are the inputs to be processed in an allocation)
r	number of replicas requested by the Customer
s_c	Customer choosing to process an input
s_v	Customer choosing to verify an output
Supplier	
π_{xmin}	payment that the Supplier requires per million instructions
$\pi_{s\varepsilon}$	cost to process a service input
π_v	cost to send output hash to the Verifier
$P(s)$	probability that the Supplier will process a particular input
e_s	$= \frac{\pi_{s\varepsilon}}{\pi_s}$: Supplier's efficiency of processing vs. the price paid to outsource
Allocator	
π_a	cost to pay Allocator and for signing the allocation
π_A	payout to the Allocator for providing an accepted allocation
Δ	$= a_{end} - a_{start}$: duration of a service allocation
π_x	market price per million instructions between π_{xmin} and π_{xmax} (determined by the Allocator)
π_s	$= \pi_x \times I$: amount to be charged/paid to a Customer/Supplier for a processed input
n	number of outputs that must be provided by the Customer for verification
λ_Δ	$= \lambda \times \Delta$: total number of inputs to be processed by each Supplier replica during an allocation
π_{cd}	Customer's security deposit for collateral prior to transaction (set to $\rho\pi_s$)
π_{sd}	Supplier's security deposit for collateral prior to transaction (set to $\rho\pi_s$)

Table 4.2: The utility of the Customer and Supplier given the 4 combinations of their pure strategies.

Supplier Customer	s_c	$\overline{s_c}$
s	$\pi_s - \pi_{s\varepsilon} - \pi_v - \pi_a^*$ $b - \pi_s - \pi_{s\varepsilon} - \pi_v - \pi_a$	$\pi_s - \pi_{s\varepsilon} - \pi_v - \pi_a$ $b - \pi_s - \pi_a^*$
$\overline{s}$	$-\pi_v - \pi_d - \pi_a$ $b - \pi_s - \pi_{s\varepsilon} - \pi_v - \pi_a^*$	$\pi_s - \pi_v - \pi_a^*$ $-\pi_a - \pi_s$

Table 4.3: The utility of the Customer and Supplier given the 4 combinations of their pure strategies after the Customer's threat to check n outputs

Supplier Customer	s_c	$\overline{s_c}$
s	$\lambda_\Delta(\pi_s - \pi_{s\varepsilon}) - \pi_v - \pi_a^*$ $\lambda_\Delta(b - \pi_s) - n\pi_{cg} - \pi_{cc} - \pi_a^*$	$\lambda_\Delta(\pi_s - \pi_{s\varepsilon}) - \pi_v - \pi_a$ $\lambda_\Delta(b - \pi_s) - \pi_{cc} - \pi_{cd} - \pi_a$
$\overline{s}$	$-\pi_v - \pi_{sd} - \pi_a$ $-\lambda_\Delta \pi_s - n\pi_{cg} - \pi_{cc} - \pi_a$	$-\pi_v - \pi_{sd} - \pi_a$ $-\lambda_\Delta \pi_s - \pi_{cc} - \pi_{cd} - \pi_a$

	o_1	o_2	o_3	o_4
	ss_c	$\overline{s}s_c$	$s\overline{s_c}$	$\overline{ss_c}$
Customer	$\lambda_\Delta(b-\pi_s) - n\pi_{cg} - \pi_{cc} - \pi_a$	$-\lambda_\Delta \pi_s - n\pi_{cg} - \pi_{cc} - \pi_a$	$\lambda_\Delta(b-\pi_s) - \pi_{cc} - \pi_{cd} - \pi_a$	$-\lambda_\Delta \pi_s - \pi_{cc} - \pi_{cd} - \pi_a$
Supplier	$\lambda_\Delta(\pi_s - \pi_{s\varepsilon}) - \pi_v - \pi_a$	$-\pi_v - \pi_{sd} - \pi_a$	$\lambda_\Delta(\pi_s - \pi_{s\varepsilon}) - \pi_v - \pi_a$	$-\pi_v - \pi_{sd} - \pi_a$
Allocator	π_A	π_A	π_A	π_A
Mediator	π_m	$\pi_m - n(\pi_{s\varepsilon} - \pi_{v\varepsilon} + \pi_s) - \pi_{mc}$	$\pi_m - n(\pi_{s\varepsilon} - \pi_{v\varepsilon} + \pi_s) - \pi_{mc}$	$\pi_m - n(\pi_{s\varepsilon} - \pi_{v\varepsilon} + \pi_s) - \pi_{mc}$

Table 4.4: Game outcomes and payments in Fig. 4.4. For example, o_2 is the outcome when the Supplier does not process all the validation inputs correctly and the customer does provide sufficient validation inputs.

			Tenants $(t_1, t_2, \cdots, t_n)$			
			Customer	Supplier	Allocator	Mediator
Cluster $(c_1, c_2, \cdots, c_n)$	Customer	MC	rw	r	r	r
		SC	rw	r	-	r
	Supplier	MC	r	rw	r	r
		SC	-	rw	-	r
	Allocator	MC	r	r	rw	r
	Mediator	MC	r	r	r	rw
		SC	r	r	r	rw

Table 4.5: Tenants represent stakeholders and can read and write to their own cluster of hardware. To participate in the market they allow read access to their *Marketplace Cluster* (MC). To participate in a service they allow read access for allocated stakeholders to their *Service Cluster* (SC)

Function Name	Approx. Gas Cost	Approx USD ($) value
Constructor	4054778	33.43
Setup	227554	1.88
CreateAllocation	369506	3.05
AddSupplier	181139	1.49
MediatorSign	101853	0.84
CustomerSign	137281	1.13
SupplierSign	188145	1.55
PostOutput	130108	1.07
ClearMarket	157458	1.30

Table 4.6: Gas Costs and USD value of each smart contract function call. Assuming 20 Gwei per 1 unit of gas.

Table 4.7: Test Parameters. The majority of these parameters were created to model the game (Table 4.4) and are defined in Table 4.1. The others are as follows. To convert gas to dollars we used [169]. Electricity cost per kWh E was acquired from [177]. For the device thermal design power (TDP), we used the specification of a Beagle Bone Black single-board computer[178].

Parameter	**Test Value**	**Description**
λ	20	frames per second
Δ	10	minutes
λ_Δ	12000	$\lambda \times \Delta$
n	12	.1% of λ_Δ
ρ	1000	Eq. (4.6)
$gasprice$	20 Gwei	price per unit of gas
π_{cc}	1.1257	$137,281$ gas at $gasprice$ to dollars
π_v	1.0669	$130,108$ gas at $gasprice$ to dollars
E	13.2	cents per kWh
TDP	4.6e-4	Thermal Design Power (kW) at max
TDP_{min}	2.1e-4	Thermal Design Power (kW) at min
$\pi_{a\varepsilon}$	0.001	$E \times TDP \times \Delta$
π_a	3.02994	$\pi_{a\varepsilon} + 369,506$ gas at $gasprice$ to dollars
π_{cg}	7e-8	$(E \times TDP \times n)/\lambda_\Delta$
π_{se}	7e-8	$(E \times TDP)/\lambda_\Delta$
π_s	2.0e-5	$b > \pi_s > \pi_{se}$
b	3.5e-5	Table 4.4
π_{cd}	0.02	$\pi_s \times \rho$
π_{sd}	0.02	$\pi_s \times \rho$
π_m	3.3e-5	$E \times TDP_{min} \times \Delta$

Chapter 5

Generalizing the Solution for Multi-Stakeholder Systems - Connecting the Stakeholders

5.1 Overview

MSCPS are subject to dynamism, heterogeneity, and increased failure potential since they typically do not operate in data centers or other controlled environments. These challenges are not exclusive to MSCPS and in fact occur in many distributed systems, though they are exacerbated in MSCPS since various stakeholders may be developing the devices that will coexist within the system. Since these problems are common across applications, middleware frameworks and platforms are created to facilitate high-level application implementation. These platforms can help to abstract away high complexity details associated with "infrastructure protocols" of the heterogeneous devices requiring low-level communication via drivers and distributed computing. The frameworks are often used to aid in development and deployment, and in coordinating and controlling the computation done on distributed computing nodes [187].

The RIAPS middleware provides a robust communication and integration layer, allowing integration with heterogeneous devices. This capability is demonstrated through a traffic controller case study. In this chapter, we discuss RIAPS and how it supports MSCPS applications.

The work comprising this chapter has been published in the IEEE 20th International Symposium on Real-Time Distributed Computing (ISORC) [38].

- S. Eisele, I. Mardari, A. Dubey, and G. Karsai, "Riaps: Resilient information architecture platform for decentralized smart systems," in 2017 IEEE 20th International Symposium on Real-Time Distributed Computing (ISORC). Toronto, ON, Canada: IEEE, May 2017, pp. 125–132.

Emerging Trends: The emerging Fog Computing paradigm provides an additional computational layer consisting of distributed computation and communication resources that can be used to monitor and control physical phenomena close to the source. It can also be used for fine-grained data collection and filtering before sending the data to a cloud service. Examples of these Fog Computation platforms include SCALE [188] and Paradrop[189]. However, while, the concept of Fog Computing is promising, a number of challenges exist that must be addressed. One of the foremost challenges is providing a stable environment for application development and deployment despite the dynamism, heterogeneity, and increased failure potential of computing resources at the edge which do not operate in data centers or some other controlled environment.

A solution to this problem is a universal computing platform, which provides the core services necessary for a stable deployment environment. Services like time synchronization, distributed data management and coordination, service discovery, and mechanisms to deploy and remotely manage the distributed applications.

RIAPS: Our team is developing the core architecture, algorithms, and programming paradigms for such a computing platform called RIAPS (Resilient Information Architecture Platform for Smart Systems) [190]. The pivotal concept of Smart Applications is the distribution of intelligence throughout the infrastructure. For example, in the smart grid domain, increasingly companies, communities, and even some customers (or prosumers) are becoming managers of power. This requires monitoring, control, and management of software applications at all levels to do their work. The centralized, control-room oriented paradigm is not sustainable, as it does not scale. Rather, a *decentralized* paradigm is required where interacting software programs deployed on devices across the network solve problems collaboratively. This is also true for distributed traffic control where each intersection controller must coordinate with other controllers based on contextual and local information [191].

Innovation: This paradigm is very different from what is being used today. In today's systems, data is collected locally and transferred to a central server or control room where control decisions are made and control commands are generated. These commands are then sent back to local controllers and actuators. This architecture incurs long round-trip times, delayed decisions, and does not lend itself to the needs of future edge applications [192] like energy management [193]. The distinguishing characteristic of RIAPS is the completely decentralized computing model: software applications are distributed across a multitude of compute nodes on a communication network, and each node has access to local measurements and actuators. An application consists of components that run in parallel on a collection of nodes. The functionality of an application is realized by the network of interacting components managed by actors. This computation architecture is an extension of the F6COM computation model [194] and [195]. The specific extensions are related to the discovery services and the platform services that we discuss in the next section.

Contributions: The contribution of this paper is the architectural description of RIAPS (section 5.3), a demonstration with a development version of the platform implementing a traffic control example (section 5.3.2) showing some results comparing the effectiveness of traffic control with distributed coordination compared to no coordination. We also briefly discuss a microgrid application example on the platform. Using these examples we motivate the need for a robust decentralized discovery service (section 5.3.4) as a critical service for the resilience of the platform. Thereafter, we discuss the design and implementation of the discovery service using a distributed hash table (section 5.4). We finally present experimental results (section 5.5) on some discovery service metrics and compare our work with the state of the art (section 5.6).

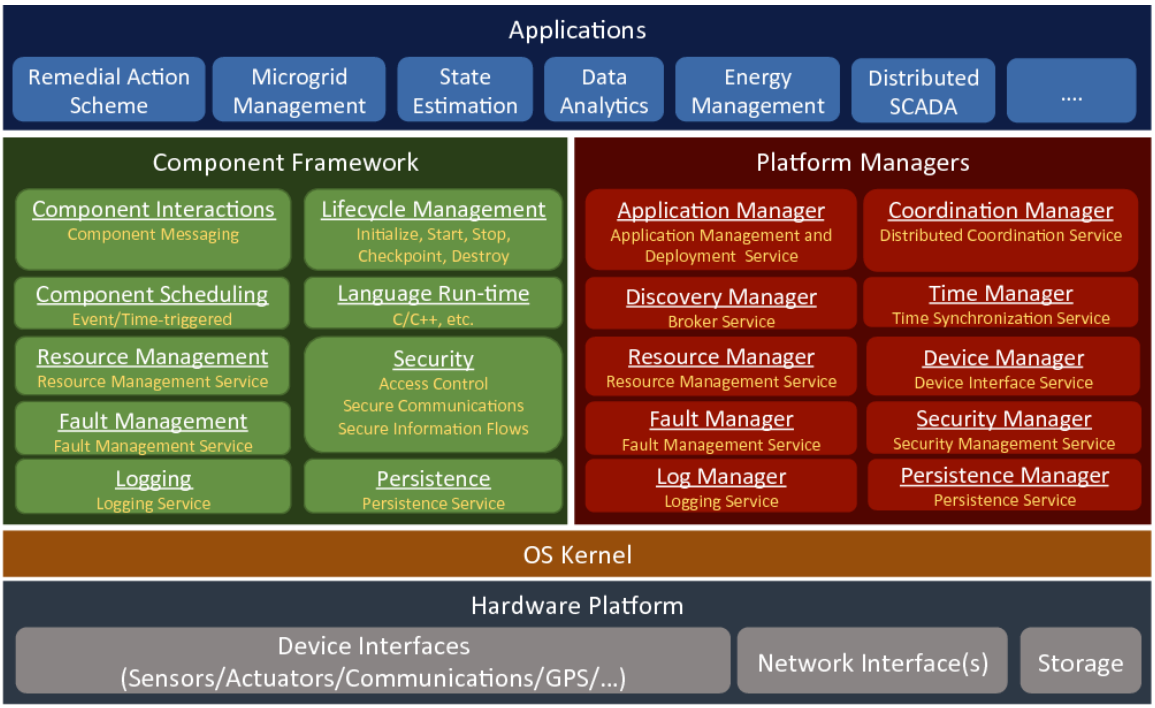


Figure 5.1: RIAPS Run-Time System Architecture

5.3 The RIAPS Computation Architecture

The goal of the RIAPS run-time system (see Fig 5.1) is to provide a software foundation for building distributed applications. It relies on an underlying operating system and includes two major ingredients: (1) a Component Framework, and (2) a suite of Platform Managers. The Component Framework is instantiated as a set of software libraries that are (dynamically) linked with the application components, while the Platform Managers are specialized operating system processes, implemented as daemons in the Linux systems. These two ingredients provide the services that will be used by a developer in supporting the implementation of the application logic. The Component Framework layer is where the implementation of the various middleware libraries reside. The goal of the Component Framework is to provide higher-level abstractions for building complex, resilient, distributed applications on the platform. The middleware libraries include the component scheduler (which implements the component execution semantics), the component interaction library (that enables publish/subscribe, and remote method invocation on the same

node or across the network. Having formal interaction semantics provide additional reasoning capabilities as shown in [196]). Furthermore, the framework provides support for lifecycle management support (that assists in remotely managing the software components), the language run-time libraries, the resource management support (to monitor computing platform resource utilization/availability), the fault management support (that detects and mitigates anomalies in software components), the security library (for secure communication), the logging library (to record component events), and the persistence library (to allow the persistent storage of data). These libraries are linked with the components used to create an application.

The Platform Managers layer includes the elements of the application framework: the various platform services that run as independent processes and implement system-level management capabilities. The services include the Application Manager (that enables remote installation and management of the applications), the Distributed Coordination Manager (that implements fault-tolerant distributed service like leader election, consensus, coordinated actions, etc.), the Discovery Manager (which determines available connections among components on the same node or other operating nodes), the Time Manager (that provides high-precision timing and time synchronization services), the Resource Manager (monitors computing resources to ensure components and Platform Managers are able to run concurrently), the Fault Manager (that provides node-level fault management services), the Device Manager (that supports access to and management of attached input/output devices), the Security Manager (that handles authentication and manages keys and digital signatures), the Log Manager (that serves as a single entry point to all log activity on a node) and the Persistence Manager (that provides non-volatile data storage facility).

The applications reside in the top layer (see figure 5.1) and they rely on the services provided by the Component Framework and Platform Managers. One application consists of one or more application managers, called *actors*, which are deployed on computing nodes. Each actor hosts one or more application components that interact solely through

the middleware interactions and rely on the available platform services. The advantage of packaging multiple components into one actor is that the cost of communication between components in one actor is much smaller than across actors running on the same node. The communication between actors running on different nodes is even more costly, as the messages have to go through a complex protocol stack and a (potentially unreliable) network.

5.3.1 Component Architecture

A RIAPS component is a reusable unit of software that implements a set of operations for manipulating its state, and ports through which it communicates and interacts with other components. A special port, called the timer port is also available. It enables time-based triggering of the component. The timing of the RIAPS component is controlled by the Time Manager service that provides high-precision timing and time synchronization services. This service is not fully implemented yet and will be discussed in future work.

The operation of a component is analogous to a typical computer process in the sense that each component is limited to a single thread of computation. This thread is managed by a trigger method that is provided by the developer of the component. The trigger method monitors the state of the component and launches operations when 1) the state of the ports change, 2) a timer expires, or 3) an operation is completed. These operations implement the application logic of the component. The ports on the component are determined by the desired communication patterns which include asynchronous request/response, synchronous client/server, and publish/subscribe. Ports are assigned a message type and when an application is deployed, the message types represent the services provided or requested by the corresponding port. A special component, called the device component has the same attributes as application components however it may have multiple threads of execution to handle interfaces to physical devices.

To run an application the components are deployed on computation nodes. The compo-

nents on a particular compute node are managed by actors. An actor provides its components with the run-time code as well as the interfaces necessary to access platform services. Additionally, the actor provides the capabilities to control and configure its components remotely. This is required to ensure that all the components of an application can be installed and configured correctly. The actor is responsible for loading a component, setting up its configuration, and initializing its state.

5.3.2 Traffic Controller Example

In order to experiment with the RIAPS framework, we developed a traffic controller example. The example involves a city simulation where the traffic lights in each intersection are controlled by a traffic controller application implemented with the RIAPS platform running on embedded single board computers. The simulation sends simulated ”sensor data” to the intersection controllers consisting of the traffic density for the incoming road segments, as well as the current state of the traffic lights. Each intersection controller shares this information with its neighboring controllers, and each uses the information to estimate the traffic incoming on each segment. This information is used to change the state of the traffic lights to improve the flow of traffic.

The testbed for this example can be seen in Figure 5.2. It consists of a 32 BeagleBone Black [197] cluster connected through an Ethernet switch to a computer running Cities: Skylines [198]. This game was chosen because it can simulate the movements of hundreds of thousands of citizens, and it has a rich game modification API with an active community. This allowed us to modify the game to be able to control the traffic lights with our embedded controllers.

The RIAPS application created for this test scenario includes an intersection controller, a light interface device, and a density sensor device whose implementation can be seen in listings 5.1, 5.2, and 5.3 respectively.

```
// Intersection Controller component.
```

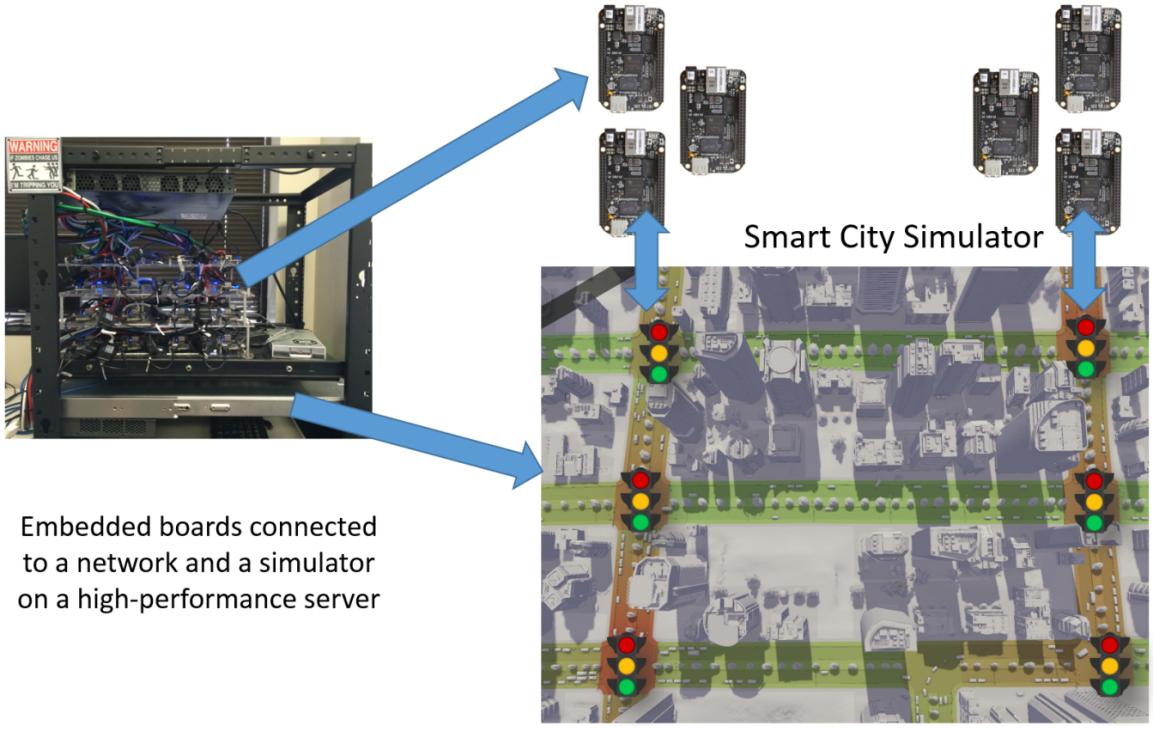


Figure 5.2: Testbed

```
component IC (parent="none"){
  timer clock 1000;
  sub densityPort : gameDensityMsg;
  sub lightPort : gameLightStateMsg;
  pub pubICPort : ICDensityMsg;
  sub subICPort : ICDensityMsg;
  req setLightsPort : (setLightReq, setLightRep);
}
```

Listing 5.1: Intersection Controller Component

```
device LightIF(rate=10, gameServerIP="host", parent="none") {
  inside trigger /* default */;
  timer clock 1000;
  pub lightPort : gameLightStateMsg;
  rep setLightsPort : (setLightReq, setLightRep);
```

```
}
```

Listing 5.2: Interface to Light device

```
device DensitySensor(rate=10, gameServerIP="host", parent="none") {
  inside trigger /* default */;
  timer clock 1000;
  pub densityPort : gameDensityMsg;
}
```

Listing 5.3: Interface to Density sensor device

The controller has 3 subscriber ports, a publisher port, a request port, and a timer. Two of the subscribers, *lightPort* and *densityPort* are for reading sensor data from the game. The *lightPort* has the message type of gameLightStateMsg and is connected to the publisher port with the corresponding message type in the light interface device when the application is deployed. The *densityPort* is essentially equivalent. Each intersection controller publishes the sensed density data at intervals determined by the firing of the timer and subscribes to the density data published by the adjacent intersections. The controller implemented in this example is fairly simple in that the state switching occurs based on thresholds. The thresholds ensure that a light remains in a particular state for some minimum time but no longer than some maximum time, similarly there are minimum and maximum density thresholds.

These components and devices of the application are contained in an *actor*. The actor is responsible for managing the components and devices and providing interfaces to platform services. As can be seen in listing 5.4 the actor contains both devices and a controller and is used to specify several parameters such as the IP address of the simulation machine. The actor registers the ports of the components with the Discovery Manager and if a matching message type is already known to the Discovery Service the client will retrieve that information and return it to the actor, which then sets up the connections between corresponding components.

```
actor Actor0 {
  local gameDensityMsg, gameLightStateMsg, setLightReq, setLightRep;//
     Local message types
  {
    ic : IC(parent="Actor0");// Intersection Controller
    LIF : LightIF(rate=2, gameServerIP="192.168.0.107", parent="Actor0")
       ;// Light interface device
    dsnsr : DensitySensor(rate=2, gameServerIP="192.168.0.107", parent="
       Actor0");// Density sensor device
  }
}
```

Listing 5.4: Actor

5.3.2.1 Experimental Results

The traffic controller implementation was run comparing the densities of the segments when running the controller with 1)only timer switching logic, 2)each controller checking its own density data and 3)each controller sharing data with its neighbors. The average segment densities for these tests can be seen in Figure 5.3.

We see from these initial experiments that monitoring intersection density is useful to decrease the segment density and sharing that information improves the situation slightly. In this study, the densities were collected from the game by querying the road segments surrounding the traffic lights. In an actual implementation, this will not work unless some sensor is installed at each road segment. Another option to obtain the data is for the cars themselves to publish their positions and routes to the cars around them and that information is then shared with the intersection controller. This way the controller does not have to guess how much traffic is coming on each of its segments. In addition, if vehicles are publishing their data, emergency vehicles may also publish this information, and controllers switch to prioritize the emergency vehicle. For such a system to be realistic the vehicles

	20s Timer			
	IC0	IC3	IC2	IC1
Seg0(S) means:	21.31	16.37	11.62	30.89
Seg1(N) means:	17.23	15	8.41	24.22
Seg2(W) means:	14.84	13.3	16.09	12.33
Seg3(E) means:	15.54	10.06	18.65	24.77
Intersection avg	17.23	13.68	13.69	23.05
	Densities			
	IC0	IC3	IC2	IC1
Seg0(S) means:	15.69	15.93	10.84	20.09
Seg1(N) means:	15.47	11.81	9.6	18.74
Seg2(W) means:	11.15	9.31	11.22	8.43
Seg3(E) means:	8.89	6.2	8.44	16.72
Intersection avg	12.8	10.81	10.03	15.99
	Share Data			
	IC0	IC3	IC2	IC1
Seg0(S) means:	19.09	18.38	11.78	22.47
Seg1(N) means:	14.12	12.59	7.59	16.61
Seg2(W) means:	9.39	5.68	7.04	8.6
Seg3(E) means:	10.28	6	9.06	15.23
Intersection avg	13.22	10.66	8.87	15.73

Figure 5.3: Mean traffic densities for each segment of each intersection controller.

and lights must be aware of each other. This motivates the need for a discovery service that can quickly capture and share ingress and egress information to intersection controllers as vehicles come and go.

5.3.3 Microgrid Example

Another application for the RIAPS platform that is currently under development is a decentralized controller for microgrids. As power requirements change or faults occur segments of a connected grid may break off and become islands. This information must propagate quickly to the controllers to ensure smooth operation and continuous service. Similarly, when the islands re-connect to the main grid the components must discover each other and share resource information. The objective is to handle the transients between these states in clusters of controllers rather than at a centralized location as this will improve scalability as well as fault tolerance.

5.3.4 Requirements for the Discovery Service

In both use cases, the set of member nodes can change over time. For example, in a microgrid application, homeowners can choose to disconnect themselves from their local photo-voltaic grid and transfer themselves to the main utility grid. Similarly, in the traffic controller example, the lights can get disconnected due to failures. Furthermore, the same system can be extended to create a traffic priority system, where the emergency vehicles entering an area can communicate with the controller and can disengage when they exit the area. Given these two use cases it is easy to see that the network of communicating entities must be able to (a) know when new nodes join the group and (b) know when nodes leave the group. Furthermore, they must know when applications (and their components) come and go - the Discovery Service is expected to keep track of the state of the applications' services and message types. This service has to be distributed and fault-tolerant. A centralized implementation is insufficient, as it does not scale and it can be a single point of failure. Fault tolerance is needed as any node or communication link can fail unexpectedly. These local failures must not result in system-wide collapse. Hence, the Discovery Service must be available on each node, and these instances need to share their state - as needed - across the network.

5.4 Discovery Service

RIAPS aims to provide modular, decentralized solutions for each service comprising the platform, so they can be used in other applications. Therefore, the *Discovery Service* runs on each node as an independent process and listens for messages from the local RIAPS applications, and other nodes with a Discovery Service. Figure 5.4 shows the main features of the service discovery: RIAPS applications register app services by providing the related details to the discovery service(message types, communication protocols, IP addresses, and ports). The Discovery Service stores the app service details and forwards the new

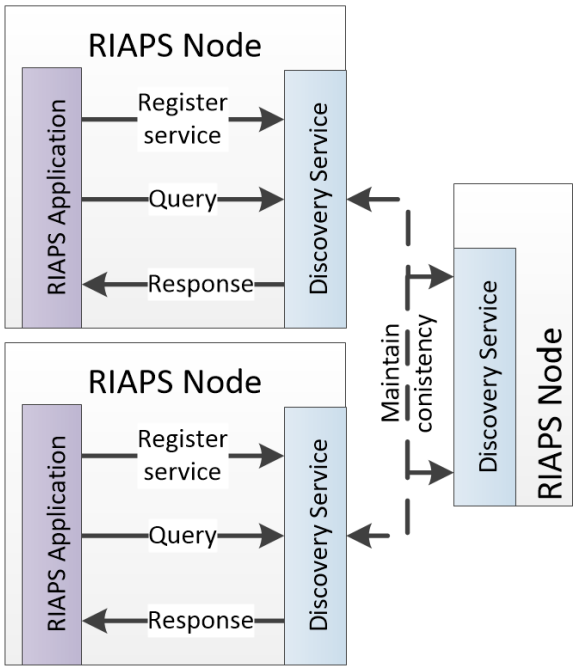


Figure 5.4: RIAPS Discovery Service Infrastructure

information to the neighboring nodes. When a RIAPS application requests an app service, it queries the local Discovery Service and the results are asynchronously sent to the RIAPS application.

The Discovery Service relies on OpenDHT [199] to store, query, and disseminate the service details through the network. OpenDHT is a fast, lightweight Distributed Hash Table (DHT) implementation. The dissemination does not mean full data replication on all nodes, OpenDHT stores the registered value locally and forwards it to a maximum of eight neighbors. Note that usage of distributed hash table for service discovery does not distinguish the nodes, (i.e. there are no "server" or "client" nodes) – nodes are peers and each operates with the same rules. If a node disconnects from the network, the Discovery Service on other nodes is still able to register new services or run queries. If a new node joins the cluster, the values stored in the network are available to the new node. For this approach to flexibly handle node ingress and egress the Discovery Services must find each other. Note that there are two major cases for network configuration: (i) the nodes are on

the same local subnet or (ii) the nodes are on different subnets of the network.

To find the available Discovery Service managers on the same local subnet the RIAPS framework uses UDP-beacons. Periodically, each Discovery Service instance announces (via IPv4 UDP broadcast) its network address and listens for incoming beacons. These UDP packets are sent and received asynchronously and the Discovery Service managers maintain the list of known addresses. Before a UDP packet is processed by the Discovery Service the received beacons are filtered to remove the non-RIAPS-specific UDP messages. These messages function as a heartbeat. If no messages are received from a known node during two time periods, then the Discovery Service removes the silent node from the list of peers. When a UDP beacon arrives from a new node the Discovery Service stores the address of the new node in OpenDHT, which then adds it to the list of known nodes.

Unfortunately, the nodes in another subnet cannot be discovered by UDP broadcasting; the remote addresses must be passed explicitly to the Discovery Service. In this case, we rely on designated gateways running the Discovery Service with IP addresses of the other subnet (assuming that routing is available between the subnets).

5.4.1 Handling the ingress and egress scenarios

In the previous section, we mentioned that the DHT-based service discovery forms 8 node clusters to share application service registration data, but we did not discuss how the stored data is used in RIAPS. When a RIAPS application starts, it registers its services in the Discovery Service. The Discovery Service stores this information in the DHT, and the DHT propagates the new information through the cluster.

In the startup phase, a RIAPS application not only announces the provided services but subscribes to needed services. If a compatible service is already in the DHT, the Discovery Service sends a notification to the requesting RIAPS application. The application processes the newly arrived notification and connects to the service. If the desired service is not available the Discovery service will issue a callback to the requesting application when it

becomes available.

OpenDHT does not provide an API to remove a service from the DHT. Instead, a service may be removed by setting an expiration value (the default being 10 minutes). After this time the service is removed from the DHT. It also means that value must be renewed periodically by the Discovery Service if the application is running. Therefore, when a service stops responding the manager does not remove it from the DHT until the current registration expires.

5.4.2 Fault Tolerance

The Discovery Service is responsible for renewing the registration of application services in the DHT. Renewal is necessary, as the stored values are otherwise removed. Before renewal, the Discovery Service must check that the RIAPS component service to be renewed is still running and available. This means that the Discovery Service must handle the case when an application service leaves the cluster abruptly, e.g. it stops without sending a message.

We are currently implementing the next version of the discovery service in which the service information is paired with the *Process ID* (PID) of the actor. Namely, when an application component registers a service then the PID of the parent actor is also registered with a time-stamp in the Discovery Service. The list of service/PID pairs are verified periodically by checking if the PID is still running. If the process has stopped, the Discovery Service removes the pair and does not renew the registration at the next DHT refresh point.

The components must be resilient as well, since the Discovery Service could stop unexpectedly. If the Discovery Service fails, the components and actors continue, but cannot receive notifications about new services, and new actors cannot be started (until the Discovery Service restarts).

Since the components are managed by actors, the components do not implement any discovery checking algorithm. The connection with the Discovery Service is maintained

by the actors. The approach for the actor checking Discovery Service liveness is the same as Discovery Service checking components. The actor knows the PID of the Discovery Service and maintains a time stamp. If the PID of the Discovery Service is not in the list of the running processes, the actor starts a re-initialization process. Reinitialization means, that the actor re-registers the running RIAPS services in a new Discovery Service instance and subscribes to the services needed. If the discovery service dies, the actors are informed and they re-register to recreate the state within the discovery service.

5.5 Tests for the Discovery Service

To test the discovery service we ran a few tests. The first was to initialize all but one node as subscribers. Once ready, the final node was added as a publisher service. This provides events for measuring service propagation time through the cluster. The node clocks are synchronized with NTP, making measurements across the cluster meaningful as their timestamps are within 300ms of each other. This test is relevant to the traffic light example because as vehicles move through the city they enter and exit new clusters which collectively transmit their density to the traffic controller allowing the controller to efficiently route traffic.

There are several measurements taken for this test. The first is the "Register service" step (s1) where each node informs the Discovery Service of which services it provides and which it requires. As mentioned this occurs first for all of the subscribers, then the publisher. The Discovery Service stores which service the subscriber is interested in, and when that service is available the Discovery service sends a message to the component's actor, this is the second measurement (s2). We assume the difference between this time and the publisher service registration time to be the time needed to propagate a new service through the DHT. The third time-stamp (s3) is when the actor notifies the corresponding component of the new service. The final time-stamp (s4) is when the subscriber is connected to the publisher. These events can be visualized using Figure 5.4 as a reference. The duration of

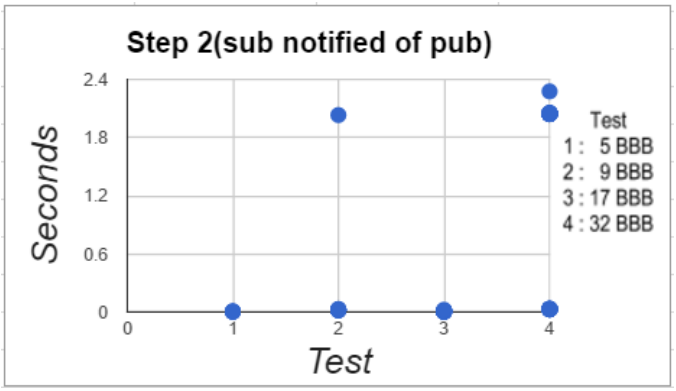


Figure 5.5: This figure shows the time between when the publisher registers with the Discovery service and the time when the subscribers receive notification of that service for the tests involving 5, 9, 17, and 32 BeagleBone Black. Several data points were very close to each other, and so overlap in the plot. The ranges of the overlapping nodes are Test 1: 4 nodes(1-7ms), Test 2: 7 nodes(18-32ms) 1 node(2.032s), Test 3: 16 nodes(2-23ms), Test 4 : 12 nodes(27-35ms), 18 nodes(2.041-2.056s), 1 node(2.276s)

steps 1 and 2 are the time stamp of the publisher registration and the subscriber s1 and s2 timestamps respectively. The duration of step 3 is s3-s2 and the duration of step 4 is s4-s3.

The test was run utilizing Zopkio [200]; a testing framework. We can see the result for step 2 in Figure 5.5. The time between the first subscriber registration and the publisher registration (step 1) is linear as nodes increase due to the implementation of deployment in Zopkio as a sequential process. Steps 3 and 4 take between 5 and 12ms.

As the accuracy of the clock between nodes is about 300ms the 5, 9, and 17 node tests are indistinguishable, however the jump in time for the 32 node experiment suggests further tests are needed to verify scalability. The times for steps 3 and 4 after notification are not relevant for this service propagation test and are of short duration as we see in the next test.

The second test consists of two nodes each requesting and providing a publisher service. This means that each node subscribes to itself and all others. The test is to have one node exit and later rejoin the cluster. This is to verify that nodes recover and provide services reliably after egress events. The result of this test is shown and described in Figure 5.6

From this test, we do see that when BBB2 rejoined all services were re-established.

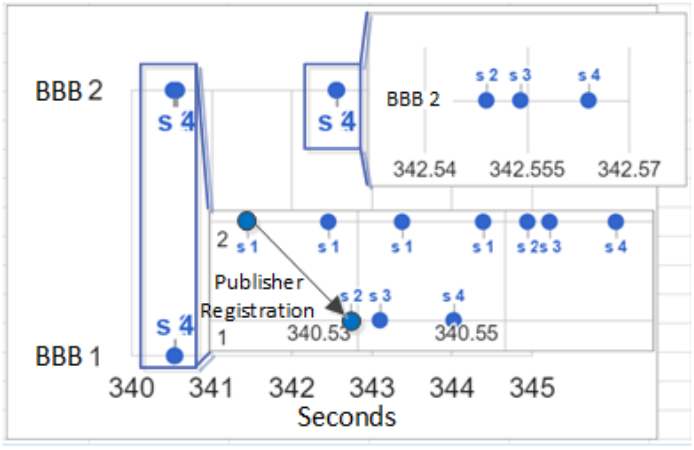


Figure 5.6: The plot here has two Beaglebones on the y-axis. The points denote the events (s1-s4) for each Beaglebone when BBB2 rejoins the cluster. The two zoom in boxes show the events in detail as they happen on timescales on the order of milliseconds. At the beginning of this test, two nodes were started. After some time BBB2 was rebooted. In the zoomed box between times 340 and 341, the first event shown is s1 for BBB2 when it registers its publisher service. The s2 event for BBB1 is when BBB1 receives the notification of BBB2's service 14ms after BBB2 registers it. 4 and 10ms later BBB1 notifies it's requesting component and that component connects to the publisher respectively. Towards the end of the first zoom-in-box, we see s2, 3, and 4 events when BBB2 receives notification of its own service. The second zoom-in-box between 342 and 343 seconds shows BBB2 again receiving notification of its own publisher. Covered by the second zoom in box at 344.5 seconds BBB2 finally receives notification of the publisher on BBB1. There is some behavior where a new node receives notification of the first service it discovers twice.

5.6 Related Literature

Resource discovery is a critical aspect of distributed applications. Zookeeper, which was originally developed to provide a distributed, eventually consistent hierarchical configuration store [201] has been adapted to serve as a resource discovery service. The infrastructure of Zookeeper includes sending notifications to clients, so service discovery has been implemented with it. However it is difficult to deploy and maintain, it also prioritizes consistency over availability [202]. This means that in a network with nodes joining and leaving new applications will be blocked waiting for resources while the service discovery waits for consistency before allocating resources. For highly dynamic systems, like the traffic example, a paradigm that prioritizes consistency is fundamentally flawed when at-

tempting service discovery. Responding to this need Apache released Apache Helix which resolves some problems [203] but the fundamental issues are still present.

Another tool developed specifically for resource discovery is Consul from Hashicorp [204] which is the industrial state-of-the-art. Consul provides several higher-level features such as health checking and distributed configuration management. It relies on servers to store and replicate the resource discovery data. It is recommended to have several consul servers for fault-tolerance. With several servers one is elected as a leader, using Raft-based consensus, in order to guarantee consistency between servers. This means that Consul too has the problems associated with prioritizing consistency over availability because in a dynamic system a device chosen as a server may leave resulting in a lack of quorum. To combat this, it is possible to make every node a server but in high device density, or resource-limited situations there are issues with data replication since every node will replicate the key/value store. An alternative is Serf, another tool from Hashicorp which is not as fully featured as Consul, but rather than using an always consistent model it has an eventually consistent model, prioritizing availability. Serf does not have a central server, making it more resilient. The problem with Serf is that it was developed for node discovery rather than service discovery and so would need extensions to provide the service discovery. In [202] the authors create a docker container for Serf calling it Serfnode and using it for service discovery.

Hoefling et al. [205] present some extensions to the C-DAX [206] middleware. C-DAX is a middleware developed to be a cyber-secure and scalable middleware for the power grid. The authors do not discuss the security aspects of C-DAX but rather reference papers demonstrating these features. Scalability in C-DAX is achieved using a cloud and broker-based publish/subscribe mechanism, making it more scalable than client/server patterns. The extensions presented by Hoefling are to address weaknesses in the C-DAX middleware with respect to interoperability with legacy applications such as SCADA, and low latency applications such as synchrophasor-based Real-Time State Estimation of Ac-

tive Distribution Networks (RTSE-ADN). SCADA relies on bidirectional communication, so the authors implement a new client which consists of both a publisher and a subscriber to communicate with IP-based applications like SCADA using a tunnel-adapters and virtual network interfaces. The clients communicate using the C-DAX middleware. For low latency applications, such as synchrophasor-based RTSE-ADN, the authors present a method of connecting publishers directly to subscribers, without a broker reducing network traffic and the number of network hops required. The problems with this approach are those that impact all cloud-based systems. As devices increase so does cloud traffic and latency. Therefore in the end edge computing will be necessary.

In our work on RIAPS, we are interested in similar issues regarding latency, however, rather than relying on cloud-based centralized databases and resolvers, we use a Distributed Hash Table (DHT) to track the participants in the network and have the nodes discover one another. The removal of the cloud allows us to achieve single-hop connections between publishers and subscribers as done in [205] but rather than needing to 1) send a join message to a Designated Node in the cloud which 2) queries the Resolver (the look-up) for the address of the topic-specific database, then 3) have the Designated node connect to the database and request the subscribers or publishers for the topic so that 5)the publisher or subscriber can update its connection rules we can simply 1)look up in our DHT Discovery Service for the message type we are interested in, 2)receive the address and 3)connect.

Data Distribution Service (DDS) is a "middleware protocol" and API open standard for data-centric connectivity" published by Object Management Group (OMG) [207]. There are many implementations of the DDS standard which vary by developer. To promote interoperability between DDS implementations OMG introduced the Real-Time Publish-Subscribe (RTPS) protocol which was designed for DDS. The main features of RTPS[208] include fault tolerance, plug-and-play connectivity (allowing for dynamic ingress and egress with automatic discovery), the capability to implement trade-offs between reliability and latency, scalability, modularity allowing constrained devices to run a subset of the standard[209]

and still communicate with the network, and type-safety to prevent mismatched endpoints from connecting. However, the tremendous complexity of the DDS discovery service due to the several QoS options make it very difficult to use. Furthermore, the discovery service is tightly integrated with DDS and is not suitable for other platforms.

In [210], Cirani et al. present work on global and local service and resource discovery for the Internet of Things. To handle resource discovery the authors present an *IoT Gateway*. For local networking, there are two ways a device can join a network. If it is aware of an IoT gateway it can join and send its resource information to it, or it can wait for a message broadcast from the gateway alerting the device to its presence. The addresses and resources of devices are added to the gateway which acts as a service look-up to the other devices in the local network and a service provider to external IoT gateways. For IoT gateways to discover and communicate with each other the authors present two P2P overlays. The first is the distributed geographic table (DGT). This table is similar to a distributed hash table but rather than the replicated data being based on hash value assignments the storage is based on the geographic location. This makes it deterministic. For a device to make itself known on a network it contacts a known gateway and shares its location to the DGT. The DGT shares this among the peers and informs the device of other gateways. Once the gateways are known, requests can be made for lists of services that can be accessed and this information is added to the distributed location service which is the other P2P overlay.

In [211] the authors present several important issues in IoT systems including standardization, mobility, networking, and Quality of Service support. To address these issues they present an architecture that combines DDS with software-defined networking (SDN). DDS is responsible for providing discovery and communication between heterogeneous devices within a domain. However, DDS is for local networks. SDN is used to allow communication outside of the local network. By decoupling the control plane from the forwarding plane a SDN controller can provide network interfaces to the local network and when requests can not be filled locally it allows for forwarding rules to be defined, to pass messages

to other networks. It is not clear from the paper how the SDN nodes find each other.

Compared to these solutions, one of the key benefits of our discovery service is that it is completely isolated and compartmentalized from the rest of the framework. In fact, we have seamlessly moved from an earlier Redis based discovery service (not described in this paper) to the DHT based discovery service mentioned in this paper as a drop-in replacement. This is due to the abstraction of register, query, and response interfaces as described in figure 5.4.

5.7 Discussion and Conclusions

Fog computing provides new opportunities for distributed applications and analytics. However as the domain becomes more complex, tools are necessary to assist developers in creating applications by handling the implementation details. One of these details is service discovery.

Service discovery is an essential aspect of fog and edge computing, particularly in dynamic environments. The current mechanisms for handling dynamic discovery are generally ill-equipped as they rely on a central server resulting in increased latency and a single point of failure, or they prioritize consistency over availability which can prevent application deployment if there is a fractured quorum. For highly dynamic discovery, which prioritizes utility over consensus, the only options do not include the discovery of services. This means that service discovery would be an addition. From our initial experiments using distributed hash tables to provide a dynamic discovery service, we see that it is tolerant to egress/ingress scenarios and can scale to at least 32 nodes.

We have shown some example applications on the RIAPS platform and demonstrated our prototype discovery service which allows for fault-tolerant dynamic discovery. There is additional work to be done to verify and improve the capabilities of the platform but the outlook is promising.

Chapter 6

Generalizing the Solution for Multi-Stakeholder Systems - General Market

6.1 Overview

MSCPS applications cover many domains and have the potential to provide participants with the capability to exchange not only data but also resources, there exist concerns regarding integrity, trust, and the need for fair and optimal solutions to the problem of resource allocation. The exchange of information and resources leads to a problem wherein the stakeholders of the system may have limited trust in each other. Thus, collaboratively reaching a consensus on when, how, and who should access certain resources becomes problematic.

In efforts to address some of these issues in a domain-agnostic way, we developed the SolidWorx platform. SolidWorx is a blockchain-based platform that provides key mechanisms required for the trading of resources, that are fungible and can be discretized, across different applications in a domain-agnostic manner. SolidWorx provides a trustworthy mechanism of resource allocation. Doing so strictly with distributed ledger technology is inefficient, as explained in Chapter 3. Therefore, SolidWorx introduces and implements a hybrid-solver pattern, where complex optimization computation is handled off-blockchain while solution validation is performed by a smart contract. We show that while using this hybrid approach, we can ensure the system remains robust against failures by enabling multiple solvers. To ensure correctness, the smart contract of *SolidWorx* is generated and verified using a model-based approach. To enable domain-agnostic trading, it describes a flexible schema that allows for the use of generalized optimization functions to solve the resource allocation problem. To show the versatility of our transaction management platform, we apply it to two case studies: a carpooling service and energy trading within a

microgrid. SolidWorx is integrated with the RIAPS middleware which provides a robust communication and integration layer, allowing SolidWorx to integrate with heterogeneous devices. In this chapter to present SolidWorx.

The works comprising this chapter was published in the 2018 IEEE International Conference on Blockchain (ICBC) [37]. Future work on this topic includes integrating our work on simulation and evaluating and analyzing decentralized fog applications [39].

- S. Eisele, A. Laszka, A. Mavridou, and A. Dubey, "SolidWorx: a resilient and trustworthy transactive platform for smart and connected communities," in 2018 IEEE International Conference on Blockchain (Blockchain 2018), Halifax, Canada, Jul. 2018

- S. Eisele, G. Pettet, A. Dubey, and G. Karsai, "Towards an architecture for evaluating and analyzing decentralized fog applications," in 2017 IEEE Fog World Congress (FWC), Oct 2017, pp. 1–6.

6.2 Introduction

Smart and connected communities (SCC) as a research area lies at the intersection of social science, machine learning, cyber-physical systems, civil infrastructures, and data sciences. This research area is enabled by the rapid and transformational changes driven by innovations in smart sensors, such as cameras and air quality monitors, which are now embedded in almost every physical device and system that we use, ranging from watches and smartphones to automobiles, homes, roads, and workplaces. The effects of these innovations can be seen in many diverse domains, including transportation, energy, emergency response, and health care, to name a few.

At its core, a smart and connected community is a multi-agent system where agents may enter or leave the system for different reasons. Agents may act on behalf of service owners, managing access to services, and ensuring that contracts are fulfilled. Agents can

also act on behalf of service consumers, locating services, entering contracts, as well as receiving and presenting results. For example, agents may coordinate carpooling services. Another example of such coordination exists in transactive energy systems [212], where homeowners in a community exchange excess energy. Consequently, these agents are required to engage in interactions, negotiate with each other, enter agreements, and make proactive run-time decisions—individually and collectively—while responding to changing circumstances.

This exchange of information and resources leads to a problem where the stakeholders of the system may have limited trust in each other. Thus, collaboratively reaching a consensus on when, how, and who should access certain resources becomes problematic. However, instead of solving these problems in a domain-specific manner, we present *SolidWorx* and show how this platform can provide key design patterns to implement mechanisms for arbitrating resource consumption across different SCC applications.

Blockchains may form a key component of SCC platforms because they enable participants to reach a consensus on the value of any state variable in the system, without relying on a trusted third party or trusting each other. Distributed consensus not only solves the trust issue but also provides fault-tolerance since consensus is always reached on the correct state as long as the number of faulty nodes is below a threshold. Further, blockchains can also enable performing computation in a distributed and trustworthy manner in the form of smart contracts. However, while the distributed integrity of a blockchain ledger presents unique opportunities, it also introduces new assurance challenges that must be addressed before protocols and implementations can live up to their potential. For instance, Ethereum smart contracts deployed in practice are riddled with bugs and security vulnerabilities. Thus, we use a correct-by-construction design toolchain, called FSolidM [213], to design and implement the smart-contract code of *SolidWorx*.

The outline of this paper is as follows. We formulate a resource-allocation problem for SCC in Section 6.3, describing two concrete applications of the platform in Section 6.3.2

and presenting extensions to the basic problem formulation in Section 6.3.3. We describe our solution architecture in Section 6.4, which consists of off-blockchain solvers (Section 6.4.2) and a smart contract (Section 6.4.3), providing a brief analysis in Section 6.4.4. In Section 6.5, we evaluate *SolidWorx* using two case studies, a carpooling assignment (Section 6.5.1) and an energy trading system (Section 6.5.2). Finally, we discuss the architecture of *SolidWorx* in the context of related research in Section 6.6, and we provide concluding remarks in Section 6.7.

6.3 Problem Formulation

We first introduce a base formulation of an abstract resource allocation problem (Section 6.3.1), which captures the core functionality of a transactive platform for SCC. Then, we describe two examples of applying this formulation to solving practical problems in SCC (Section 6.3.2). We conclude the section by introducing various extensions to the base problem formulation, in the form of alternative objectives and additional constraints (Section 6.3.3). A list of the key symbols used in the resource allocation problem can be found in Table 6.1.

6.3.1 Resource Allocation Problem

In essence, the objective of the transactive platform is to allocate resources from users who provide resources to users who consume them. The sets of *resource providers* and *resource consumers* are denoted by P and C, respectively. Note that a user may act both as a resource provider and as a resource consumer, in which case the user is a member of both P and C. Resources that are provided or consumed belong to a set of *resource types*, which are denoted by T. A resource type is an abstract concept, which captures not only the inherent characteristics of a resource, but all aspects related to providing or consuming resources. For example, a resource type could correspond to energy production or consumption in a specific time interval, or it could correspond to a ride between certain

Table 6.1: List of Symbols

Symbol	Description
P	set of resource providers
C	set of resource consumers
T	set of resource types
OP	set of providing offers
OC	set of consumption offers
o_P	resource provider who posted offer $\boldsymbol{o} \in OP$
o_C	resource consumer who posted offer $\boldsymbol{o} \in OC$
$o_Q(t)$	amount of resources of type $t \in T$ provided or requested by offer $\boldsymbol{o}$
$o_V(t)$	unit reservation price of offer $\boldsymbol{o}$ for resource type $t \in T$
a_{OP}	providing offer from which assignment $\boldsymbol{a}$ allocates resources
a_{OC}	consuming offer to which assignment $\boldsymbol{a}$ allocates resources
a_Q	amount of resources allocated by assignment $\boldsymbol{a}$
a_T	type of resources allocated by assignment $\boldsymbol{a}$
a_V	unit price for the resources allocated by assignment $\boldsymbol{a}$

locations at a certain time.

Each provider $p \in P$ may post a set of *providing offers*. Each providing offer $\boldsymbol{o}$ is a tuple $\boldsymbol{o} = \langle o_P, o_Q, o_V \rangle$, where $o_P \in P$ is the provider who posted the offer, $o_Q \in T \mapsto \mathbb{N}$ is the amount of resources offered from each type (i.e., $o_Q(t)$ is the amount of resources offered from type $t \in T$), and $o_V \in T \mapsto \mathbb{N}$ is the unit reservation price asked for each resource type (i.e., $o_V(t)$ is the value asked for providing a unit resource of type $t \in T$). Each offer $\boldsymbol{o} = \langle o_P, o_Q, o_V \rangle$ defines a set of alternatives: provider o_p offers to provide either $o_Q(t_1)$ resources of type $t_1 \in T$ or $o_Q(t_2)$ resources of type $t_2 \in T$, but not at the same time. However, convex linear combinations, such as providing $\lfloor \alpha \cdot o_Q(t_1) \rfloor$ resources of type $t_1 \in T$ and $\lfloor (1-\alpha) \cdot o_Q(t_2) \rfloor$ resources of type $t_2 \in T$ at the same time (where $\alpha \in [0,1]$),

are allowed. For example, an offer $\boldsymbol{o}$ providing $o_Q(t_1)$ units of energy in time interval t_1 or $o_Q(t_2)$ units of energy in time interval t_2 may provide $\lfloor 0.5 \cdot o_Q(t_1) \rfloor$ energy in time interval t_1 and $\lfloor 0.5 \cdot o_Q(t_2) \rfloor$ energy in time interval t_2. The set of all offers posted by all the providers is denoted by OP.

Each consumer $c \in C$ posts a set of *consumption offers*. Each consumption offer $\boldsymbol{o}$ is a tuple $\boldsymbol{o} = \langle o_C, o_Q, o_V \rangle$, where $o_C \in C$ is the consumer who posted the offer, $o_Q \in T \mapsto \mathbb{N}$ is the amount of resources requested from each type (i.e., $o_Q(t)$ is the amount of resources requested from type $t \in T$), and $o_V \in T \mapsto \mathbb{N}$ is the unit reservation price offered for each resource type (i.e., $o_V(t)$ is the value offered for a unit resource of type $t \in T$). Similar to providing offers, consumption offers also define a set of alternatives. The set of all offers posted by all the consumers is denoted by OC.

A *resource allocation* A is a set of resource assignments. Each resource assignment $\boldsymbol{a} \in A$ is a tuple $\boldsymbol{a} = \langle a_{OP}, a_{OC}, a_Q, a_T, a_V \rangle$, where $a_{OP} \in OP$ is a providing offer posted by a provider, $a_{OC} \in OC$ is a consumption offer posted by a consumer, $a_Q \in \mathbb{N}$ and $a_T \in T$ are the amount and type of resources allocated from offer a_{OP} to a_{OC}, and $a_V \in \mathbb{N}$ is the unit price for the assignment.

A resource allocation A is *feasible* if

$$\forall \boldsymbol{o} \in OP: \sum_{t \in T} \sum_{\substack{\boldsymbol{a} \in A: \\ a_{OP} = \boldsymbol{o} \wedge a_T = t}} \frac{a_Q}{o_Q(t)} \leq 1 \tag{6.1}$$

$$\forall \boldsymbol{o} \in OC: \sum_{t \in T} \sum_{\substack{\boldsymbol{a} \in A: \\ a_{OC} = \boldsymbol{o} \wedge a_T = t}} \frac{a_Q}{o_Q(t)} \leq 1 \tag{6.2}$$

$$\forall \boldsymbol{a} \in A: (a_{OP})_V(a_T) \leq a_V \tag{6.3}$$

$$\forall \boldsymbol{a} \in A: (a_{OC})_V(a_T) \geq a_V. \tag{6.4}$$

In other words, a resource allocation is feasible if the resources assigned from each providing offer (or consuming offer) is a convex linear combination of the offered (or requested) resources, and if the value in each assignment is higher than (or lower than) the

reservation price of the providing offer (or consuming offer).

The objective of the base formulation of the *resource allocation problem* is to maximize the amount of resources assigned from providers to consumers. We define the base formulation of the problem as follows.

Definition 1 (Resource Allocation Problem)**.** Given sets of providing and consumption offers OP and OC, find a feasible resource allocation A that attains the maximum

$$\max_{A:A \text{ is feasible}} \sum_{a \in A} a_Q. \tag{6.5}$$

6.3.2 Example Applications

To illustrate how the Resource Allocation Problem (RAP) may be applied in smart and connected communities, we now describe two example problems that can be expressed using RAP.

6.3.2.1 Energy Futures Market

We consider a residential energy-futures market in a transactive microgrid. In this application, resource consumers model residential energy consumers (i.e., households), while resource providers model the subset of consumers who have energy-providing capabilities (e.g., solar panels, batteries). We divide time into fixed-length intervals (e.g., 15 minutes), and let each resource type correspond to providing or consuming a unit amount of power (e.g., 1 W) in a particular time interval.

Based on their predicted energy supply and demand, residential consumers (or smart homes acting on their behalf) post offers to provide or consume energy in future time intervals. For instance, a provider may predict that it will be able to generate a certain amount of power π using its solar panel during time intervals $t_1, t_2, \ldots, t_N \in T$. Then, it will submit a *set of N offers*: for each time interval $t \in \{t_1, \ldots, t_N\}$ in which energy may be

produced, it posts an offer specifying

$$o_Q(t) = \begin{cases} \pi & \text{if } t = t \\ 0 & \text{otherwise.} \end{cases} \tag{6.6}$$

Alternatively, the provider may have a fully charged battery, which could be discharged in any of the next N intervals $t_1, t_2 \ldots, t_N$. Let π denote the amount of power that could be provided if the battery was fully discharged in a single time interval. Then, the provider will submit a *single offer* specifying

$$o_Q(t) = \begin{cases} \pi & \text{if } t \in \{t_1, t_2, \ldots, t_N\} \\ 0 & \text{otherwise.} \end{cases} \tag{6.7}$$

The reservation prices of the offers should consider the energy prices of the utility company (i.e., the alternative to local trading) and the cost of providing energy (e.g., cost of battery depreciation due to charging and discharging).

6.3.2.2 Carpooling Assignment

We consider the problem of assigning carpooling riders to drivers with empty seats in their cars. In this application, resource consumers model riders, while resource providers model drivers. We again divide time into fixed-length intervals, and we divide the space of pick-up locations into a set of areas (e.g., city blocks). Then, we let a resource type correspond to a ride from a particular area in a particular time interval to a particular area. A unit of a resource is a single seat for a ride.

A provider (i.e., driver) who has π empty seats in its car will post a providing offer. Let $\Pi \subseteq T$ denote the set of combinations of pick-up and drop-off areas and pick-up times that

are feasible for the provider. Then, the provider's offer specifies

$$o_Q(t) = \begin{cases} \pi & \text{if } t \in \Pi \\ 0 & \text{otherwise.} \end{cases} \tag{6.8}$$

Similarly, a consumer (i.e., rider) who needs 1 seat will post a consuming offer, specifying

$$o_Q(t) = \begin{cases} 1 & \text{if } t \in \Pi \\ 0 & \text{otherwise,} \end{cases} \tag{6.9}$$

where Π is the set of combinations (i.e., pick-up and drop-off areas and pick-up times) that are feasible for the rider.

6.3.3 Problem Formulation Extensions

The Resource Allocation Problem that we introduced in Section 6.3.1 can capture a wide range of real-world problems. However, some problems may not be easily expressed using the constraints (Equations (6.1) to (6.4)) and the objective (Equation (6.5)) of the base problem formulation. For this reason, here we introduce a set of alternative objective formulations and additional constraints for resource allocation.

6.3.3.1 Objectives

We first introduce alternative objective formulations, which quantify the utility of a resource allocation based on alternative goals.

Resource Type Preferences: Equation (6.5) assumes that exchanging a unit of any resource type is equally beneficial. In some practical scenarios, exchanging certain resource types may be more beneficial than exchanging others. For each resource type $t \in T$, let β_t denote the utility derived from exchanging a unit of resources of type t. Then, the utility of

a resource allocation A can be expressed as

$$\sum_{\boldsymbol{a} \in A} \beta_{(a_T)} \cdot a_Q. \tag{6.10}$$

Provider and Consumer Benefit: The reservation price $o_V(t)$ of a providing offer $\boldsymbol{o}$ means that provider o_P is indifferent to (i.e., derives zero benefit from) exchanging resources of type t at unit price $o_V(t)$. Hence, the unit benefit derived by the provider from exchanging at a higher price $a_V \geq o_V(t)$ is equal to $a_V - o_V(t)$. Similarly, the unit benefit derived by a consumer, who posted an offer $\boldsymbol{o}$, from exchanging resources of type t at price a_V is equal to $o_V(t) - a_V$. Therefore, the total benefit created by an assignment $\boldsymbol{a}$ for provider a_{OP} and consumer a_{OC} is

$$\begin{aligned} & a_Q \cdot [a_V - (a_{OP})_V (a_T)] + a_Q \cdot [(a_{OC})_V (a_T) - a_V] \\ =& a_Q \cdot [(a_{OC})_V (a_T) - (a_{OP})_V (a_T)], \end{aligned} \tag{6.11}$$

and the total benefit created by a resource allocation A for all the providers and consumers is

$$\sum_{\boldsymbol{a} \in A} a_Q \cdot [(a_{OC})_V (a_T) - (a_{OP})_V (a_T)]. \tag{6.12}$$

6.3.3.2 Constraints

Next, we introduce additional feasibility constraints that may be imposed on the resource allocations.

Price Constraints: A regulator (e.g., utility company in a transactive energy platform) may impose constraints on the prices at which resources may be exchanged (e.g., based on bulk-market prices). If the minimum and maximum unit prices for resource type $t \in T$ are min_t and max_t, respectively, then we can express price constraints as

$$\forall \boldsymbol{a} \in A : \; min_{(a_T)} \leq a_V \leq max_{(a_T)}. \tag{6.13}$$

Pairwise Constraints: Due to physical constraints, exchanging resources of certain types between certain pairs of prosumers may be impossible. If the set of prosumer pairs that may exchange resources of type $t \in T$ is denoted by $E_t \subseteq P \times C$, we can express pairwise constraints as

$$\forall \boldsymbol{a} \in A : (a_{OP}, a_{OC}) \in E_{(a_T)}. \tag{6.14}$$

Real-Valued Offers and Allocations: Finally, we may also relax some of the constraints of the base formulation. In particular, we may allow real-valued quantities in offers and allocations (i.e., $o_Q : T \mapsto \mathbb{R}_+$ and $a_Q \in \mathbb{R}_+$) as well as real-valued prices (i.e., $o_V : T \mapsto \mathbb{R}_+$ and $a_V \in \mathbb{R}_+$).

6.4 *SolidWorx*: A Decentralized Transaction Management Platform

Now, we describe the *SolidWorx* platform, which (1) allows prosumers[1] to post offers, and (2) can find a solution to the resource allocation problem in an efficient and trustworthy manner. SolidWorx follows the actor-based architecture, which was proposed initially in [214], and which has been accepted as a standard model for building distributed applications. The key aspect of an actor-based system are interfaces with well-defined execution models [215]. The following subsections will describe the transaction management platform in more detail. Here, we provide a brief overview.

Figure 6.1 shows the key actors of our transaction management platform, while Figure 6.2 describes the data flow between these actors. A directory actor provides a mechanism to look up connection endpoints, including the address of a deployed smart contract. We described how to create a decentralized directory service using distributed hash tables in Chapter 5. Prosumer actors (i.e., resource providers and consumers) post offers to the platform via functions provided by a smart contract. These functions check the correctness of each offer and then store it within the smart contract. An optional mixer service can be used to obfuscate the identity of the prosumers [216]. By generating new anonymous addresses

[1] An actor or an agent that can both provide and consume resources.

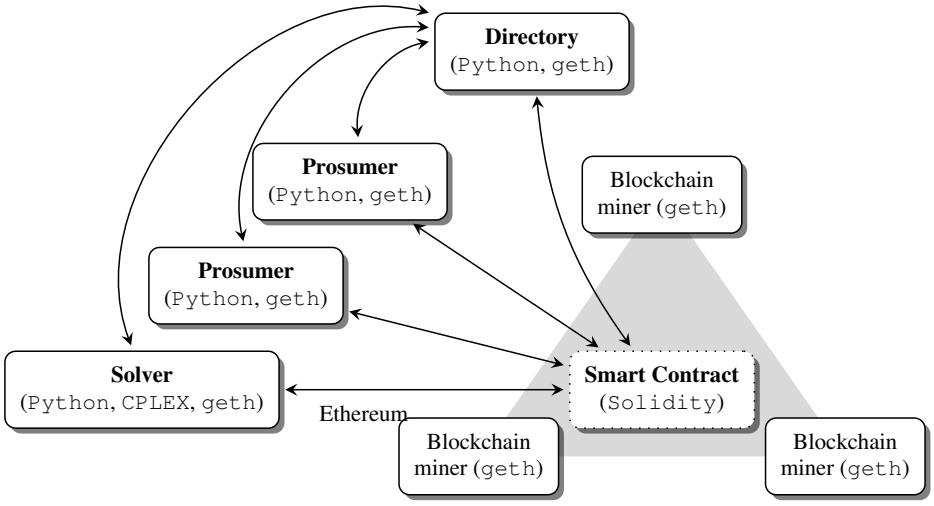


Figure 6.1: Implementation view of the *SolidWorx*. A private Ethereum network (used for testing purposes) is the decentralized computation platform running the smart

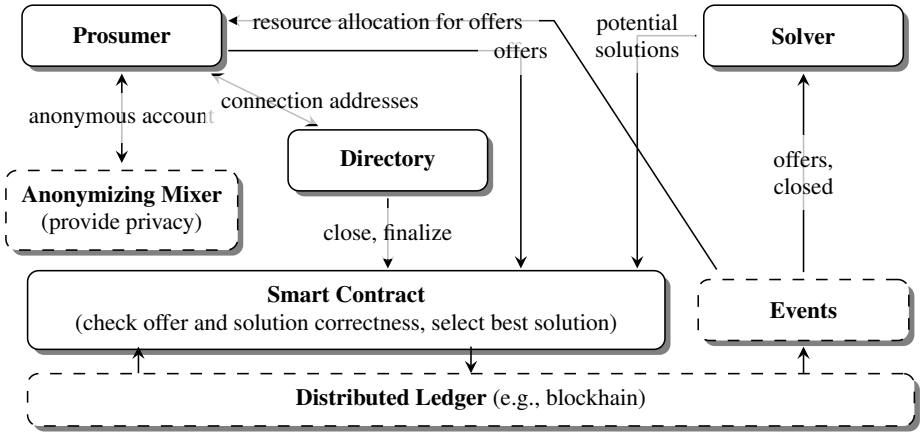


Figure 6.2: Data flow between actors of *SolidWorx*.

at random periodically, prosumers can prevent other entities from linking the anonymous addresses to their actual identities [62, 216], thereby keeping their activities private. Solver actors, which are pre-configured with constraints and an objective function, can listen to smart-contract events, which provide the solvers with information about offers. Solvers run at pre-configured intervals, compute a resource allocation, and submit the solution allocation to the smart contract. The directory, acting as a service director, can then finalize a solution by invoking a smart-contract function, which chooses the best solution from all the allocations that have been submitted. Once a solution has been finalized, the prosumers

are notified using smart-contract events.

6.4.1 Scheduling

Once the system is deployed, providers and consumers will need to use it repeatedly for finding optimal resource allocations. For example, riders and drivers want to find optimal carpooling assignments every day, while users in an energy futures market want to find optimal energy trades every, e.g., 20 minutes. Consequently, the platform has to gather offers and solve the resource allocation problem at regular time intervals. Each one of these cycles is divided into two phases. First, an *offering phase*, in which providers and consumers can post new offers or cancel their existing offers (e.g., if they wish to change their offer based on changes in the market). Second, a *solving phase*, in which the resource allocation problem is solved for the posted (but not canceled) offers. At the end of the second phase, the assignments between providers and consumers are finalized based on the solution. Then, a new cycle begins with an offering phase.

6.4.2 Hybrid Solver Architecture

The Resource Allocation Problem described in Section 6.3 can be solved by formulating it as an (integer) linear program (LP): feasibility constraints (Equations (6.1) to (6.4)) and constraint extensions (Section 6.3.3.2) can all be formulated as linear inequalities, while the objective function (Equation (6.5)) as well as the alternative objectives (Section 6.3.3.1) can be formulated as linear functions. Arguably, we could set up a solver actor that would receive offers from prosumers, formulate a linear program, and use a state-of-the-art LP-solver (e.g., CPLEX [72]) to find an optimal solution. However, a simple N-to-1 architecture with N prosumers and 1 solver would suffer from the following problems:

- *Lack of trust in solver nodes:* Prosumers would need to trust that the solver is acting selflessly and is providing correct and optimal solutions.

- *Vulnerability of the transaction management platform:* A single solver would be a single point of failure. If it were faulty or compromised, the entire platform would be faulty or compromised.

- *Data storage:* For the sake of auditability, information about past offers and allocations should remain available even in case of node failures.

A decentralized ledger with distributed information storage and consensus provided by blockchain solutions, such as Ethereum, is an obvious choice for ensuring the auditability of all events and providing distributed trust. However, computation is relatively expensive on blockchain-based distributed platforms[2], solving the trading problem using a blockchain-based smart contract would not be scalable in practice. In light of this, we propose a *hybrid implementation approach*, which combines the trustworthiness of blockchain-based smart contracts with the efficiency of more traditional computational platforms.

Thus, the key idea of our hybrid approach is to (1) use a high-performance computer to solve the computationally expensive linear program *off-blockchain* and then (2) use a smart contract to record the solution *on the blockchain*. To implement this hybrid approach securely and reliably, we must address the following issues.

- A computation that is performed off-blockchain does not satisfy the auditability and security requirements that smart contracts do. Thus, the results of any off-blockchain computation must be verified by the smart contract before recording them on the blockchain.

- Due to network disruptions and other errors (including deliberate denial-of-service attacks), the off-blockchain solver might fail to provide the smart contract with a solution on time (i.e., before assignments are supposed to be finalized). Thus, the smart contract must not be strongly coupled to the solver.

[2]Further, Solidity, the preferred high-level language for Ethereum, currently lacks built-in support for certain features that would facilitate the implementation of an LP solver, such as floating-point arithmetics.

- For the sake of reliability, the smart contract should accept solutions from multiple off-blockchain sources; however, these sources might provide different solutions. Thus, the smart contract must be able to choose from multiple solutions (some of which may come from a compromised computer).

6.4.3 Smart Contract

We implement a smart contract that can (1) verify whether a solution is feasible and (2) compute the value of the objective function for a feasible solution. Compared to finding an optimal solution, these operations are computationally inexpensive, and they can easily be performed on a blockchain-based decentralized platform. Thus, we implement a smart contract that provides the following functionality:

- Solutions may be submitted to the contract at any time during the solving phase. The contract verifies the feasibility of each submitted solution, and if the solution is feasible (i.e., if it satisfies the constraint Equations (6.1) to (6.4)), then it computes the value of the objective function (i.e., Equation (6.5)). The contract always keeps track of the best feasible solution submitted so far, which we call the *candidate solution*.

- At the end of the solving phase, the contract finalizes resource assignments for the cycle based on the candidate solution. If no solution has been submitted to the contract, then an empty allocation is used as a solution, which is always feasible but attains zero objective.

This simple functionality achieves a high level of security and reliability. Firstly, it is clear that an adversary cannot force the contract to finalize assignments based on an incorrect (i.e., infeasible) solution since such a solution would be rejected. Similarly, an adversary cannot force the contract to choose an inferior solution instead of a superior one. In sum, the only action available to the adversary is proposing a superior feasible solution, which would actually improve the transactive management platform.

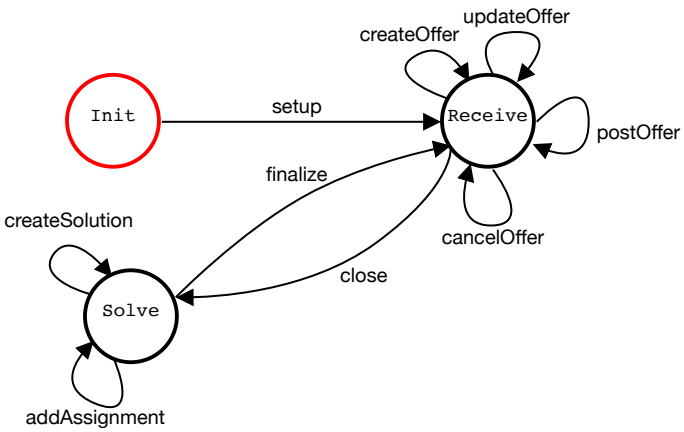


Figure 6.3: FSolidM model of the *SolidWorx* smart contract.

To ensure that the smart-contract code is correct-by-construction [217], we use the formal design environment FSolidM [213] to design and generate the Solidity code of the smart contract. FSolidM allows designing Ethereum smart contracts as Labelled Transition Systems (LTS) with formal semantics. Each LTS can then be given to the NuSMV model checker [218] to verify liveness, deadlock-freedom, and safety properties, which can capture important security concerns.

In Figure 6.3, we present the LTS representation of the transactive-platform smart contract, designed with FSolidM. The contract has three states:[3]

- `Init`, in which the contract has been deployed but not been initialized. Before the contract can be used, it must be initialized (i.e., numerical parameters must be set up).
- `Receive`, which corresponds to the *offering* phase of a cycle (see Section 6.4.1). In this state, prosumers may post (or cancel) their offers.
- `Solve`, which corresponds to the *solving* phase of a cycle (see Section 6.4.1). In this state, solvers may submit solutions (i.e., resource allocations) based on the posted (but not canceled) offers.

In FSolidM, smart-contract functions are modeled as LTS transitions. Note that by

[3]Generated smart-contract code is not included in the paper because of space constraints. However, interested readers can view the code at https://github.com/visor-vu/transaction-management-platform

design, each function may be executed only if the contract is in the origin state of the corresponding transition. Our smart contract has the following transitions (after the name of each transition, we list the function parameters):

- from state `Init`:
  - `setup(uint64 numTypes, uint64 precision, uint64 maxQuantity, uint64 lengthReceive, uint64 lengthSolve)`: initializes a contract with numerical parameter values, setting up the number of resource types, the arithmetic precision for calculations, the maximum quantity that may be offered, and the time length of the offering and solving phases; upon execution, the contract transitions to state `Receive`.
- from state `Receive`:
  - `createOffer(bool providing, uint64 misc))`: creates a blank offer (belonging to the prosumer invoking this transition) within the smart contract; parameter `providing` is true for providers and false for consumers, parameter `misc` is an arbitrary value that prosumers may use for their own purposes (e.g., to distinguish between their own offers); emits an `OfferCreated` event.
  - `updateOffer(uint64 ID, uint64 resourceType, uint64 quantity, uint64 value)`: sets quantity and value for a resource type in an existing offer (identified by the `ID` given in the `OfferCreated` event); may be invoked only by the entity that created the offer, and only if the offer exists but has not been posted yet; emits an `OfferUpdated` event.
  - `postOffer(uint64 ID)`: posts an existing offer, enabling solvers to include this offer in a solution; may be invoked only by the entity that created the offer; emits an `OfferPosted` event.
  - `cancelOffer(uint64 ID)`: cancels (i.e., "un-posts") an offer, forbidding solvers from including this offer in a solution; may be invoked only by the entity that created the offer; emits an `OfferCanceled` event.
  - `close()`: protected by a guard condition on time, which prevents the execution of this transition before the offering phase of the current cycle ends; transitions to state `Solve`; emits a `Closed` event.

- from state `Solve`:
  - `createSolution(uint64 misc)`: creates a new, empty solution (i.e., resource allocation) within the smart contract; parameter `misc` is an arbitrary value that solvers may use for their own purposes (e.g., to distinguish between their own solutions); emits a `SolutionCreated` event.
  - `addAssignment(uint64 ID, uint64 providingOfferID, uint64 consumingOfferID, uint64 resourceType, uint64 quantity, uint64 value)`: adds a resource assignment to an existing solution (identified by the `ID` given in the `SolutionCreated` event); may be invoked only by the entity that created the solution; checks several constraints ensuring that the solution remains valid if this assignment is added; emits an `AssignmentAdded` event.
  - `finalize()`: selects the best solution and finalizes it by emitting an `AssignmentFinalized` event for each assignment in the solution; protected by a guard condition on time, which prevents the execution of this transition before the solving phase of the current cycle ends; transitions to state `Receive`.

Notice that posting an offer or submitting a solution requires at least three or two function calls, respectively. The reason for dividing these operations into multiple function calls is to ensure that the computational cost of each function call is constant:

- `createOffer`, `postOffer`, `cancelOffer`, and `createSolution` are obviously constant-cost.
- `updateOffer` adds a single resource type to an offer.
- `addAssignment` simply updates the sum amounts on the left-hand sides of Equations (6.1) and (6.2) for a single providing and a single consuming offer, respectively; and then it updates the sum in Equation (6.5).

With variable-cost functions, posting a complex offer, or submitting a complex solution could be infeasible due to large computational costs, which could exceed the gas limit.[4]

[4] In Ethereum, each transaction is allowed to consume only a limited amount of gas, which corresponds to the computational and storage cost of executing the transaction.

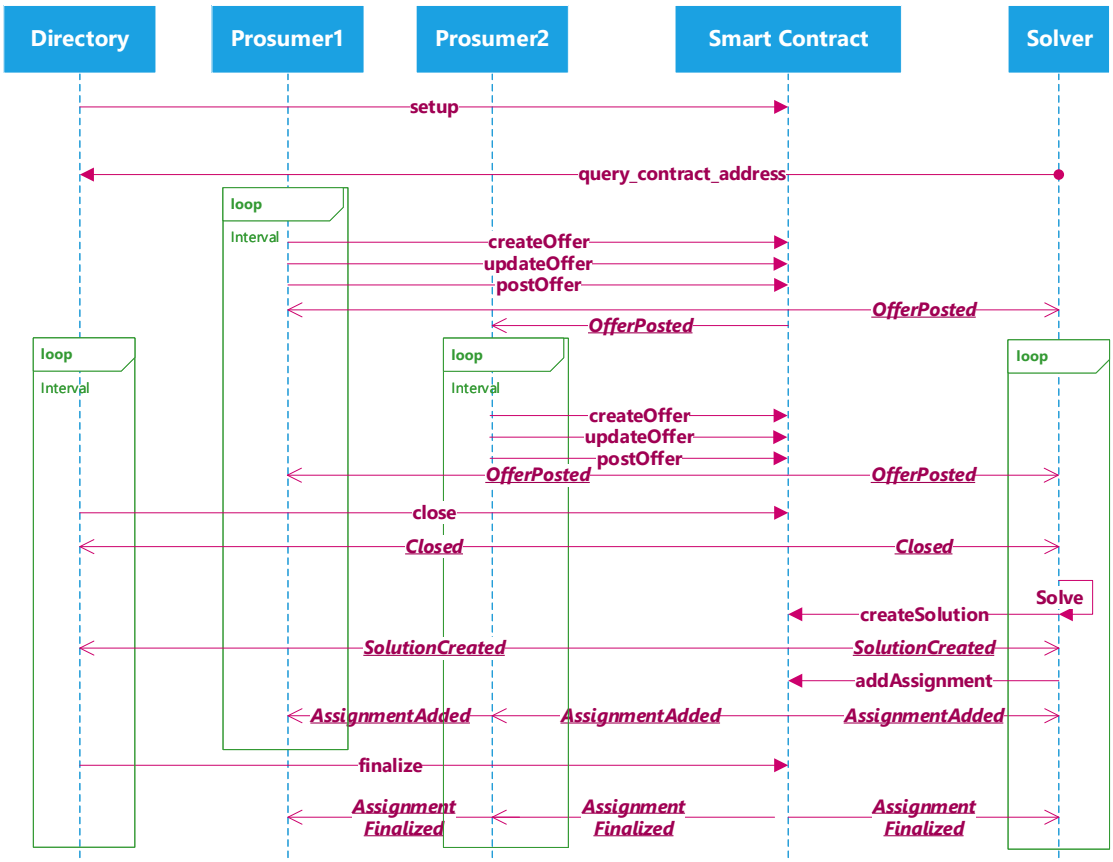


Figure 6.4: A possible sequence of operations in *SolidWorx*. Underlined text denotes events emitted by the smart contract. Some events, such as `OfferUpdated`, are omitted for simplicity.

A typical sequence of function calls and events in *SolidWorx* is shown in Figure 6.4.

6.4.4 Analysis

The computational cost of every smart-contract function is constant (i.e., $O(1)$) except for `finalize`, whose cost is an affine function of the size of the solution (i.e., $O(|A|)$). Note that the cost of `finalize` depends on the size of the solution A only because it emits an event for every assignment $\boldsymbol{a} \in A$. These could be omitted for the sake of maximizing performance since the assignments have already been recorded in the blockchain anyway. The number of function calls required for posting an offer depends on the number of resource types with non-zero quantity in the offer. If there are n such resource types, then $n+2$ calls are required (`createOffer`, n `updateOffer`, and `postOffer`). The num-

ber of function calls required for submitting a solution A is $1 + |A|$ (`createSolution` and $|A|$ `addAssignment`).

6.4.4.1 Verification

For the specification of safety and liveness properties, we use Computation Tree Logic (CTL) [219]. CTL formulas specify properties of execution trees generated by transitions systems. The formulas are built from atomic predicates that represent transitions and statements of the transition system (i.e., smart contract), using several operators, such as `AX`, `AF`, `AG` (unary) and, `A`$[\cdot$`U`$\cdot]$, `A`$[\cdot$`W`$\cdot]$ (binary). Each operator consists of a quantifier on the branches of the tree and a temporal modality, which together define when in the execution the operand sub-formulas must hold. The intuition behind the letters is the following: the branch quantifier is `A` (for "All") and the temporal modalities are `X` (for "neXt"), `F` (for "some time in the Future"), `G` (for "Globally"), `U` (for "Until") and `W` (for "Weak until"). A property is satisfied if it holds in the initial state of the transition systems. For instance, the formula `A`$[p$`W`$q]$ specifies that in *all execution branches* the predicate p must hold *up to the first state* (not including this latter) where the predicate q holds. Since we used the weak until operator `W`, if q never holds, p must hold forever. As soon as q holds in one state of an execution branch, p does not need to hold anymore, even if q does not hold. On the contrary, the formula `AG A`$[p$`W`$q]$ specifies that the subformula `A`$[p$`W`$q]$ must hold in *all branches at all times*. Thus, p must hold whenever q does not hold, i.e., `AG A`$[p$`W`$q] =$ `AG`$(p \vee q)$.

We verified correctness of behavioral semantics with the NuSMV model checker [218], by verifying the following properties:

- *deadlock-freedom*, which ensures that the contract cannot enter a state in which progress is impossible;
- *"if* `close` *happens, then* `postOffer` *or* `cancelOffer` *can happen only after* `finalize`*"*, translated to CTL as: `AG`(`close`) $\rightarrow$ `AX A` [$\neg$(`postOffer` $\wedge$ `cancelOffer`) `W finalize`], which ensures that prosumers cannot post or cancel their offers once the solvers have started

working;

- *"`OfferPosted(ID)` can happen only if `(ID < offers.length) && !offers[ID].posted && (offers[ID].owner == msg.sender)`"*, translated to CTL as: $\mathbf{A}[\neg$`OfferPosted(ID)` $\mathbf{W}$ `(ID < offers.length) && !offers[ID].posted && (offers[ID].owner == msg.sender)`$]$, which ensures that an offer can be posted only if it has been created (but not yet posted) and only by its creator;
- *"`OfferCanceled(ID)` can happen only if `(ID < offers.length) && offers[ID].posted && (offers[ID].owner == msg.sender)`"*, translated to CTL as: $\mathbf{A}[\neg$`OfferCanceled(ID)` $\mathbf{W}$ `(ID < offers.length) && offers[ID].posted && (offers[ID].owner == msg.sender)`$]$, which ensures that an offer can be canceled only if it has been posted and only by the poster;
- *"if `finalize` happens, then `createSolution` can happen only after `close`"*, translated to CTL as: $\mathbf{AG}($`finalize`$) \rightarrow \mathbf{AX}\ \mathbf{A}\ [\neg$`createSolution` $\mathbf{W}$ `close`$]$, which ensures that solutions can be posted only in the solving phase.

6.5 Case Studies

To evaluate our platform, we present two case studies, based on the energy trading and carpooling problems (Section 6.3.2), with numerical results. The computational results for the carpool example were obtained on a virtual machine configured with 16 GB of RAM and 4 cores of an i7-6700HQ processor. The energy market example results were obtained on a virtual machine configured with 8GB of RAM and 2 cores of an i7-6700HQ processor. For these experiments, we used our private Ethereum blockchain network [69].

6.5.1 Carpooling Problem

In this section, we describe a simulated carpooling scenario. The problem of carpooling assignment was introduced earlier in Section 6.3.2.2. Here, we model a carpool prosumer

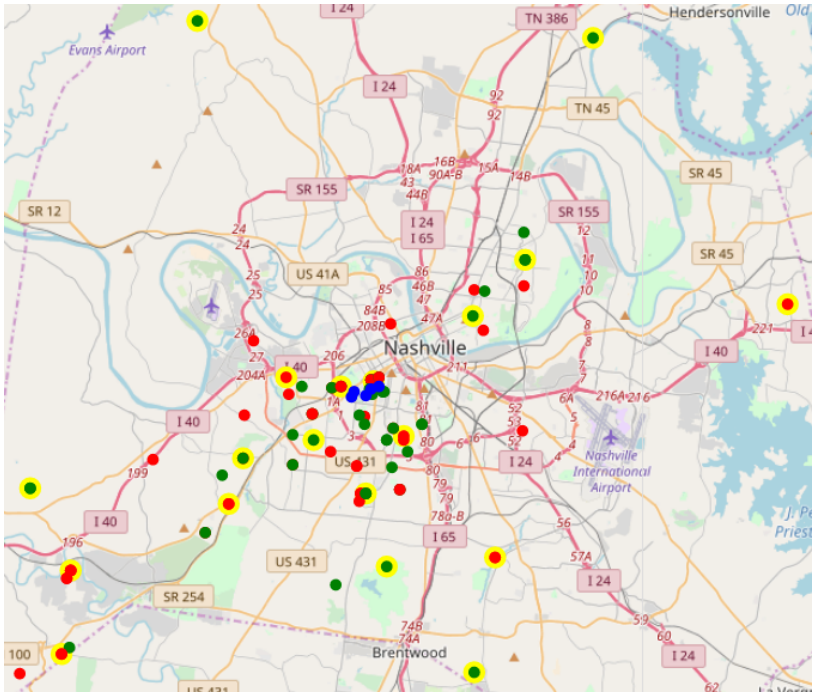


Figure 6.5: Green and red dots mark the 75 residences (anonymized and resampled). Blue dots are destinations on campus. We used K-Means to identify 20 central locations (yellow dots) for pickup.

as an actor that specifies 1. whether it is providing or requesting a ride, 2. the number of seats being offered/requested, 3. a residence, 4. a destination, 5. a time interval during which the ride is available/required, 6. and a radius specifying how far out of their way they are willing to travel. To set up the carpooling problem, we need to identify these parameters and encode them as offers.

Residences were generated by sampling from the real-trip distribution data of Vanderbilt University. Destinations were chosen uniformly at random for each prosumer from the 5 garages around Vanderbilt University. Other parameters were also chosen randomly: number of seats from the range of 1 to 3, prosumer type from producer or consumer, time interval from 15-minute intervals between 7:00 and 9:30 AM. The "out of the way" metric was chosen to be half of the distance between the residence and the destination. For a provider, the center of the pick-up circle is the midway point between the residence and the destination, and for a consumer, the center is the residence.

Since each prosumer has a distinct residence, encoding it as a unique resource type would mean that every prosumer would need to have the address of every other prosumer to determine if they are in their pick-up range. Instead, we specify *pick-up points* which are public locations where carpoolers can meet. Each prosumer can determine which pick-up points are within their out-of-the-way radius and list those points in their offer. To encode these values, we assign an ID to each pickup point and destination. Finally, we encode each 15-minute interval using a timestamp.

An offer consists of a collection of alternative resource types, each with a quantity and value. We encode a resource type, which is a combination of a time interval, a pick-up point, and a destination, as a 64-bit unsigned integer. For example, if the timestamp is 1523621700, the pick-up location ID is 15, and the destination ID is 3, then the resource type is 1523621700153. A complete offer may look as follows:

$$\{\texttt{True},\ \{1523623500173:2, 1523623500153:2,$$
$$1523624400153:2, 1523624400173:2\},$$
$$\{1523623500173:10, 1523623500153:10,$$
$$1523624400153:10, 1523624400173:10\}\ \}.$$

In this offer, the prosumer is offering rides (`True` for providing), has two pick-up locations in range (17 and 15), drives to destination 3, is available in two time intervals, offers 2 seats, and asks for value 10 in exchange for a ride.

In our experiment, we selected 75 prosumers for the carpool service simulation. The red and green points in Figure 6.5 are the locations of the consumers and producers randomly sampled from the anonymized distribution data of employees of Vanderbilt University. The yellow points were selected as pick up locations using K-Means clustering choosing 20 clusters. The blue points are 5 garages around the Vanderbilt campus where employees typically park.

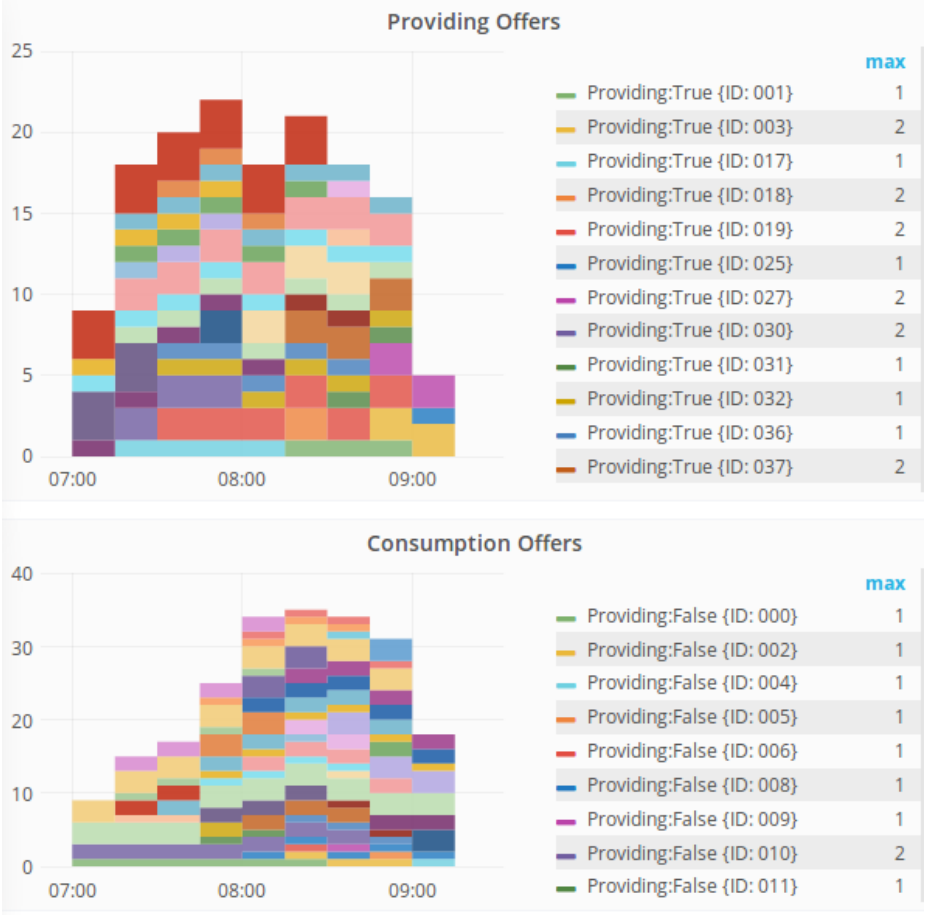


Figure 6.6: Each bar is a 15-minute interval. Each color in a bar is an offer that is valid during that interval. The height of each color is the number of seats offered. If the color appears during another interval that means it could be matched in any one of them, but no more than one.

Figure 6.6 shows all the offers posted to the platform. Figure 6.7 shows the production and consumption offers that were matched. The running time of the solver was 23 ms, while the time between the request for finalization and emission of `AssignmentFinalized` events was 29 s.

6.5.2 Energy Trading Problem

To show the versatility of our transaction management platform, we now apply it to the problem of energy trading within a microgrid, which we introduced in Section 6.3.2.1.

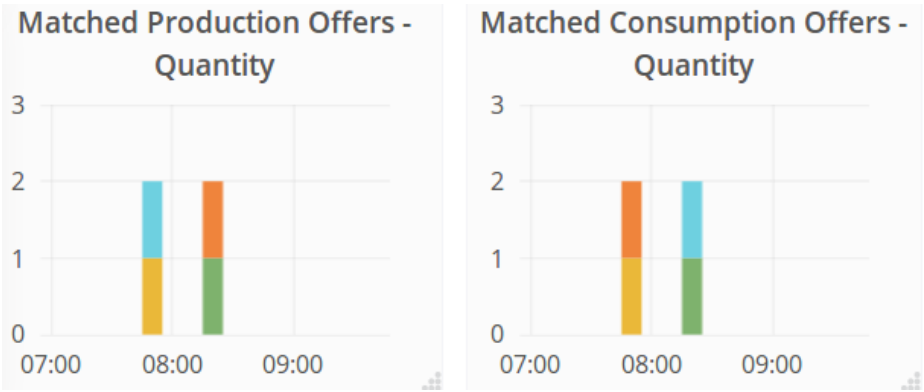


Figure 6.7: 4 production and 4 consumption offers that were matched. The blue and yellow production offers are matched with the orange and yellow consumption offers. The height of each color is the number of seats in that offer that were matched.

In this example, a prosumer is modeled as an actor with an energy generation and consumption profile for the near future. In practice, the generation profile would be typically derived from predictions based on the weather, energy generation capabilities, and the amount of battery storage available. The consumption profile would be derived from flexible power loads, like washers and electric vehicles.

To represent future generation or consumption at a certain time, resource types encode timestamps for 15-minute intervals, during which the power will be generated or consumed. As an example, consider a battery that has 500 Wh energy, which could be discharged at any time between 9 and 10 AM. This can be represented by an offer having resource types 900, 915, 930, and 945, specifying a quantity of 500 Wh for each.

For our simulation, the prosumer energy profiles are load traces recorded by Siemens during a day from a microgrid in Germany, containing 102 homes (5 producers and 97 consumers). Since the dataset does not include prices, we assume reservation prices to be uniform in our experiments and focus on studying the amount of energy traded and the performance of the system.

Figure 6.8 shows the total production and consumption across this microgrid, as well as the total energy traded per interval using our platform. The horizontal axis shows the starting time for each of the 96 intervals.

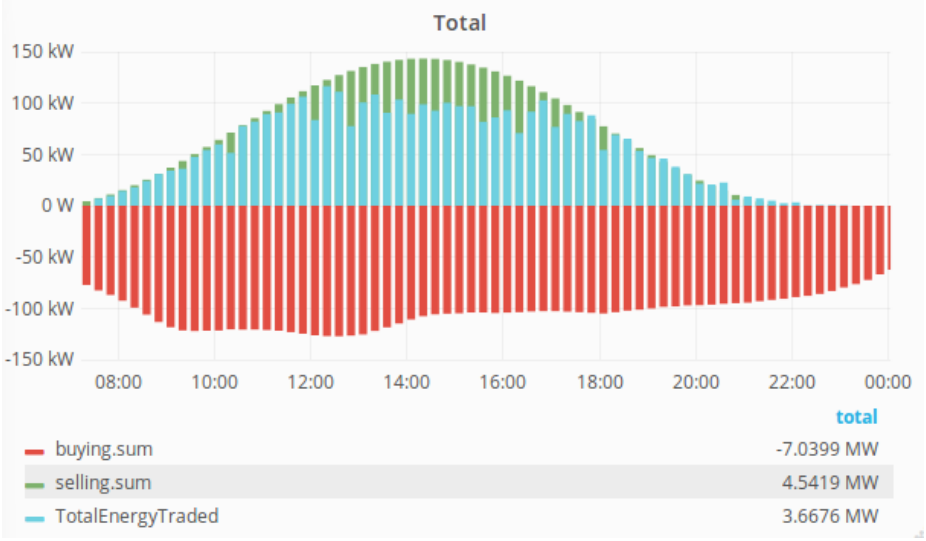


Figure 6.8: Total energy production capacity (green) and energy demand (red) for each interval, as well as the total energy traded in each interval (blue).

In another simulation, we exercise the hybrid solver architecture by running multiple solvers, and after some time, cause one to fail. This result is shown in Figure 6.9. The narrow vertical red line indicates when solver 1 fails at 8:15 AM. Up until that point, solver 1 submitted the green, yellow, light blue, orange solutions, with the final solution being red. On the other hand, we see that solver 2 continues to provide solutions for later intervals.

6.6 Related Research

Online Information Management Platforms: Smart and connected community systems are designed to collect, process, transmit, and analyze data. In this context, data collection usually happens at the edge because that is where edge devices with sensors are deployed to monitor surrounding environments. *SolidWorx* does not suggest a specific data collection methodology. Rather, it follows an actor-driven design pattern where "prosumer" actors can integrate their own agents into *SolidWorx* by using the provided APIs. Another concern of these platforms is the cost of processing. Traditionally, this problem was solved using scalable cloud resources in-house [220]. However, *SolidWorx* enables a decentralized ecosystem, where components of the platform can run directly on edge nodes, which is one

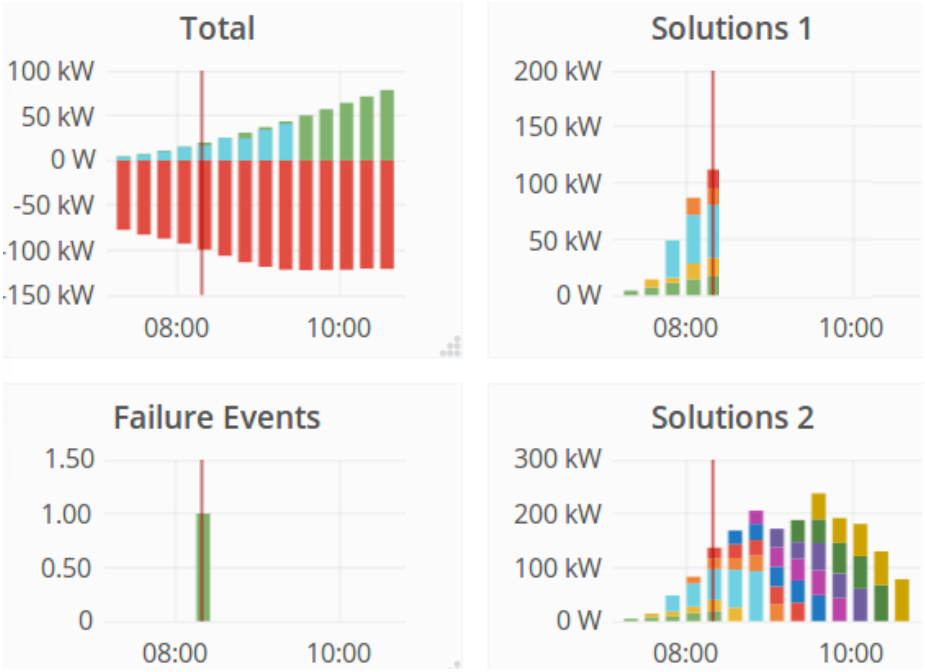


Figure 6.9: A failure scenario with failure at 8:15 AM. The solver can submit new solutions as time progresses; the most recent solution is the color that is on the top of the stack for an interval.

of the reasons why we designed it to be asynchronous.

To an extent, the information architecture of *SolidWorx* can be compared to dataflow engines [221, 222, 223]. All of these existing dataflow engines use some form of a computation graph, comprising computation nodes and dataflow edges. These engines are designed for batch-processing and/or stream-processing high volumes of data in resource-intensive nodes and do not necessarily provide additional "platform services" like trust management or solver nodes.

Integration with Blockchains: *SolidWorx* integrates a blockchain because it enables the digital representation of resources, such as energy and financial assets, and their secure transfer from one party to another. Further, blockchains constitute an immutable, complete, and fully auditable record of all transactions that have ever occurred in the system. This is in line with the increased interest and commercial adoption of blockchains [224], which has yielded market capitalization surpassing USD 75 billion [225] for Bitcoin and USD 36

billion USD for Ethereum [226]. Prior work has also considered the security and privacy of IoT and Blockchain integrations [227, 228, 229].

The biggest challenge in these integrated systems comes from computational-complexity limitations and the complexity of the consensus algorithms. In particular, their transaction-confirmation time is relatively long and variable, primarily due to the widely-used proof-of-work algorithm. Further, blockchain-based computation is relatively expensive, which is the main reason why we separated finding a solution and validating the solution into two separate components in *SolidWorx*.

Correctness of Smart Contracts: Both verification and automated vulnerability discovery are considered in the literature for identifying smart-contract vulnerabilities. For example, Hirai performs a formal verification of a smart contract that is used by the Ethereum Name Service [230]. However, this verification proves only one property and it involves a relatively large amount of manual analysis. In later work, Hirai defines the complete instruction set of the Ethereum Virtual Machine in Lem, a language that can be compiled for interactive theorem provers [231]. Using this definition, certain safety properties can be proven for existing contracts.

Bhargavan et al. outline a framework for verifying the safety and correctness of Ethereum smart contracts [232]. The framework is built on tools for translating Solidity and Ethereum Virtual Machine bytecode contracts into F^*, a functional programming language aimed at program verification. Using the F^* representations, the framework can verify the correctness of the Solidity-to-bytecode compilation as well as detect certain vulnerable patterns. Luu et al. provide a tool called OYENTE, which can analyze smart contracts and detect certain typical security vulnerabilities [233]. The main difference between prior work and the approach that we are using (i.e., verifying FSolidM models with NuSMV) is that the former can prevent a set of typical vulnerabilities, but they are not effective against vulnerabilities that are atypical or belong to types which have not been identified yet.

6.7 Conclusion

Smart and connected community applications require decentralized and scalable platforms due to the large number of participants and the lack of mutual trust between them. In this paper, we introduced a transactive platform for resource allocation, called *SolidWorx*. We first formulated a general problem that can be used to represent a variety of resource allocation problems in smart and connected communities. Then, we described an efficient and trustworthy platform based on a hybrid approach, which combines the efficiency of traditional computing environments with the trustworthiness of blockchain-based smart contracts. Finally, we demonstrated the applicability of our platform using two case studies based on real-world data.

Chapter 7

Conclusions and Future Directions

7.1 Conclusions

This dissertation makes several important contributions to the field, which we reiterate here.

First, this dissertation addresses the conflicting requirements of privacy and safety in a transactive energy market. We presented an analysis and design of privacy groups that can achieve k-anonymity while maintaining safety in MSCPS, by the development of a platform known as TRANSAX (Chapter 2). This is important because some applications require data in order to take control actions that ensure overall system safety, but the sharing of that information may violate user privacy. By allocating an in-network currency (representing safe trading limits based on physical system constraints) and using a mixing protocol, we demonstrated an ability to provide a meaningful degree of privacy to participants in a transactive market.

Second, this dissertation presented market protocols designed to enable trust in decentralized systems without relying exclusively on cryptographic protocols. In the development of MODiCuM, we constructed an incentive-compatible market for outsourcing computation. To do so, we performed a game-theoretic analysis of participant strategies to verify that a rational participant would behave correctly with overwhelming probability (Chapter 3). We initially designed MODiCuM for batch processing, but then redesigned the platform to be able to function for stream processing (Chapter 4); in doing so we were able to construct the market such that rational participants will behave correctly.

Third, we presented elements necessary for the generalization of these solutions, including a connection substrate and a generalized resource allocation market. In addition

to these, we presented a hybrid-solver pattern for blockchain-based markets, which allows stakeholders to coordinate without a central authority while overcoming some of the inefficiencies of a purely blockchain-based market, allowing for reduction of costs and preservation of trust. This hybrid-solver pattern enables the system to execute computation with the non-secure resource and use the secure resource to verify the result (Chapter 6).

7.2 Future Work

MSCPS are complex systems that require special considerations in their design and development. However, such systems are rising to prominence in Smart and Connected Communities, which present many opportunities for many stakeholders to interact in mutually beneficial ways. Distributed ledgers are a valuable tool in the development of these systems, allowing for decentralization while maintaining trust between participants, but the application of distributed ledger technology to MSCPS is not straightforward.

While we have presented several platforms that utilize blockchains and may facilitate the exchange of resources among many stakeholders, there is significant work yet to be accomplished. The overarching goal of research in this arena is to have a unified platform wherein all the requirements of MSCPS can be met across a variety of domains. Such a platform would encompass the difficult-to-implement aspects of MSCPS and account for all of the requirements when it comes to communication between nodes, system resilience, system testing and diagnostics, etc. This theoretical platform could allow for groups to write market-based solutions for whatever domain they are interested in, and allow those groups to test their solutions, provide strategies to their users, and explore an array of market-based opportunities.

While we have not yet achieved this goal, this dissertation provides key patterns and initial platforms which can be further expounded upon as the field continues this pursuit.

BIBLIOGRAPHY

[1] Edward A. Lee and Sanjit A. Seshia. *Introduction to Embedded Systems: a Cyber-Physical Systems Approach*. MIT Press, second edition, 2017.

[2] Edward A. Lee. Cyber Physical Systems: Design Challenges. Technical Report UCB/EECS-2008-8, EECS Department, University of California, Berkeley, Jan 2008.

[3] Victor Bolbot, Gerasimos Theotokatos, Luminita Manuela Bujorianu, Evangelos Boulougouris, and Dracos Vassalos. Vulnerabilities and Safety assurance methods in Cyber-Physical Systems: A comprehensive review. *Reliability Engineering & System Safety*, 182:179 – 193, 2019.

[4] George Coulouris, Jean Dollimore, Tim Kindberg, and Gordon Blair. *Distributed Systems: Concepts and Design*. Addison-Wesley Publishing Company, USA, 5th edition, 2011.

[5] Wei Zhang, Michael S. Branicky, and Stephen M. Phillips. Stability of networked control systems. *IEEE Control Systems Magazine*, 21:84–99, 2001.

[6] Michel Barbeau, Georg Carle, Joaquín García-Alfaro, and Vicenç Torra. Next generation resilient cyber-physical systems. *CoRR*, abs/1907.08849, 2019.

[7] Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance. *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, page 14, 1999.

[8] NERC, 3353 Peachtree Road NE Suite 600, North Tower, Atlanta, GA 30326. *Reliability Guideline Methods for Establishing IROLs*, 2018.

[9] Marisol García-Valls, Abhishek Dubey, and Vicent Botti. Introducing the new paradigm of Social Dispersed Computing: Applications, Technologies and Challenges. *Journal of Systems Architecture*, 91:83 – 102, 2018.

[10] U. Bodkhe, D. Mehta, S. Tanwar, P. Bhattacharya, P. K. Singh, and W. Hong. A survey on decentralized consensus mechanisms for cyber physical systems. *IEEE Access*, 8:54371–54401, 2020.

[11] Diego Ongaro and John Ousterhout. In Search of an Understandable Consensus Algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, page 305–320, USA, 2014. USENIX Association.

[12] Leslie Lamport et al. Paxos made simple. *ACM Sigact News*, 32(4):18–25, 2001.

[13] 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)*, pages 245–256. IEEE, 2011.

[14] Atul Singh, Pedro Fonseca, Petr Kuznetsov, Rodrigo Rodrigues, and Petros Maniatis. Zeno: Eventually Consistent Byzantine-Fault Tolerance. *Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, page 16, 2009.

[15] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The Honey Badger of BFT Protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 31–42. Association for Computing Machinery, 2016.

[16] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Available at: http://www.bitcoin.org/bitcoin.pdf, 2009.

[17] Christopher Natoli, Jiangshan Yu, Vincent Gramoli, and Paulo Esteves-Verissimo. Deconstructing Blockchains: A Comprehensive Survey on Consensus, Membership and Structure. *arXiv:1908.08316 [cs]*, 2019.

[18] Hao Chen, Kim Laine, and Rachel Player. Simple Encrypted Arithmetic Library - SEAL v2.1. In Michael Brenner, Kurt Rohloff, Joseph Bonneau, Andrew Miller, Peter Y.A. Ryan, Vanessa Teague, Andrea Bracciali, Massimiliano Sala, Federico Pintore, and Markus Jakobsson, editors, *Financial Cryptography and Data Security*, volume 10323, pages 3–18. Springer International Publishing, 2017.

[19] Sai Sri Sathya, Praneeth Vepakomma, Ramesh Raskar, Ranjan Ramachandra, and Santanu Bhattacharya. A Review of Homomorphic Encryption Libraries for Secure Computation. *arXiv:1812.02428 [cs]*, 2018.

[20] Carlos Aguilar Melchor, Marc-Olivier Kilijian, Cédric Lefebvre, and Thomas Ricosset. A Comparison of the Homomorphic Encryption Libraries HElib, SEAL and FV-NFLlib. In Jean-Louis Lanet and Cristian Toma, editors, *Innovative Security Solutions for Information Technology and Communications*, volume 11359, pages 425–442. Springer International Publishing, 2019.

[21] Shen Noether. Ring Signature Confidential Transactions for Monero. Cryptology ePrint Archive, Report 2015/1098, 2015.

[22] Tim Ruffing, Pedro Moreno-Sanchez, and Aniket Kate. CoinShuffle: Practical Decentralized Coin Mixing for Bitcoin. In *Proceedings of the 19th European Symposium on Research in Computer Security (ESORICS)*, pages 345–364, 2014.

[23] George Bissias, A Pinar Ozisik, Brian N Levine, and Marc Liberatore. Sybil-resistant mixing for Bitcoin. In *Proceedings of the 13th Workshop on Privacy in the Electronic Society*, WPES '14, pages 149–158, New York, NY, USA, 2014. ACM.

[24] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty Computation from Somewhat Homomorphic Encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417, pages 643–662. Springer Berlin Heidelberg, 2012. Series Title: Lecture Notes in Computer Science.

[25] Minghao Ruan and Yongqiang Wang. Secure and Privacy-Preserving Average Consensus. *Proceedings of the 2017 Workshop on Cyber-Physical Systems Security and Privacy*, 2017.

[26] Yuan Sun, Gang Yang, and Xing-she Zhou. A survey on run-time supporting platforms for cyber physical systems. *Frontiers of Information Technology & Electronic Engineering*, 18(10):1458–1478, 2017.

[27] Xi Zheng and Christine Julien. Verification and Validation in Cyber Physical Systems: Research Challenges and a Way Forward. In *2015 IEEE/ACM 1st International Workshop on Software Engineering for Smart Cyber-Physical Systems*, pages 15–18, 2015.

[28] Abdulmalik Humayed, Jingqiang Lin, Fengjun Li, and Bo Luo. Cyber-Physical Systems Security—A Survey. *IEEE Internet of Things Journal*, 4(6):1802–1831, 2017.

[29] Marianna Belotti, Nikola Božić, Guy Pujolle, and Stefano Secci. A Vademecum on Blockchain Technologies: When, Which, and How. *IEEE Communications Surveys Tutorials*, 21(4):3796–3838, 2019.

[30] Vitalik Buterin et al. Ethereum white paper, 2013.

[31] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. Technical Report EIP-150, Ethereum Project – Yellow Paper, April 2014.

[32] Scott Eisele, Taha Eghtesad, Nicholas Troutman, Aron Laszka, and Abhishek Dubey. Mechanisms for Outsourcing Computation via a Decentralized Market. In *Proceedings of the 14TH ACM International Conference on Distributed and Event-Based Systems (DEBS)*, Montreal, Quebec, Canada, July 2020.

[33] Scott Eisele, Michael Wilbur, Taha Eghtesad, Fred Eisele, Ayan Mukhopadhyay, Aron Laszka, and Abhishek Dubey. Protocol, strategies and analysis for enabling a distributed computation market for stream processing. Pending Review, October 2020.

[34] Aron Laszka, Scott Eisele, Abhishek Dubey, Gabor Karsai, and Karla Kvaternik. Transax: A blockchain-based decentralized forward-trading energy exchange for transactive microgrids. In *Proceedings of the 24th IEEE International Conference on Parallel and Distributed Systems (ICPADS)*, pages 918–927, 2018.

[35] Scott Eisele, Carlos Barreto, Abhishek Dubey, Xenofon Koutsoukos, Taha Eghtesad, Aron Laszka, and Anastasia Mavridou. Blockchains for Transactive Energy Systems: Opportunities, Challenges, and Approaches. *Computer*, 53(9):66–76, 2020.

[36] Scott Eisele, Taha Eghtesad, Keegan Campanelli, Prakhar Agrawal, Aron Laszka, and Abhishek Duby. Safe and Private Forward-Trading Platform for Transactive Microgrids. Accepted, Pending Publication, 2020.

[37] Scott Eisele, Aron Laszka, Anastasia Mavridou, and Abhishek Dubey. SolidWorx: a resilient and trustworthy transactive platform for smart and connected communities. In *2018 IEEE International Conference on Blockchain (Blockchain 2018)*, Halifax, Canada, July 2018.

[38] Scott Eisele, Istvan Mardari, Abhishek Dubey, and Gabor Karsai. Riaps: Resilient information architecture platform for decentralized smart systems. In *2017 IEEE*

20th International Symposium on Real-Time Distributed Computing (ISORC), pages 125–132, Toronto, ON, Canada, May 2017. IEEE.

[39] Scott Eisele, Geoffry Pettet, Abhishek Dubey, and Gabor Karsai. Towards an architecture for evaluating and analyzing decentralized fog applications. In *2017 IEEE Fog World Congress (FWC)*, pages 1–6, Oct 2017.

[40] Vladimir Oleshchuk. Internet of things and privacy preserving technologies. In *2009 1st International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace Electronic Systems Technology*, pages 336–340, 2009.

[41] Tom Randall. The way humans get electricity is about to change forever. Available at: https://www.bloomberg.com/news/articles/2017-09-13/the-way-we-get-electricity-is-about-to-change-forever, June 2015.

[42] Timo Lehtola and Ahmad Zahedi. Solar energy and wind power supply supported by storage technology: A review. *Sustainable Energy Technologies and Assessments*, 35:25–31, 2019.

[43] Koen Kok and Steve Widergren. A society of devices: Integrating intelligent distributed resources with transactive energy. *IEEE Power and Energy Magazine*, 14(3):34–45, 2016.

[44] Farrokh A Rahimi and Ali Ipakchi. Transactive energy techniques: closing the gap between wholesale and retail markets. *The Electricity Journal*, 25(8):29–35, 2012.

[45] William Cox and Toby Considine. Structured energy: Microgrids and autonomous transactive operation. In *4th IEEE Conference on Innovative Smart Grid Technologies (ISGT)*, pages 1–6, February 2013.

[46] Ronald B Melton. Gridwise transactive energy framework. Technical report, Pacific Northwest National Laboratory, Richland, WA, 2013.

[47] Amandeep Kaur, Jitender Kaushal, and Prasenjit Basak. A review on microgrid central controller. *Renewable and Sustainable Energy Reviews*, 55:338–345, 2016.

[48] Benedikt Römer, Philipp Reichhart, Johann Kranz, and Arnold Picot. The role of smart metering and decentralized electricity storage for smart grids: The importance of positive externalities. *Energy Policy*, 50:486–495, 2012.

[49] Oben Dag and Behrooz Mirafzal. On stability of islanded low-inertia microgrids. In *2016 Clemson University Power Systems Conference (PSC)*, pages 1–7, Clemson, SC, USA, March 2016. IEEE.

[50] Donald J Hammerstrom, Ron Ambrosio, Teresa A Carlon, John G DeSteese, Gale R Horst, Robert Kajfasz, Laura L Kiesling, Preston Michie, Robert G Pratt, Mark Yao, et al. Pacific Northwest GridWise™ testbed demonstration projects; Part I. Olympic Peninsula project. Technical report, Pacific Northwest National Lab (PNNL), 2008.

[51] Lawrence Orsini, Scott Kessler, Julianna Wei, and Heather Field. How the Brooklyn Microgrid and TransActive Grid are paving the way to next-gen energy markets. In Wencong Su and Alex Q. Huang, editors, *The Energy Internet*, pages 223 – 239. Woodhead Publishing, 2019.

[52] Anselma Wörner, Arne Meeuw, Liliane Ableitner, Felix Wortmann, Sandro Schopfer, and Verena Tiefenbeck. Trading solar energy within the neighborhood: field implementation of a blockchain-based electricity market. *Energy Informatics*, 2, 2019.

[53] Yuhua Du, Hao Tu, Srdjan Lukic, Abhishek Dubey, and Gabor Karsai. Distributed microgrid synchronization strategy using a novel information architecture platform.

In *2018 IEEE Energy Conversion Congress and Exposition (ECCE)*, pages 2060–2066. IEEE, 2018.

[54] Chen Yang, Anupam A Thatte, and Le Xie. Multitime-scale data-driven spatio-temporal forecast of photovoltaic generation. *IEEE Transactions on Sustainable Energy*, 6(1):104–112, 2014.

[55] Merlinda Andoni, Valentin Robu, David Flynn, Simone Abram, Dale Geach, David Jenkins, Peter McCallum, and Andrew Peacock. Blockchain technology in the energy sector: A systematic review of challenges and opportunities. *Renewable and Sustainable Energy Reviews*, 100:143–174, 2019.

[56] Anastasia Mavridou, Aron Laszka, Emmanouela Stachtiari, and Abhishek Dubey. VeriSolid: Correct-by-Design Smart Contracts for Ethereum. In *23rd International Conference on Financial Cryptography and Data Security*, February 2019.

[57] H. Tu, Y. Du, H. Yu, A. Dubey, S. Lukic, and G. Karsai. Resilient information architecture platform for the smart grid (RIAPS): A novel open-source platform for microgrid control. *IEEE Transactions on Industrial Electronics*, pages 1–11, 2019.

[58] Huashan Chen, Marcus Pendleton, Laurent Njilla, and Shouhuai Xu. A survey on Ethereum systems security: Vulnerabilities, attacks and defenses. *arXiv preprint arXiv:1908.04507*, 2019.

[59] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. Making Smart Contracts Smarter. In *23rd ACM Conference on Computer and Communications Security*, page 254–269, New York, NY, USA, 2016. Association for Computing Machinery.

[60] Alain Brenzikofer, Arne Meeuw, Sandro Schopfer, Anselma Wörner, and Christian Dürr. Quartierstrom: A decentralized local P2P energy market pilot on a self-

governed blockchain. In *25th International Conference on Electricity Distribution*, June 2019.

[61] Naveed Ul Hassan, Chau Yuen, and Dusit Niyato. Blockchain technologies for smart energy systems: Fundamentals, challenges, and solutions. *IEEE Industrial Electronics Magazine*, 13(4):106–118, 2019.

[62] Aron Laszka, Abhishek Dubey, Michael Walker, and Doug Schmidt. Providing privacy, safety and security in IoT-based transactive energy systems using distributed ledgers. In *Proceedings of the 7th International Conference on the Internet of Things*, IoT '17, pages 13:1—-13:8, New York, NY, USA, 2017. ACM.

[63] S. Eisele, A. Dubey, G. Karsai, and S. Lukic. WIP Abstract: Transactive Energy Demo with RIAPS Platform. In *2017 ACM/IEEE 8th International Conference on Cyber-Physical Systems (ICCPS)*, pages 91–92, "Pittsburgh, PA", April 2017.

[64] Institute for Software Integrated Systems. Resilient Information Architecture Platform for Smart Grid. https://riaps.isis.vanderbilt.edu, Jan 2020.

[65] H. Tu, Y. Du, H. Yu, A. Dubey, S. Lukic, and G. Karsai. Resilient Information Architecture Platform for the Smart Grid (RIAPS): A Novel Open-Source Platform for Microgrid Control. *IEEE Transactions on Industrial Electronics*, 2019.

[66] David P Chassin, K Schneider, and C Gerkensmeyer. Gridlab-d: An open-source power systems modeling and simulation environment. In *2008 IEEE/PES Transmission and Distribution Conference and Exposition*, pages 1–5. IEEE, 2008.

[67] Peter Volgyesi, Abhishek Dubey, Timothy Krentz, Istvan Madari, Mary Metelko, and Gabor Karsai. Time Synchronization Services for Low-cost Fog Computing Applications. In *28th International Symposium on Rapid System Prototyping (RSP)*, pages 57–63, New York, NY, USA, Oct 2017. IEEE.

[68] Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, pages 173–186, Berkeley, CA, USA, 1999. USENIX Association.

[69] H Diedrich. *Ethereum: Blockchains, Digital Assets, Smart Contracts, Decentralised Autonomous Organisations*. CreateSpace Independent Publishing Platform, 2016.

[70] Oben Dag and Behrooz Mirafzal. On stability of islanded low-inertia microgrids. In *2016 Clemson University Power Systems Conference (PSC)*, pages 1–7, Clemson, SC, USA, Mar 2016. IEEE.

[71] Y. Du, H. Tu, S. Lukic, D. Lubkeman, A. Dubey, and G. Karsai. Implementation of a distributed microgrid controller on the resilient information architecture platform for smart systems (riaps). In *2017 North American Power Symposium (NAPS)*, pages 1–6, Sep. 2017.

[72] IBM ILOG CPLEX. V12. 1: User's manual for CPLEX. *International Business Machines Corporation*, 46(53), 2009.

[73] Selim Ciraci, Jeff Daily, Jason Fuller, Andrew Fisher, Laurentiu Marinovici, and Khushbu Agarwal. FNCS: a framework for power system and communication networks co-simulation. In *Proceedings of the Symposium on Theory of Modeling & Simulation - DEVS Integrative*, pages 1–8, 2014.

[74] Florian Barbieri, Sumedha Rajakaruna, and Arindam Ghosh. Very short-term photovoltaic power forecasting with cloud modeling: A review. *Renewable and Sustainable Energy Reviews*, 75:242 – 263, 2017.

[75] Donald J Hammerstrom, Ron Ambrosio, Teresa A Carlon, John G DeSteese, Gale R Horst, Robert Kajfasz, Laura L Kiesling, Preston Michie, Robert G Pratt, Mark Yao, et al. Pacific Northwest GridWise™ testbed demonstration projects; Part I. Olympic Peninsula project. Technical report, Pacific Northwest National Lab (PNNL), 2008.

[76] Srinivas Katipamula, David P Chassin, Darrel D Hatley, Robert G Pratt, and Donald J Hammerstrom. Transactive controls: A market-based GridWiseTM controls for building systems. Technical report, Pacific Northwest National Lab, Richland, WA (United States), 2006.

[77] Sijie Chen and Chen-Ching Liu. From demand response to transactive energy: state of the art. *Journal of Modern Power Systems and Clean Energy*, 5(1):10–19, 2017.

[78] Esther Mengelkamp, Johannes Gärttner, Kerstin Rock, Scott Kessler, Lawrence Orsini, and Christof Weinhardt. "Designing microgrid energy markets: A case study: The Brooklyn Microgrid". *Applied Energy*, 210:870 – 880, 2018.

[79] European Power Exchange. EPEX SPOT operational rules, 2017.

[80] F Yucel, E Bulut, and K Akkaya. Privacy Preserving Distributed Stable Matching of Electric Vehicles and Charge Suppliers. In *Proceedings of the 2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, pages 1–6, Aug 2018.

[81] HSVS Kumar Nunna and Suryanarayana Doolla. Multiagent-based distributed-energy-resource management for intelligent microgrids. *IEEE Transactions on Industrial Electronics*, 60(4):1678–1687, April 2013.

[82] Power Ledger Pty Ltd. Power Ledger Whitepaper, 2019. (Accessed on 28 August 2019).

[83] Bodhisattwa P Majumder, M Nazif Faqiry, Sanjoy Das, and Anil Pahwa. An efficient iterative double auction for energy trading in microgrids. In *2014 IEEE Symposium on Computational Intelligence Applications in Smart Grid (CIASG)*, pages 1–7, Orlando, FL, USA, Dec 2014. IEEE.

[84] J. Kang, R. Yu, X. Huang, S. Maharjan, Y. Zhang, and E. Hossain. Enabling localized peer-to-peer electricity trading among plug-in hybrid electric vehicles using

consortium blockchains. *IEEE Transactions on Industrial Informatics*, 13(6):3154–3164, Dec 2017.

[85] J. Guerrero, A. C. Chapman, and G. Verbič. Decentralized p2p energy trading under network constraints in a low-voltage network. *IEEE Transactions on Smart Grid*, pages 1–1, 2018.

[86] Jian Wang, Qianggang Wang, Niancheng Zhou, and Yuan Chi. A Novel Electricity Transaction Mode of Microgrids Based on Blockchain and Continuous Double Auction. *Energies*, 10(12), 2017.

[87] M. Khorasany, Y. Mishra, and G. Ledwich. Auction based energy trading in transactive energy market with active participation of prosumers and consumers. In *Proceedings of the 2017 Australasian Universities Power Engineering Conference (AUPEC)*, pages 1–6, Nov 2017.

[88] Shengnan Zhao, Beibei Wang, Yachao Li, and Yang Li. Integrated Energy Transaction Mechanisms Based on Blockchain Technology. *Energies*, 11(9), 2018.

[89] Chenghua Zhang, Jianzhong Wu, Yue Zhou, Meng Cheng, and Chao Long. Peer-to-Peer energy trading in a Microgrid. *Applied Energy*, 220:1–12, 2018.

[90] N W A Lidula and A D Rajapakse. Microgrids research: A review of experimental microgrids and test systems. *Renewable and Sustainable Energy Reviews*, 15(1):186–202, 2011.

[91] Taha Selim Ustun, Cagil Ozansoy, and Aladin Zayegh. Recent developments in microgrids and example cases around the world – A review. *Renewable and Sustainable Energy Reviews*, 15(8):4030–4041, 2011.

[92] Liang Zhao, Kwangho Park, and Ying-Cheng Lai. Attack vulnerability of scale-free networks due to cascading breakdown. *Physical Review E*, 70(3):35101, Sep 2004.

[93] Bing Huang, Alvaro A. Cardenas, and Ross Baldick. Not Everything is Dark and Gloomy: Power Grid Protections Against IoT Demand Attacks. In *28th USENIX Security Symposium*, pages 1115–1132. USENIX Association, August 2019.

[94] Robert M. Lee, Michael J. Assante, and Tim Conway. Analysis of the cyber attack on the Ukrainian power grid: Defense use case. Technical report, Electricity Information Sharing and Analysis Center (E-ISAC), 2016.

[95] Kim Zetter. Inside the Cunning, Unprecedented Hack of Ukraine's Power Grid. Available at: https://www.wired.com/2016/03/inside-cunning-unprecedented-hack-ukraines-power-grid/, March 2016.

[96] M Yazdanian and A Mehrizi-Sani. Distributed Control Techniques in Microgrids. *IEEE Transactions on Smart Grid*, 5(6):2901–2909, November 2014.

[97] J W Simpson-Porco, Q Shafiee, F Dörfler, J C Vasquez, J M Guerrero, and F Bullo. Secondary Frequency and Voltage Control of Islanded Microgrids via Distributed Averaging. *IEEE Transactions on Industrial Electronics*, 62(11):7025–7038, November 2015.

[98] Michael G Reed, Paul F Syverson, and David M Goldschlag. Anonymous connections and onion routing. *IEEE Journal on Selected Areas in Communications*, 16(4):482–494, 1998.

[99] Peipeng Liu, Lihong Wang, Qingfeng Tan, Quangang Li, Xuebin Wang, and Jinqiao Shi. Empirical Measurement and Analysis of I2P Routers. *Journal of Networks*, 9:2269–2278, 2014.

[100] S Jebri, M Abid, and A Bouallegue. STAC-protocol: Secure and Trust Anonymous Communication protocol for IoT. In *2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 365–370, June 2017.

[101] S J Murdoch and G Danezis. Low-cost traffic analysis of Tor. In *2005 IEEE Symposium on Security and Privacy*, pages 183–195, May 2005.

[102] Simon Barber, Xavier Boyen, Elaine Shi, and Ersin Uzun. *Bitter to Better — How to Make Bitcoin a Better Currency*, pages 399–414. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[103] Fergal Reid and Martin Harrigan. *An Analysis of Anonymity in the Bitcoin System*, pages 197–223. Springer New York, New York, NY, 2013.

[104] M Apostolaki, A Zohar, and L Vanbever. Hijacking Bitcoin: Routing Attacks on Cryptocurrencies. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 375–392, May 2017.

[105] A Biryukov and I Pustogarov. Bitcoin over Tor isn't a Good Idea. In *Proceedings of the 2015 IEEE Symposium on Security and Privacy*, pages 122–134, May 2015.

[106] Nicholas van Saberhagen. CryptoNote v 2.0. Available at: https://cryptonote.org/whitepaper{_}v2.pdf, Dec 2012.

[107] I Miers, C Garman, M Green, and A D Rubin. Zerocoin: Anonymous Distributed E-Cash from Bitcoin. In *2013 IEEE Symposium on Security and Privacy*, pages 397–411, Berkeley, CA, USA, May 2013. IEEE.

[108] David Chaum and Eugène van Heyst. Group Signatures. In *Proceedings of the 10th Annual International Conference on Theory and Application of Cryptographic Techniques (EUROCRYPT)*, pages 257–265, 1991.

[109] Ronald L Rivest, Adi Shamir, and Yael Tauman. How to Leak a Secret. In *Proceedings of the 7th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, pages 552–565, 2001.

[110] Eiichiro Fujisaki and Koutarou Suzuki. *Traceable Ring Signature*, pages 181–200. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[111] Patrick McDaniel and Stephen McLaughlin. Security and privacy challenges in the smart grid. *IEEE Security & Privacy*, 7(3), 2009.

[112] Costas Efthymiou and Georgios Kalogridis. Smart grid privacy via anonymization of smart metering data. In *1st IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pages 238–243. IEEE, 2010.

[113] Onur Tan, Deniz Gunduz, and H Vincent Poor. Increasing smart meter privacy through energy harvesting and storage devices. *IEEE Journal on Selected Areas in Communications*, 31(7):1331–1341, 2013.

[114] A. Hussain, V. H. Bui, and H. M. Kim. A Resilient and Privacy-Preserving Energy Management Strategy for Networked Microgrids. *IEEE Transactions on Smart Grid*, 2017.

[115] Daniel Friedman. *The Double Auction Market Institutions, Theories, and Evidence*. Routledge, 1 edition, June 2018.

[116] Gaurav Baranwal and Deo Prakash Vidyarthi. A fair multi-attribute combinatorial double auction model for resource allocation in cloud computing. *Journal of Systems and Software*, 108:60–76, October 2015.

[117] IHS Technology. IoT Platforms: Enabling the Internet of Things. Technical report, IHS Markit, 2016.

[118] D. P. Anderson, C. Christensen, and B. Allen. Designing a Runtime System for Volunteer Computing. In *Proceedings of the 2016 ACM/IEEE Conference on Supercomputing (SC)*, pages 33–33, 2006.

[119] Laurence Field, D Spiga, I Reid, H Riahi, and L Cristella. CMS@ home: Integrating the Volunteer Cloud and High-Throughput Computing. *Computing and Software for Big Science*, 2(1):2, 2018.

[120] Muhammad Nouman Durrani and Jawwad A. Shamsi. Volunteer Computing: Requirements, Challenges, and Solutions. *Journal of Network and Computer Applications*, 39:369–380, 2014.

[121] Rafael Brundo Uriarte and Rocco DeNicola. Blockchain-based decentralized cloud/-fog solutions: Challenges, opportunities, and standards. *IEEE Communications Standards Magazine*, 2(3):22–28, 2018.

[122] Sarah Underwood. Blockchain beyond Bitcoin. *Communications of the ACM*, 59(11):15–17, 2016.

[123] Jason Teutsch and Christian Reitwießner. A Scalable Verification Solution for Blockchains. Available at: https://people.cs.uchicago.edu/teutsch/papers/truebit.pdf, 2017.

[124] iExec: Blockchain-based decentralized cloud computing. https://iex.ec/whitepaper/iExec-WPv3.0-English.pdf. Accessed: 12-01-2019.

[125] MODiCuM GitHub. https://github.com/scope-lab-vu/MODiCuM.

[126] Scott Eisele, Taha Eghtesad, Nicholas Troutman, Aron Laszka, and Abhishek Dubey. Mechanisms for outsourcing computation via a decentralized market, 2020.

[127] Changyu Dong, Yilei Wang, Amjad Aldweesh, Patrick McCorry, and Aad van Moorsel. Betrayal, Distrust, and Rationality: Smart Counter-Collusion Contracts for Verifiable Cloud Computing. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 211–227. ACM, 2017.

[128] Mira Belenkiy, Melissa Chase, C. Chris Erway, John Jannotti, Alptekin Küpçü, and Anna Lysyanskaya. Incentivizing Outsourced Computation. In *Proceedings of the 3rd International Workshop on Economics of Networked Systems*, NetEcon '08, pages 85–90, New York, NY, USA, 2008. ACM.

[129] Hadrien Croubois. PoCo Series #2 — On the use of staking to prevent attacks. Available at: https://medium.com/iex-ec/poco-series-2-on-the-use-of-staking-to-prevent-attacks-2a5c700558bd. Accessed: 2019-12-26.

[130] Luis FG Sarmenta. Sabotage-Tolerance Mechanisms for Volunteer Computing Systems. In *Proceedings of the 1st IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 337–346. IEEE, 2001.

[131] Brendan Burns, Brian Grant, David Oppenheimer, Eric Brewer, and John Wilkes. Borg, Omega, and Kubernetes. *Queue*, 14(1):10, 2016.

[132] Dirk Merkel. Docker: lightweight Linux containers for consistent development and deployment. *Linux Journal*, 2014(239):2, 2014. Available at: https://www.linuxjournal.com/content/docker-lightweight-linux-containers-consistent-development-and-deployment.

[133] Geth: Ethereum Command Line Interface. Online, https://github.com/ethereum/go-ethereum/wiki/geth.

[134] Mick Bauer. Paranoid Penguin: An Introduction to Novell AppArmor. *Linux Journal*, 2006(148):13, August 2006.

[135] seccomp. http://man7.org/linux/man-pages/man2/seccomp.2.html. Accessed: 2020-1-12.

[136] Thanh Bui. Analysis of Docker security. *arXiv preprint arXiv:1501.02967*, 2015.

[137] Contained Advisor. Github, https://github.com/google/cadvisor.

[138] Linux container images. http://crunchtools.com/comparison-linux-container-images/. Accessed: 2019-12-26.

[139] ETH Gas Station. https://ethgasstation.info/calculatorTxV.php. Accessed: 2019-09-30.

[140] AWS Pricing Calculator. https://aws.amazon.com/lambda/pricing/, 2000. Accessed: 2019-09-30.

[141] Sharding roadmap. https://github.com/ethereum/wiki/wiki/Sharding-roadmap. Accessed: 2020-1-4.

[142] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The PARSEC benchmark suite: Characterization and architectural implications. In *17th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 72–81. ACM, 2008.

[143] S. Shekhar, A. D. Chhokra, A. Bhattacharjee, G. Aupy, and A. Gokhale. Indices: Exploiting edge resources for performance-aware cloud-hosted services. In *2017 IEEE 1st International Conference on Fog and Edge Computing (ICFEC)*, pages 75–80, May 2017.

[144] Jianbao Ren, Yong Qi, Yuehua Dai, Yu Xuan, and Yi Shi. Nosv: A lightweight nested-virtualization VMM for hosting high performance computing on cloud. *Journal of Systems and Software*, 124:137 – 152, 2017.

[145] Apache Pulsar – An open-source distributed pub-sub messaging system, 2019.

[146] Apache BookKeeper – A scalable, fault-tolerant, and low-latency storage service optimized for real-time workloads, 2019.

[147] Jay Kreps, Neha Narkhede, Jun Rao, et al. Kafka: A distributed messaging system for log processing, 2011.

[148] William Barbour, Michael Wilbur, Ricardo Sandoval, Caleb Van Geffen, Brandon Hall, Abhishek Dubey, and Dan Work. Data Driven Methods for Effective Micromobility Parking, 2020.

[149] Geoffrey Pettet, Ayan Mukhopadhyay, Mykel Kochenderfer, Yevgeniy Vorobeychik, and Abhishek Dubey. On algorithmic decision procedures in emergency response systems in smart and connected communities. In *Proceedings of the 19th Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2020, Auckland, New Zealand*, pages 1740–1752, Auckland, New Zealand, 2020. ACM.

[150] Afiya Ayman, Michael Wilbur, Amutheezan Sivagnanam, Philip Pugliese, Abhishek Dubey, and Aron Laszka. Data-Driven prediction of Route-Level energy use for Mixed-Vehicle transit fleets, June 2020.

[151] Soohwan Kim and Jonghyuk Kim. Building occupancy maps with a mixture of Gaussian processes. In *2012 IEEE International Conference on Robotics and Automation*, pages 4756–4761, Saint Paul, MN, 2012. IEEE.

[152] Geoffrey Pettet, Saroj Sahoo, and Abhishek Dubey. Towards an adaptive multimodal traffic analytics framework at the edge. In *IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops 2019, Kyoto, Japan, March 11-15, 2019*, pages 511–516, Piscataway, New Jersey, 2019. IEEE.

[153] M Mazhar Rathore, Hojae Son, Awais Ahmad, Anand Paul, and Gwanggil Jeon. Real-time big data stream processing using GPU with spark over Hadoop ecosystem. *International Journal of Parallel Programming*, 46(3):630–646, 2018.

[154] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache Flink: Stream and batch processing in a single engine, 2015.

[155] Ted Dunning and Ellen Friedman. *Streaming architecture: new designs using Apache Kafka and MapR streams*. " O'Reilly Media, Inc.", Sebastopol, CA 95472 USA, 2016.

[156] Yitian Zhang, Jiong Yu, Liang Lu, Ziyang Li, and Zhao Meng. L-heron: An open-source load-aware online scheduler for apache heron. *Journal of Systems Architecture*, 106:101727, 2020.

[157] Marisol García-Valls, Abhishek Dubey, and Vicent Botti. Introducing the new paradigm of social dispersed computing: applications, technologies and challenges. *Journal of Systems Architecture*, 91:83–102, 2018.

[158] Mary R Schurgot, Michael Wang, Adrian E Conway, Lloyd G Greenwald, and P David Lebling. A dispersed computing architecture for resource-centric computation and communication. *IEEE Communications Magazine*, 57(7):13–19, 2019.

[159] David P. Anderson. BOINC: A platform for volunteer computing. *Journal of Grid Computing*, 18(1):99–122, 2020.

[160] Cisco. Cisco annual internet report - cisco annual internet report (2018–2023) white paper. Technical report, Cisco, 2018.

[161] Mutable. Mutable: the public edge cloud, 2020.

[162] Mitch Cherniack, Hari Balakrishnan, Magdalena Balazinska, Don Carney, Uğur Çetintemel, Ying Xing, and Stan Zdonik. Scalable distributed stream processing, 2003.

[163] Scott Eisele, Taha Eghtesad, Nicholas Troutman, Aron Laszka, and Abhishek Dubey. Mechanisms for outsourcing computation via a decentralized market. In *Proceedings of the 14th ACM International Conference on Distributed and Event-based Systems*, pages 61–72, New York, NY, USA, 2020. ACM.

[164] Agostino Nuzzolo, Umberto Crisalli, Luca Rosati, and Angel Ibeas. Stop: a short term transit occupancy prediction tool for aptis and real time transit management systems. In *16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013)*, pages 1894–1899, Piscataway, New Jersey, 2013. IEEE.

[165] Michael Wilbur, Afiya Ayman, Anna Ouyang, Vincent Poon, Riyan Kabir, Abhiram Vadali, Philip Pugliese, Daniel Freudberg, Aron Laszka, and Abhishek Dubey. Impact of covid-19 on public transit accessibility and ridership, 2020.

[166] Xinshang Wang. *Online Algorithms for Dynamic Resource Allocation Problems*. PhD thesis, Columbia University, 2017.

[167] Ernest J Henley and Hiromitsu Kumamoto. *Reliability engineering and risk assessment*, volume 568. Prentice-Hall, Englewood Cliffs, NJ, 1981.

[168] Blocks, 2020.

[169] ETH Gas Station. https://ethgasstation.info/calculatorTxV.php. Accessed: 2020-10-25.

[170] Anonymous. Online appendix.

[171] James Usevitch and Dimitra Panagou. Resilient finite time consensus: A discontinuous systems approach, 2020.

[172] Ethereum Solidity. Solidity documentation, 2017.

[173] Amritraj Singh, Kelly Click, Reza M. Parizi, Qi Zhang, Ali Dehghantanha, and Kim-Kwang Raymond Choo. Sidechain technologies in blockchain networks: An examination and state-of-the-art review. *Journal of Network and Computer Applications*, 149:102471, 2020.

[174] Apache. Companies using or contributing to apache pulsar.

[175] Apache. Multi tenancy.

[176] Apache. Authentication and authorization in pulsar.

[177] Electric Choice. Electricity rates.

[178] Gerald Coley. Beaglebone black system reference manual.

[179] Apache Kafka, 2020.

[180] Roshan Sumbaly, Jay Kreps, and Sam Shah. The big data ecosystem at LinkedIn. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 1125–1134, 2013.

[181] Sanjeev Kulkarni, Nikunj Bhagat, Maosong Fu, Vikas Kedigehalli, Christopher Kellogg, Sailesh Mittal, Jignesh M Patel, Karthik Ramasamy, and Siddarth Taneja. Twitter Heron: Stream processing at scale. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 239–250, 2015.

[182] Muhammad Hussain Iqbal and Tariq Rahim Soomro. Big data analysis: Apache storm perspective. *International journal of computer trends and technology*, 19(1):9–14, 2015.

[183] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. Apache spark: a unified engine for big data processing. *Communications of the ACM*, 59(11):56–65, 2016.

[184] Paolo Bonacquisto, Giuseppe Di Modica, Giuseppe Petralia, and Orazio Tomarchio. A procurement auction market to trade residual cloud computing capacity. *IEEE Transactions on Cloud Computing*, 3(3):345–357, 2014.

[185] Mitch Cherniack, Hari Balakrishnan, Magdalena Balazinska, Donald Carney, Ugur Cetintemel, Ying Xing, and Stanley B Zdonik. Scalable distributed stream processing. In *CIDR*, volume 3, pages 257–268, 2003.

[186] Elad Verbin. Mutable: Building the low-latency cloud, 2020.

[187] Mahmoud Ammar, Giovanni Russello, and Bruno Crispo. Internet of Things: A survey on the security of IoT frameworks. *Journal of Information Security and Applications*, 38:8 – 27, 2018.

[188] Kyle Benson, Charles Fracchia, Guoxi Wang, Qiuxi Zhu, Serene Almomen, John Cohn, Luke D'arcy, Daniel Hoffman, Matthew Makai, Julien Stamatakis, et al. Scale: Safe community awareness and alerting leveraging the internet of things. *IEEE Communications Magazine*, 53(12):27–34, 2015.

[189] Dale Willis, Arkodeb Dasgupta, and Suman Banerjee. ParaDrop: A Multi-tenant Platform to Dynamically Install Third Party Services on Wireless Gateways. In *Proceedings of the 9th ACM workshop on Mobility in the evolving internet architecture*, pages 43–48. ACM, 2014.

[190] RIAPS. https://riaps.isis.vanderbilt.edu/redmine/projects/riaps.

[191] Sei Ping Lau, Geoff V Merrett, Alex S Weddell, and Neil M White. A traffic-aware street lighting scheme for smart cities using autonomous networked sensors. *Computers & Electrical Engineering*, 45:192–207, 2015.

[192] Pedro Garcia Lopez, Alberto Montresor, Dick Epema, Anwitaman Datta, Teruo Higashino, Adriana Iamnitchi, Marinho Barcellos, Pascal Felber, and Etienne Riviere. Edge-centric computing: Vision and challenges. *SIGCOMM Comput. Commun. Rev.*, 45(5):37–42, September 2015.

[193] EPRI. Transforming smart grid devices into open application platforms. Electric Power Research Institute Report 3002002859, July 2014.

[194] William R. Otte, Abhishek Dubey, Subhav Pradhan, Prithviraj Patil, Aniruddha Gokhale, Gabor Karsai, and Johnny Willemsen. F6COM: A Component Model for Resource-Constrained and Dynamic Space-Based Computing Environment. In *Proceedings of the 16th IEEE International Symposium on Object-oriented Real-time Distributed Computing (ISORC '13)*, Paderborn, Germany, June 2013.

[195] Abhishek Dubey, Gabor Karsai, and Nagabhushan Mahadevan. A Component Model for Hard Real-time Systems: CCM with ARINC-653. *Software: Practice and Experience*, 41(12):1517–1550, 2011.

[196] Abhishek Dubey, Gabor Karsai, and Nagabhushan Mahadevan. Formalization of a component model for real-time systems. Technical report, Vanderbilt University, 04/2012 2012.

[197] BeagleBoard.org - black.

[198] Cities skyline. http://www.citiesskylines.com.

[199] OpenDHT. https://github.com/savoirfairelinux/opendht.

[200] Zopkio: A functional and performance test framework for distributed systems.

[201] Service discovery overview.

[202] Joe Stubbs, Walter Moreira, and Rion Dooley. Distributed Systems of Microservices Using Docker and Serfnode. *2015 7th International Workshop on Science Gateways (IWSG)*, pages 34–39, 2015.

[203] Apache helix - service discovery.

[204] Introduction - consul by HashiCorp.

[205] Michael Hoefling, Florian Heimgaertner, and Michael Menth. Advanced communication modes for the publish/subscribe c-DAX middleware. In *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*, pages 1309–1314. IEEE, 2016.

[206] Cyber-secure data and control cloud for power grids. http://www.cdax.eu/.

[207] What is DDS? http://portals.omg.org/dds/what-is-dds-3/.

[208] O. M. G. Specification. RTPS specification. *Object Management Group pct/07-08-04*, 2007.

[209] Kai Beckmann and Olga Dedi. sDDS: A portable data distribution service implementation for WSN and IoT platforms. In *Intelligent Solutions in Embedded Systems (WISES), 2015 12th International Workshop on*, pages 115–120. IEEE, 2015.

[210] Simone Cirani, Luca Davoli, Gianluigi Ferrari, Remy Leone, Paolo Medagliani, Marco Picone, and Luca Veltri. A scalable and self-configuring architecture for service discovery in the internet of things. *IEEE Internet of Things Journal*, 1(5):508–521, 2014.

[211] Akram Hakiri, Pascal Berthou, Aniruddha Gokhale, and Slim Abdellatif. Publish/subscribe-enabled software defined networking for efficient and scalable IoT communications. *IEEE Communications Magazine*, 53(9):48–54, 2015.

[212] R. B. Melton. Gridwise transactive energy framework. Technical report, Pacific Northwest National Laboratory, 2013.

[213] Anastasia Mavridou and Aron Laszka. Designing secure Ethereum smart contracts: A finite state machine based approach. In *Proceedings of the 22nd International Conference on Financial Cryptography and Data Security (FC)*, February 2018.

[214] Gul A Agha. Actors: A model of concurrent computation in distributed systems. Technical report, Massachusetts Institute of Technology, Artificial Intelligence Lab, 1985.

[215] Ananda Basu, Bensalem Bensalem, Marius Bozga, Jacques Combaz, Mohamad Jaber, Thanh-Hung Nguyen, and Joseph Sifakis. Rigorous component-based system design using the BIP framework. *IEEE Software*, 28(3):41–48, 2011.

[216] Jonatan Bergquist, Aron Laszka, Monika Sturm, and Abhishek Dubey. On the design of communication and transaction anonymity in blockchain-based transactive microgrids. In *Proceedings of the 1st Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers (SERIAL)*, pages 3:1–3:6. ACM, 2017.

[217] Joseph Sifakis. Rigorous system design. *Foundations and TrendsÂ® in Electronic Design Automation*, 6(4):293–362, 2013.

[218] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. Nusmv 2: An opensource tool for symbolic model checking. In *International Conference on Computer Aided Verification*, pages 359–364. Springer, 2002.

[219] Christel Baier, Joost-Pieter Katoen, and Kim Guldstrand Larsen. *Principles of model checking*. MIT press, 2008.

[220] Douglas C Schmidt, Jonathan White, and Christopher D Gill. Elastic Infrastructure to Support Computing Clouds for Large-scale Cyber-Physical Systems. In *2014 IEEE 17th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC)*, pages 56–63, 2014.

[221] Apache Software Foundation. Apache Storm. http://storm.apache.org/.

[222] Apache Software Foundation. Apache Spark. http://spark.apache.org/.

[223] Leonardo Neumeyer, Bruce Robbins, Anish Nair, and Anand Kesari. S4: Distributed stream computing platform. In *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*, pages 170–177. IEEE, 2010.

[224] Marco Iansiti and Karim Lakhani. The truth about blockchain. https://hbr.org/2017/01/the-truth-about-blockchain, January 2017. (accessed on 08/30/2017).

[225] CoinMarketCap. Bitcoin (BTC) price, charts, market cap, and other metrics. https://coinmarketcap.com/currencies/bitcoin/, August 2017. (accessed on 08/30/2017).

[226] CoinMarketCap. Ethereum (ETH) $381.84 (3.83%). https://coinmarketcap.com/currencies/ethereum/, August 2017. (accessed on 08/30/2017).

[227] Ali Dorri, Salil S Kanhere, Raja Jurdak, and Praveen Gauravaram. Blockchain for IoT security and privacy: The case study of a smart home. In *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pages 618–623, 2017.

[228] Aafaf Ouaddah, Anas Abou Elkalam, and Abdellah Ait Ouahman. Towards a novel privacy-preserving access control model based on blockchain technology in IoT. In *Europe and MENA Cooperation Advances in Information and Communication Technologies*, pages 523–533. Springer, 2017.

[229] Konstantinos Christidis and Michael Devetsikiotis. Blockchains and smart contracts for the internet of things. *IEEE Access*, 4:2292–2303, 2016.

[230] Yoichi Hirai. Formal Verification of Deed Contract in {Ethereum} Name Service. https://yoichihirai.com/deed.pdf, Nov 2016.

[231] Yoichi Hirai. Defining the Ethereum Virtual Machine for Interactive Theorem Provers. In *1st Workshop on Trusted Smart Contracts, in conjunction with the 21st*

International Conference of Financial Cryptography and Data Security (FC), Apr 2017.

[232] Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Anitha Gollamudi, Georges Gonthier, Nadim Kobeissi, Aseem Rastogi, Thomas Sibut-Pinote, Nikhil Swamy, and Santiago Zanella-Béguelin. Short Paper: Formal Verification of Smart Contracts. In *11th ACM Workshop on Programming Languages and Analysis for Security (PLAS), in conjunction with ACM CCS 2016*, pages 91–96, October 2016.

[233] Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. Making smart contracts smarter. In *23rd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 254–269. ACM, October 2016.

LIST OF ABBREVIATIONS

BFT Byzantine fault-tolerant

CPS cyber physical systems

DL distributed ledger

DLT distributed ledger technology

IoT Internet of Things

MPC multi-party computation

MSCPS multi-stakeholder cyber physical systems

TES transactive energy systems