A Proof-of-Trust Consensus Protocol for Enhancing Accountability in Crowdsourcing Services

Jun Zou, *Member, IEEE,* Bin Ye, *Member, IEEE,* Lie Qu, *Member, IEEE,* Yan Wang, *Senior Member, IEEE,* Mehmet Orgun, *Senior Member, IEEE* and Lei Li, *Senior Member, IEEE*

Abstract—Incorporating accountability mechanisms in online services requires effective trust management and immutable, traceable source of truth for transaction evidence. The emergence of the blockchain technology brings in high hopes for fulfilling most of those requirements. However, a major challenge is to find a proper consensus protocol that is applicable to the crowdsourcing services in particular and online services in general. Building upon the idea of using blockchain as the underlying technology to enable tracing transactions for service contracts and dispute arbitration, this paper proposes a novel consensus protocol that is suitable for the crowdsourcing as well as the general online service industry. The new consensus protocol is called "Proof-of-Trust" (PoT) consensus; it selects transaction validators based on the service participants' trust values while leveraging RAFT leader election and Shamir's secret sharing algorithms. The PoT protocol avoids the low throughput and resource intensive pitfalls associated with Bitcoin' s "Proof-of-Work" (PoW) mining, while addressing the scalability issue associated with the traditional Paxos-based and Byzantine Fault Tolerance (BFT)-based algorithms. In addition, it addresses the unfaithful behaviors that cannot be dealt with in the traditional BFT algorithms. The paper demonstrates that our approach can provide a viable accountability solution for the online service industry.

Index Terms—Accountability; Blockchain; Byzantine Agreement; Crowdsourcing; Consensus; PoT; PoW; PoS

1 Introduction

1.1 Background

In the last decade, an increasing number of businesses have transformed their business models by going online. In recent years, the process of transformation has accelerated and businesses have entered into the cloud computing era, where virtually turning anything into an online service (XaaS) has become a distinct possibility [7]. A notable example is the emergence of crowdsourcing sites. A crowdsourcing site enables efficient matching of service supplies and demands on a large scale that can never be achieved by the traditional off-line service market. While technology development and changes in business models are steaming ahead, the infrastructure that supports the integrity and accountability of business services is seriously left behind. The infrastructure here refers to the foundation that provides the functions of accounting for evidence, auditing and dispute arbitration.

The fundamental problem is that the traditional business infrastructure is built around a face-to-face, manual model in an offline environment with a centralized authority to supervise certain aspects of a business conduct. This kind of a model is not suitable for the Internet and cloud service environments any more, e.g., crowdsourcing sites. This is because the Internet is inherently distributed and services are mainly transacted automatically by software agents.

In a distributed environment, neither a service provider's version of the monitoring of service contract execution nor a consumer's one can be taken as the source of truth. If a dispute between the provider and the consumer arises during the execution of a service contract, it is very difficult to resolve it as the provider and the consumer may each have their own version of logs or arguments. Therefore, there is an urgent need for an infrastructure that upholds accountability in online service environments. Without such an infrastructure, service participants cannot be assured of the fairness of service transactions and the accountability of service delivery, which will seriously inhibit the healthy growth of service economy in the cloud computing era.

Accordingly, we outline the basic requirements of such accountability that enables service accountability.

Req1: The infrastructure must be decentralized to some extent, without the presence of a solely centralized authority, as a centralized authority has its obvious limitations with regards to scalability, security, fairness and objectivity [10];

Req2: The service contract should be publicly visible to all service participants. The obligations and the associated service activities must be clearly specified [38];

Req3: There should be a common and immutable log of service activities during contract execution for the contracted parties;

Req4: There should be an arbitration mechanism to resolve disputes between a service provider and a consumer. The arbitration mechanism should be nonpartisan and fair, yet it needs to provide an incentive to honest arbiters while keeping the overall arbitration cost low.

With an effective supporting infrastructure that is inherently distributed without a centralized authority, service participants can be assured of the fairness of service transactions and the accountability of service delivery. Hence it is imperative to build such a service infrastructure that can ensure fairness and represent the consensus of the majority of the service participants. In particular, this infrastructure should help address the accountability concerns of business services, providing transparency of service obligations, traceability of service activities, monitoring of contract executions, liability assignment in case of obligation violation, and finally, the arbitration of a contract when a dispute arises.

- *J. Zou, B. Ye and L. Qu are with the Department of Computing, Macquarie University, Sydney, Australia*
  E-mail: jun.zou@students.mq.edu.au
- *Y. Wang and M. Orgun are with Macquarie University.*
- *L. Li is with Hefei University of Technology.*

Manuscript revised March 15, 2017.

1.2 Motivations, Problems and Challenges

Here we take a crowdsourcing service as an example to highlight the accountability issues and flesh out the problem statements. A crowdsourcing site provides an online market for service providers and service consumers to match their supplies and demands respectively [37]. Typically a service consumer can publish tasks, and different service providers can bid for those tasks. Furthermore, service providers can also publish their service offerings and wait for service consumers to enter into a deal. Crowdsourcing sites like CrowdFlower or Mechanical Turk can have over millions of service providers and service consumers. In general, the crowdsourcing sites are centrally operated by the owner of the sites. The site owners set out the governance rules and they normally take bonds from both the service providers and the service consumers. Rules for service transactions and dispute arbitrations are all set and enforced by the site owner.

The major problem of this centralized governance model is that it lacks mechanisms to hold each party accountable in a fair and objective way. There are three typical scenarios on crowdsourcing sites.

(1) An unfaithful site owner or operator can abuse his privileges. For example, he may hold a bond too long by deliberately postponing the release of certain tasks;
(2) Some dishonest service consumers may solicit a lot of solutions from service providers, and then they may copy the solutions or ideas without paying the service providers, and some even pirate the intellectual properties of the service providers;
(3) Some service providers may also deliver an unsatisfactory service once a consumer enters into a deal.

Currently, all the transaction data are kept by the site owner. It cannot be told if the data are genuine in the first place or have not been modified since then. When a dispute arises, it is all up to the site operator to determine which party is at fault. Some crowdsourcing operators also enforce a fine against the party at fault when they arbitrate a dispute. Even for parties who do not accept the site operator's arbitration and wish to go through the normal legal process to resolve the matter, they would find it hard to gather any evidence.

In order to address these problems, a distributed accountability infrastructure needs to be built. To this end, the architecture of the Bitcoin crypto-currency [28] can be used as a reference model to build such an infrastructure. In blockchain based applications, e.g., Bitcoin, the most critical technology is the consensus protocol, which enables the establishment of the source of truth of an open network in the absence of a central authority. Bitcoin uses "Proof-of-Work (PoW)" as the underlying consensus protocol. While Bitcoin's PoW is suitable for a crypto-currency application, the pitfalls of low throughput, intensive resources associated with Bitcoin's PoW prohibit extending the Bitcoin platform to cater for a wide range of applications. More concretely, Bitcoin's PoW only allows an average of 1 transaction per second [31]. In addition, the PoW heavily depends on significant power consumption of participants. A report [6] estimates that Bitcoin's electricity consumption of the Bitcoin network may draw over 14 Gigawatts of electricity by 2020, which is equivalent to all of Denmark's consumption. By contrast, other main stream consensus protocols such as Proof-of-Stake (PoS) [26], Paxos-based consensus [24] and BFT-based consensus [18] have either weak security, poor fairness or low scalability issues.

The weaknesses of the existing consensus algorithms discussed above present a significant challenge to implement a practical service accountability infrastructure. In order to build a practical accountability infrastructure solution for the crowdsourcing environment, certain trade-offs need to be made, with the aim of achieving the following requirements in addition to the aforementioned four basic requirements (**Req1-Req4**).

Req5: The consensus protocol must be able to provide low latency and high throughput;
Req6: The infrastructure should support millions of participants;
Req7: The consensus protocol should manage unfaithful and Byzantine faults in an open environment;
Req8: The consensus process should be impartial and fair.

1.3 The Contributions

This paper is an extension of the paper "A peer-to-peer dispute arbitration protocol based on a distributed service contract management scheme" published in the ICWS 2016 proceedings [39] with major innovations in both the architecture and the protocol. The ICWS2016 paper[1] outlines an architecture that meets the aforementioned four basic requirements (**Req1-Req4**) of a distributed accountability infrastructure. It adapts the blockchain technology to a totally new area, where tracking cloud service obligations, monitoring contract execution and dispute resolution in a distributed fashion are the major concerns.

This paper further addresses the additional requirements (**Req5-Req8**) in the accountability infrastructure for a large scale service environment like crowdsourcing. It focuses on the consensus protocol, which is the core of the blockchain platform that determines the security, consistency, performance, scalability and fairness of the overall system. The main contributions of the extended paper are summarized below:

1. We present a practical blockchain-based solution to cater for both the functional and non-functional aspects of requirements for a service accountability infrastructure.
2. We propose a "Proof-of-Trust" (PoT) consensus protocol that integrates a trust component to meet the practical requirements in service industry, that is, together with the incentive measures, to address the unfaithful behaviors that quite often occur in an open, public service network.
3. In traditional blockchain environments, the validation and the block creation activities of the consensus process are normally conducted by the same set of nodes. But this is neither efficient nor secure in the crowdsourcing case. We therefore present a novel approach that separates the transaction validation and block recording to two different groups. Such an arrangement can better trade off between centralization and decentralization, and security and fairness.
4. We propose a hybrid blockchain solution which utilizes a consortium blockchain as the underlying deployment architecture, yet the transaction validation of the consensus protocol is performed through an open, public network environment, which exhibits the fairness and impartiality properties of a public blockchain.
5. Finally we conduct experiments to compare and contrast our PoT consensus protocol to other consensus approaches. We demonstrate that the PoT consensus protocol can outperform those consensus protocols, achieving 100% accuracy with sound performance, security and scalability.

1.4 The Structure of the Paper

The remainder of this paper is structured as follows. Section 2 introduces the related work in accountability, blockchain and consensus. In Section 3 we briefly introduce the service contract management scheme proposed in [39]. Section 4 outlines the overall framework of the Proof-of-Trust consensus protocol. Next in Section 5 we analyse the properties of PoT, focusing on *agreement*, *validity*, *liveness*, *performance*, *scalability*, *fainess* and discuss some attack scenarios. Section 6 presents experiments, analysis and discussion of the experiment results in a crowdsourcing context. Finally, we summarize our contributions and discuss future research directions in Section 7.

2 Related Work

As our paper crosses the topics of accountability, consensus protocol and blockchain, we present a brief literature review for each topic in order to set the stage for the discussion of our PoT consensus protocol in Section 3.

1. The winner of the Best Student Paper Award of IEEE ICWS 2016

2.1 Distributed Consensus

In 1982, Lamport published his seminal paper "The Byzantine Generals Problem" [15]. In his paper, Lamport proved that under the condition of synchronous communication, there must be at least $3f+1$ nodes in order to reach a consensus for the non-faulty nodes, with at most f "Byzantine" fault nodes. In contrast to the so called "crash recovery" fault which does not impact the rest of the system, "Byzantine" fault means that even when a node fails with an arbitrary and inconsistent behavior, it may continue interacting with the rest of the system, or collude in order to devise more malicious output [32].

In [11], Fischer *et al.* prove that even with one faulty node present, a deterministic algorithm cannot achieve consensus in a distributed environment using asynchronous communication. It means that it is impossible to find a consensus protocol that can guarantee *Agreement*, *Validity* and *Liveness* at the same time in an asynchronous environment where the message delays are not bounded.

The first consensus protocol that addresses the "Byzantine" fault is the Practical Byzantine Fault Tolerance (PBFT) protocol, proposed by Castro and Liskov in 1999 [5]. PBFT achieves consensus in an environment where communication is asynchronous, but message delays are bounded, and at most $f < n/3$ of the servers are Byzantine servers.

Other BFT tolerance consensus protocols such as Zyzzyva [13], FaB Paxos [18], XFS [17], provide different trade-offs on the number of Byzantine nodes which can be tolerated, as well as the efficiency of the communication. But overall, BFT-based consensus protocols are suitable in small scale, state machine replication scenarios.

Another type of consensus protocols is based on Paxos proposed by Lamport in [14]. Paxos is a deterministic algorithm that works in asynchronous systems and tolerates $f < n/2$ "crash recovery" failures, where f is the number of the fault nodes. Paxos sacrifices *Termination* for correctness, which is the *Agreement* and *Validity* properties. But the protocol can be applied in practice by simply setting up a longer timeout value. The Paxos protocol has inspired researchers to develop many Paxos like consensus protocols, like Google's Chubby, Yahoo's Zookeeper, etcd, Raft and others [1].

The problem with the Paxos protocol is that it is difficult to understand, and implement [24]. Ongaro *et al.* propose an easily understood consensus protocol called Raft in [24]. They show that Raft's efficiency is equivalent to Paxos, but it has a more understandable structure than Paxos and also provides a better foundation for building practical systems. In order to enhance understandability, Raft separates the key elements of consensus, such as leader election, log replication, and safety, and it enforces a stronger degree of coherence to reduce the number of states that must be considered.

2.2 Service Accountability

We refer the reader to the literature [16] for a detailed review on *accountability*. Traditionally, an accountability infrastructure is built upon the best practices standardized in the Infrastructure Technical Information Library (ITIL) [25] and policy governance frameworks such as COBIT [34] and SOA Governance [3]. The existing solutions as seen in service level agreement (SLA) and QoS monitoring [19], business activity monitoring (BAM) solutions [8] all rely on a centralized service to monitor the overall health of IT services. While these solutions serve traditional on-premise IT services well, they fall short in today's distributed cloud environment. In a vast distributed environment like the Internet, any centralized management solution will expose itself to the issues of scalability, single-point of failure and hacking. More importantly, a centralized authority can hardly escape from the influence of the government agencies, various interest groups or powerful institutes, hence it is prone to loosing openness and fairness.

2.3 Blockchain and Blockchain Consensus

The most prominent example of a decentralized infrastructure on the Internet is the Bitcoin system. In 2009, Nakamoto published the Bitcoin paper [28], in which a peer-to-peer electronic cash system was proposed. A year after, Nakamoto released the implementation of the Bitcoin system. The Bitcoin system broadcasts transactions to all the nodes in the network. Each node collects the new transactions into a block, and then works on finding a difficult "Proof-of-Work" for its block, which is called the "mining" process. The mining process involves scanning for a value when it is hashed with SHA-256, and the result begins with a number of zero bits. Since the average scanning work required is exponential in the number of zero bits required but can be verified by simply executing a single hash, it is a perfect approach to presenting a "Proof-of-Work" (PoW). The number of zero bits is used as a parameter to adjust the difficulty, i.e., the average time that a block is found, which normally is around 10 minutes. When a node finds a Proof-of-Work, it broadcasts the block to all the nodes. The other nodes accept the block only if all transactions in it are valid and not already spent. Once accepted, each node links the block with the previously accepted blocks by using the previous hash, thus forming a chain of blocks, which is called the blockchain. The blockchain functions like a transaction ledger, and together with the mining process, Nakamoto's Bitcoin protocol sorts the critical double-spending problem in crypto-currency without a central authority, with the assumption that the majority of the CPU power in the Bitcoin network is controlled by honest nodes that are not cooperating to attack the network. As such, the longest blockchain represents the consensus of the transaction history.

Saito and Yamada [27] show that blockchain is a probabilistic state machine in which participants can never commit on decisions. They suggest that Bitcoin is not suitable where the finality of decisions is needed. In [12], Garay *et al.* formally describe and analyze the core of the Bitcoin protocol. In their analysis, they have proved that Bitcoin's consensus protocol is a probabilistic Byzantine Agreement (BA) protocol, which can guarantee the *Agreement* property with an overwhelming probability when the adversarial power is below 50%. However, when the adversarial power is close to 50%, the *Validity* property can be shown with a constant probability, but not as overwhelming as required. This means that the consensus mechanism in Bitcoin does not always provide correct results when the dishonest nodes possess nontrivial computing power.

The most widely criticized focal point on Bitcoin's PoW consensus protocol is the significant amount of energy consumed on the mining process. According to Deetman, Bitcoin could consume as much electricity as Denmark by 2020 [6]. Such a resource intensive nature prevents Bitcoin's style of PoW consensus from being used in other application areas.

Tschorsch and Scheuermann [33] present a comprehensive survey on Bitcoin technology and a thorough review on the existing Proof-of-X consensus protocols, pointing out the strengths and weaknesses of each consensus scheme. They conclude that it remains unclear which Proof-of-X approach is most promising to actually improve Bitcoin, and which one will survive in practice.

In [2], Antonpoulos categorizes three different types of Bitcoin-like systems:

The first type of systems alter the parameters of the "Proof-of-Work" generation, such as hashing algorithms, average time for mining a block, and coin supply limit. Examples are LiteCoin, DogeCoin and Freicoin. The problem of this kind of approaches is that it may jeopardize security and transaction validity as Bitcoin's PoW parameter setting is a careful trade-off between security and resource consumption, and between validity and performance. For example, in order to improve the transaction throughput, most "alt coins" systems adopt the practice of altering the average time for mining a block. This will sacrifice the validity properties of the blockchain as Garay *et al.* have pointed out in [12].

Eyal *et al.* [9] present Bitcoin-NG (Next Generation), a new blockchain protocol designed to scale. Bitcoin-NG achieves this performance improvement by decoupling Bitcoins blockchain operation into two planes: leader election and transaction serialization. However, Bitcoin-NG does not address the energy consumption issue of the PoW mining process.

The second type of systems alter the consensus forming approach from "proof of work" to "proof of stake" (PoS), or using a combination of PoW and PoS. Instead of requiring users to do a certain amount of power-intensive hashing "work", PoS requires users to own a certain stake of the currency in order for them to mine new coins, and asks users to prove the ownership of a certain amount of currency to avoid the heavy power consumption of Bitcoin mining, as well as mitigating the 51% attack risks inherent from Nakamoto's original assumption. Examples of such systems are Peercoin, Blackcoin and NXT.

However, the security and Byzantine Agreement properties of PoS have not been rigorously analysed in the literature. Poelstra argued that a distributed consensus from PoS is impossible [26]. He reaffirmed his view again in 2015 and stated that it in general suffers the nothing at stake attack problem, where block generators have nothing to lose by voting for multiple blockchain histories, which leads to the consensus never resolving problem [26]. This is aggravated from the lack of a cost barrier in working on several chains. Another downside of PoS is that it lacks "fairness" as the richest participants are always given the easiest mining puzzle [22].

The third type of Bitcoin like systems are called "alt chain" [2], in which the crypto-currency is not the primary concern and the chain is altered for other purposes. Systems like Ethereum[2] and Hyperledger Fabric[3] are gaining momentum by extending Bitcoin's distributed consensus mechanism to areas outside of crypto-currency. However, we have not seen any existing works on the distributed accountability infrastructure for service computing.

Sharples and Domingue [30] suggest the idea of Proof of Intellectual Work, which uses blockchain as a vehicle to record evidence of intellectual work. They also mention that reputation could be mined through a proof-of-stake algorithm, but no details of the protocol is given.

Min *et al.* [21] propose a permissioned blockchain framework for supporting instant transaction and dynamic block size. Natoli and Gramoli [23] present the Blockchain anomaly that prevents users from executing a transaction based on the current state of the blockchain. Their investigation results reinforce our belief that consensus protocols like PoW and PoS do not ensure deterministic agreement and therefore cannot be used in a private chain setting, where strong consistency is a prerequisite.

Our literature review found that none of the existing consensus algorithms from both the public blockchain and the consortium/private blockchain satisfies the non-functional aspect of requirements for a large scale service platform. The consensus protocols of public blockchains either have too low throughput, like Bitcoin's PoW, or lack security, like PoS. On the other hand, the consensus protocols of the consortium or the private blockchain typically lack scalability, and they cannot support an environment that involves tens of thousands of participants.

3 Towards a Practical Accountability Infrastructure

3.1 The Distributed Service Contract Management Scheme

In [39], Zou *et al.* proposed a distributed service contract management scheme (DSCMS) that consists of a static component and a dynamic component. In the static component, rather than using a centralized registry which may lead to the issues of poor scalability and a single point of failure, our approach is to keep a local service registry for each participant in order to enable the publication of service offers and service discovery. In the dynamic component, the service provider will broadcast the service offer in the network, and each participant will receive the broadcast message and record the service offer in their local registry. Hence the service consumer can perform service discovery in their own registry. Together with previous work on formal contract specifications [38], this work addresses the requirements **Req1** and **Req2** given in Section 1.2. To address **Req3**, Bitcoin's blockchain is extended from a simple coin transaction ledger to a versatile service interaction public ledger, recording the activities from both the provider and the consumer during the execution of service contracts.

In the dynamic component, through the "Proof-of-Work" mining process, the consensus can be established amongst the honest participants on the execution logs of the service contract as long as the collective computation power of the honest nodes exceeds 50% of that of the whole network. The dishonest participants cannot tamper the honest participants' blockchains. Thus the source of truth for service contract execution can be established even in a peer-to-peer environment where trust is scarce.

Zou *et al.* [39] also propose a peer-to-peer dispute arbitration protocol to resolve any disputes that may arise during the service contract execution. They address **Req4** by leveraging the blockchain and the underlying Byzantine Agreement as a vehicle to build consensus amongst honest parties while keeping the overall cost low in the system, coupling with a majority function to achieve a reasonable level of accuracy in the final judgement.

While the work reported in [39] outlines a framework for distributed service contract management and presents a dispute arbitration protocol, the resource intensive nature and low efficiency of the PoW consensus protocol may render it impractical in its implementation in a large scale service environment. In addition, most of the existing online service platforms are designed, implemented and operated with a centralized architecture and it is not realistic to rebuild them with a fully decentralized one from scratch. We need to explore a more practical approach in implementing the accountability infrastructure for the online service industry.

3.2 Motivation for a New Blockchain Consensus Protocol

Generally speaking, Blockchain can be categorized into three different types of deployment models according to the founder of Ethereum Buterin: *public blockchain*, *consortium blockchain* and *private blockchain*. A public blockchain is also called a "permissionless" blockchain, which means anyone can have access to the blockchain and participate in the consensus process. A consortium blockchain is a blockchain where the consensus process is controlled by a set of pre-selected nodes, which typically belong to multiple organizations. Access to the blockchain is also controlled by the permission settings. A consortium blockchain is partly decentralized [4]. A private blockchain is fully controlled by a single organization. Both consortium blockchain and private blockchain are also called "permissoned" blockchain. Systems like Bitcoin and Ethereum are mainly designed to be used as a public blockchain. The Fabric platform under the Hyperledger project is designed as a "permissioned" blockchain, which can be used to create a consortium blockchain or a private blockchain.

While the above three categories of the consensus algorithms can address the downside of PoW, they all have their own weaknesses, which are summarized below:

I. "Proof-of-Stake" in general suffers the "Nothing at Stake problem, where block generators have nothing to lose by voting for multiple blockchain histories, which leads to consensus never resolving [26]. This is aggravated from the lack of a cost barrier in working on several chains. Another downside of PoS is that it lacks "fairness" as the richest participants are always given the easiest mining puzzle [22].

II. Paxos-based consensus and Byzantine fault tolerance (BFT) algorithms normally lack scalability, so they are only suitable

2. http://ethereum.org
3. http://hyperledger.org

for small scale distributed environments. When the number of the participant nodes increases, the performance of the consensus process will degrade exponentially.

3.3 Design Principles and Objectives of the Accountability Infrastructure

3.3.1 Accountability Infrastructure Principles

We situate our discussion in a crowdsourcing context. The principles of building an accountability infrastructure for crowdsourcing are:

1) **PR1:** Maintain a source of truth for service transactions. Service transactions should be logged in an immutable and traceable way for service providers and consumers who have gone into a service contract.
2) **PR2:** Address the practicality issues in implementation, *i.e.*, adapting to the constraints of the existing service platform in terms of functional and non-functional requirements.
3) **PR3:** Enable the service platform's transition to a more decentralized architecture.

3.3.2 Design Objectives

In a crowdsourcing site, typically there are over millions of service providers and consumers. Therefore, the first design objective is that the consensus protocol must provide scalability that can support over millions of users. Moreover, the crowdsourcing site may run in an open, Internet environment, which exposes it to adversary attacks and collusions. Hence, our second design objective is that our consensus protocol should tolerate Byzantine faults, as well as unfaithful faults. The third design objective is that the consensus protocol should provide eventual consistency, which means that the blockchain that keeps the service transactions should not permanently fork under any circumstances.

3.4 A Hybrid Blockchain Architecture for the Accountability Infrastructure

Guided by the principles **PR1-PR3**, and keeping the above objectives in mind, we change the blockchain architecture underpinning the accountability infrastructure as described in [39]. In particular, we need to change the pure public blockchain architecture and more importantly remove the PoW consensus mechanism, as these are the main factors inhibiting the performance and causing the intensive resource consumption of Bitcoin. In doing the architectural design, we need to trade-off the architectural concerns pertaining to the blockchain architecture as listed below.

Trust vs trustless: One of the major features of blockchain is the so called "trustless" property [36], having the ability of maintaining consensus in a trustless environment without the aid of a central authority. Therefore, the current convention is to treat the blockchain architecture as antagonistic to a trust mechanism. This treatment appears to be dogmatic. A trust mechanism, if leveraged properly, can relieve some of the overhead of achieving consensus in a totally trustless environment.

Centralization vs decentralization: Another highlighted feature of blockchain is "decentralization", which is frequently taken to the extreme by some blockchain practitioners. A good example is the demise of the "The DAO" project [35]. "The DAO" is a crowdfund project aiming at building a fully decentralized autonomous organization that automatically manages an investment fund. However, shortly after the release of the DAO, a vulnerability was exploited by a hacker, which caused the loss of 60 million US dollars of the crowfunder's fund. Due to the fully decentralized architecture, no one could stop hacking at the time. In the end, the Ethereum community had to adopt a centralized approach – the hard fork approach – to reclaim the lost funds from the hacker. The DAO lessons show that decentralization is not a panacea. On the other hand, the blockchain architecture is not equivalent to a decentralized architecture. For example, the architecture of a consortium blockchain and a private blockchain is partially decentralized and partially centralized respectively.

Permissioned vs permissionless: Currently the consensus process of a blockchain is either permissioned-based or permissionless-based. This may present a significant challenge for a consensus process to meet multiple objectives like scalability, security and performance. An optimal approach is to divide the consensus process into different phases, and examine which phase may require permissioned participants and which phase may be appropriate with permissionless participants. In this way, through a combination of permissioned and permissionless participants, a balance of multiple objectives can be achieved.

Security vs performance: Security and performance are the two major blockchain architectural concerns that blockchain designers need to properly deal with. In Bitcoin's design, Nakamoto favors security over performance, as security is crucial for a crypto-currency application. While a lot of designers of the "altcoins" modify Bitcoin's PoW parameters, or use a higher performance consensus protocol like PoS to replace PoW, what they really do is to trade-off security for performance. It is imperative to have a consensus protocol strong in both security and performance.

Considering the requirements of a crowdsourcing service site, we make the following architectural design decisions in order to meet our objectives:

AD1: We use a hybrid blockchain architecture which consists of a permissioned-base consortium chain, and dynamic transaction validation groups in the open crowdsourcing network. The consortium members can be crowdsourcing operators, regulators, notary or other stakeholders. The members of the transaction validation group are selected from the registered users of the crowdsourcing platform based on their trust values kept in each consortium member's trust database;

AD2: Security is the most important concern to be addressed in the consensus process. The consensus protocol should tolerate Byzantine faults as well as unfaithful faults.

AD3: The consensus process is divided into four phases. Phase 1 is the leader election phase in the consortium. The proven Raft [24] leader election algorithm will be used in the first phase. Phase 2 is for the leader to nominate transaction validators and obtain approval from other consortium members through a voting mechanism. Phase 3 is for the selected transaction validation group to validate the transactions to be included in the next block. Phase 4 is for the consortium members to package the validated transactions into a block and link it into the consortium blockchain.

AD4: The consensus protocol uses incentives to motivate crowdsourcing users to participate in the transaction validation process.

Based on the above architectural decisions, we provide the architecture overview of the accountability infrastructure as shown in Fig. 1.

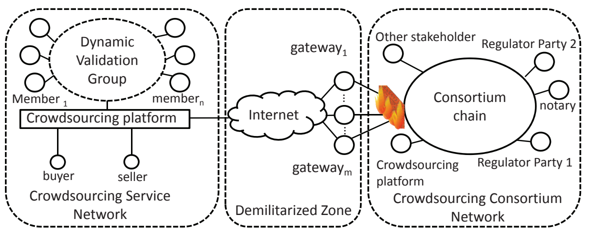


Fig. 1. Architecture Overview of the Accountability Infrastructure

As illustrated in Fig. 1, the leftmost block shows the crowdsourcing service network, which is an open, public network through which all service consumers and providers can register and participate in service transactions. The rightmost block shows the crowdsourcing consortium network, which is a private network, consisting of the consortium members. The consortium members can include the crowdsourcing site operator, various types of regulators, notaries or other stakeholders. Each consortium member has a consortium ledger management node and a gateway node. The ledger management nodes choose validators from the open crowdsourcing network, whereas the gateway nodes communicate with the validator nodes. The gateway nodes situate in the dimilitarized zone (DMZ), providing isolation of the private consortium network from the open Internet environment. The dynamically chosen validating nodes in the public crowdsourcing network achieve consensus on the chosen transactions. The consensus results will pass over to the ledger management nodes through the gateway nodes. The ledger management nodes vote on the transactions chosen by the majority of the validating nodes, just to mitigate the risk of selecting unfaithful validating group members. Next, the leader of the ledger management nodes decides the transaction sequences and packages the transactions in a block and broadcasts it to the consortium network. In the end, the gateway nodes broadcast the new block to the crowdsourcing network. The hybrid blockchain architecture is illustrated in Fig. 2.

4 The Proof-of-Trust (PoT) Consensus Protocol

The Proof-of-Trust (PoT) protocol is a consensus protocol designed for the hybrid blockchain architecture.

4.1 Basic Definitions and Assumptions

Definition 1. **Hybrid Blockchain Architecture for Accountability Infrastructure (HBAAI)**: An HBAAI system, is a tuple $\langle P, M, G, A, B \rangle$, where P is a set of nodes in a public crowdsourcing network; G is a set of gateway nodes and M is a set of ledger management nodes, $C = G \bigcup M$, C is a set of nodes in a consortium network; $G \bigcap M = \emptyset$ and $|G| = |M|$; where A is a consensus protocol and B is a shared ledger, *i.e.,* a blockchain managed by the consortium network.

We refer the reader to [39] for the blockchain related concept definitions. When a transaction is recorded in ledger B, we say that it is confirmed. We assume that each node keeps a local buffer for unconfirmed transactions. Transactions are broadcast across consortium nodes C, and through gateway nodes G, are broadcast across ledger management nodes M as well.

Definition 2. **Byzantine Node**: A network node that exhibits an arbitrary error behavior, such as crashing, disobeying protocol rules or even sabotaging behaviors.

Definition 3. **Unfaithful Node**: A network node that disobeys the protocol rules by engaging in collusion activities in order to gain self-benefits.

Note that the Byzantine nodes and unfaithful nodes can overlap.

In order to make the protocol tolerate Byzantine and unfaithful faults, we set the parameters of public crowdsourcing network P as follow.

a1. $|P| = n$, *i.e.*, there will be n nodes in the public network;
a2. $n \geq 3b + 1$, where b is the number of Byzantine nodes;
a3. $n \geq 5u + 1$, where u is the number of unfaithful nodes, $u \leq b$;
a4. Communication on P is asynchronous with a message timeout value bound on t_1.

We set the above boundary conditions as assumptions for the fault tolerance proof (see Sections 5.1 and 5.2). The parameters for the consortium network C are:

b1. $|C| = 2m$, *i.e.*, there will be an equal number m of gateway nodes G and ledger management nodes M;
b2. The number of the gateway nodes $m \geq 3f + 1$, and the number of the ledger nodes satisfies $m \geq 3f + 1$, where f is the number of the Byzantine nodes;
b3. Also the number of the gateway nodes satisfies $m \geq 5u + 1$, and the ledger nodes $m \geq 5u + 1$, where u is the number of the unfaithful nodes, $u \leq f$;
b4. Communication on C is asynchronous with a message timeout value bound on t_2.

Definition 4. **Agreement**: A consensus outcome is in an agreement when honest nodes (non Byzantine and unfaithful nodes) all agree on the same set of valid transactions $T, T \neq \emptyset$, which are recorded in an immutable blockchain B.

Definition 5. **Validity**: A consensus outcome is valid when the agreed transactions in T are all from the honest nodes.

Definition 6. **Liveness**: A consensus process is live if it can always achieve an agreement status in a bounded time.

Definition 7. **View**: A view is a tuple of configuration information that identifies a PoT cycle, $v = \langle i, G, M, V \rangle$, where i is the leader, G and M are defined in Def 1, which are a set of gateway nodes and a set of ledger management nodes respectively, and V is the validator group.

4.2 Consensus Process Description

The consensus process diagram is illustrated in Fig. 3, and the consensus sequence interaction diagram is illustrated in Fig. 4.

1). **Phase 1 - Leader election for the ledger management.**

We adopt the proven Raft leader election algorithm to elect the leader of the consortium's ledger management group. The members of the ledger management group have three roles: *leader*, *candidate* and *follower*. Each node waits for a timeout value, and if no leader vote request has been received, it becomes a candidate and then sends out a vote request $(i, VoteRequest)pr_i$, where i is the candidate node id. The message is signed with the private key of node i, denoted by pr_i. If it receives responses $(j, AckVote)pr_j$ from the majority of ledger management nodes, the candidate becomes a leader. In the response message, j is the follower id and pr_j stands for signing with its private key. The leader will lead the consensus process until its term has expired. When the leader's term has expired, a new leader election process starts again. Within the leader's term, if there is no normal message exchange between the leader and the ledger management nodes, the leader needs to send out a heartbeat message to other ledger management nodes, otherwise other ledger management nodes can compete for the leader role if the timeout is up.

2). **Phase 2 - The ledger management leader nominates a service transaction validation group.**

As per Algorithm 1, the leader first constructs a list of validator candidates based on the following criteria:

(2.1) Service parties whose trust value is greater than a predefined threshold x;
(2.2) Trust value is calculated based on a predefined trust model;
(2.3) The service party is not currently in the current buffered transactions;
(2.4) The leader sends the validator candidate list $ValidatorList$ to the other followers to vote;
(2.5) For each validator candidate, the followers check their own trust database, and then vote *yes* or *no* and broadcast their votes $(j, Votes)pr_j$ to the consortium network, where j is the follower's node id. See Algorithm 2 for this logic.
(2.6) The leader node i counts the votes and chooses v members who receive the majority votes, and then form the validation group. If the number of the required members is not enough, the leader adds more parties in it and restarts the voting process. When the number is enough, the leader broadcasts the validator member list $(h, ValidatorList)pr_i$ to the consortium network, where h is the current height of the blockchain.
(2.7) If the leader's trust database is empty, i.e., in the initial bootstrap stage, the leader uses some predefined third-parties approved by the consortium as the validation group.

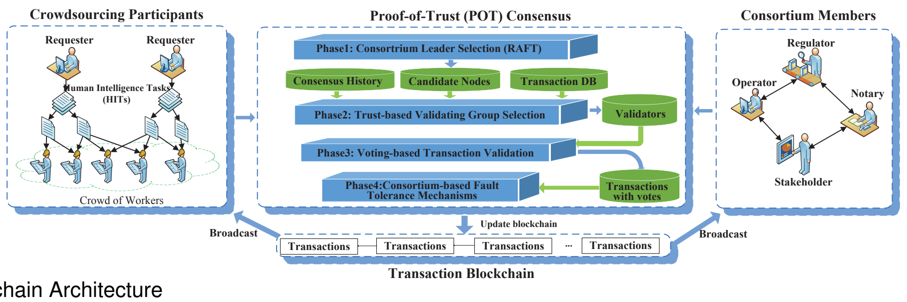


Fig. 2. The Hybrid Blockchain Architecture

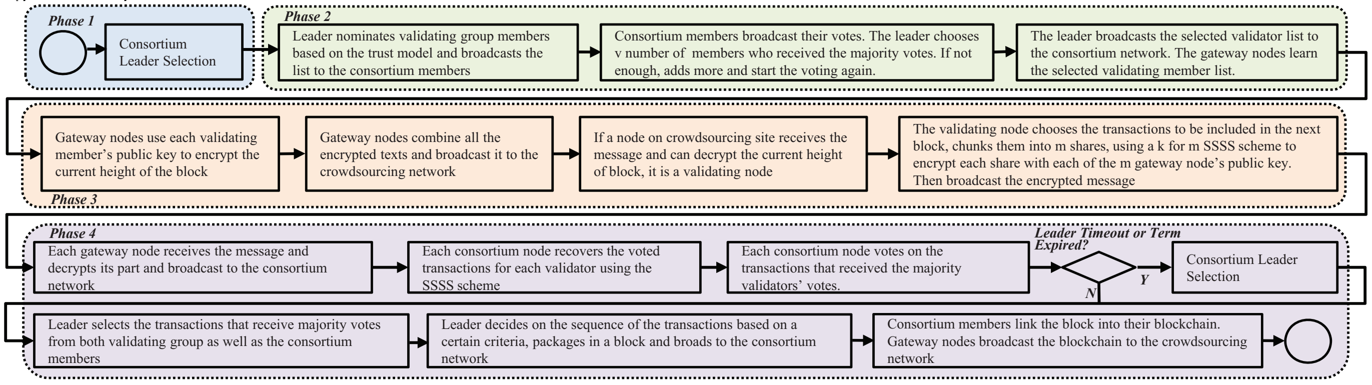

Phase 1
Consortium Leader Selection
Phase 2
Leader nominates validating group members based on the trust model and broadcasts the list to the consortium members
Consortium members broadcast their votes. The leader chooses v number of members who received the majority votes. If not enough, adds more and start the voting again.
The leader broadcasts the selected validator list to the consortium network. The gateway nodes learn the selected validating member list.
Gateway nodes use each validating member's public key to encrypt the current height of the block
Gateway nodes combine all the encrypted texts and broadcast it to the crowdsourcing network
If a node on crowdsourcing site receives the message and can decrypt the current height of block, it is a validating node
The validating node chooses the transactions to be included in the next block, chunks them into m shares, using a k for m SSSS scheme to encrypt each share with each of the m gateway node's public key. Then broadcast the encrypted message
Phase 3
Phase 4
Each gateway node receives the message and decrypts its part and broadcast to the consortium network
Each consortium node recovers the voted transactions for each validator using the SSSS scheme
Each consortium node votes on the transactions that received the majority validators' votes.
Leader Timeout or Term Expired?
N
Y
Consortium Leader Selection
Leader selects the transactions that receive majority votes from both validating group as well as the consortium members
Leader decides on the sequence of the transactions based on a certain criteria, packages in a block and broads to the consortium network
Consortium members link the block into their blockchain. Gateway nodes broadcast the blockchain to the crowdsourcing network

Fig. 3. The PoT Consensus Process

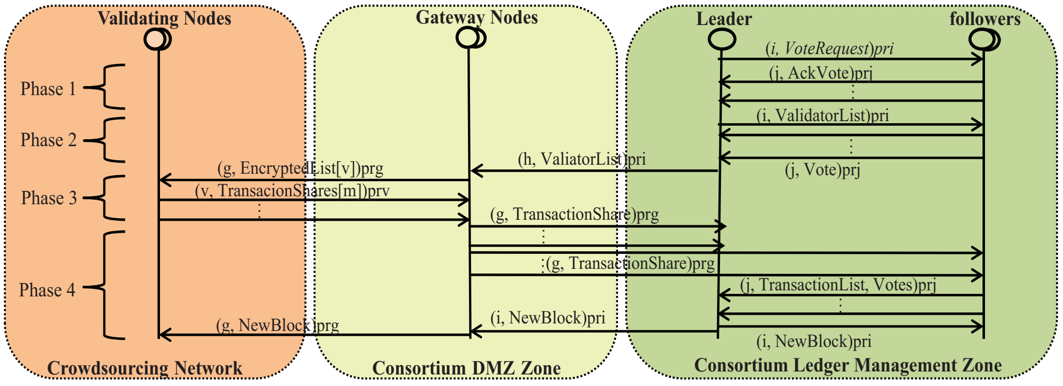


Fig. 4. The PoT Consensus Interaction Sequence

3). **Phase 3 - The transaction validation group members vote for the transactions that should fill in the next block**

(3.1) The gateway nodes receive the leader broadcast message $(h, ValidatorList)pr_i$ and use each validating member's public key pk_v to encrypt the current height of the block h. The encrypted text is hpk_v. Then they combine all the encrypted texts into an array $EncryptedList[v]$ and broadcast the message $(g, EncryptedList[v])pr_g$ to the crowdsourcing network, where g is the gateway node id. Algorithm 3 describes the gateway role's logic.

(3.2) If a node on the crowdsourcing site receives the message and can decrypt the current height of the block, it is a validating node.

(3.3) The validating node chooses the transactions to be included in the next block, and chunks them into m shares, using a k for m Shamir's Secret Sharing Scheme (SSSS) [29] to encrypt each share with the gateway node's public key pk_g. Then it broadcasts the encrypted message $(v, (TransactionShares[m])pk_g)pr_v$ where m is the number of the gateway nodes, v is the validator ID. See Algorithm 4 for the validator node's logic.

4) **Phase 4 - Ledger Management Voting and Book Keeping**
The consortium members recover the validators' transactions using the SSSS scheme, and then cast a vote to each transaction based on their transaction buffer. Finally, the leader counts the votes and packages the transactions that pass both the majority of the validator group and the consortium ledger management nodes in a new block and links it into the consortium blockchain. Algorithms 1, 2 and 4 cover the logic in Phase 4.

(4.1) Each gateway node receives the message $(v, (TransactionShares[m])pk_g)pr_v$, decrypts its part and broadcasts message $(v, g, TrasactionShare)pr_g$ to the consortium network, where v is the original validator id, and g is the gateway node id.

(4.2) Each consortium ledger management node recovers the voted transactions $TransactionList$ for each validator using the SSSS scheme.

(4.3) Each consortium ledger management node votes on the transactions that have received the majority validators' votes and broadcasts message $(l, TransactionList, Vote)pr_l$, where l is the ledger management node's id, and $Vote$ is the ledger node's vote.

(4.4) If the leader is timeout or the leader term has expired, the leader selection process restarts.

(4.5) The leader receives the broadcast message and selects the transactions that receive the majority votes from both the validating group as well as the consortium's ledger management nodes.

(4.6) The leader decides on the sequence of the transactions based on the following criteria:

a. The number of votes from both the validators as well as the ledger management nodes;
b. The transaction fees; and
c. The time spent in the buffer.

(4.7) The leader packages the ordered transactions into a Merkle tree and puts them into a block, including the previous block's hash, generating a new block hash, and broadcasts the new block $(i, NewBlock)pr_i$ to the consortium chain, where i is the leader id, $NewBlock$ is the newly generated block.

(4.8) All the nodes in the consortium chain receive the new block and broadcast a response with *agree* or *disagree* confirmation on the new block to peers in the consortium network. If the majority responses agree with the new block, each node links it to its local blockchain. Otherwise, a view change request is initiated by any ledger management nodes and the leader election process is started.

(4.9) The Gateway nodes broadcast the new block message $(g, NewBlock)pr_g$ to the crowdsourcing network.

4.3 PoT Algorithms

We here present the algorithms for all the roles that are involved in the PoT consensus process. Note that Phase 1 leader election algorithm and the view change algorithm are omitted since they are existing algorithms [5], [24].

Algorithm 1: Consortium Chain Leader Role

```
Data: phase $phase$ leaderTerm $leaderTerm$ , trust value threshold
      $trustthrehold$, consortium size $consortiumSize$, validators size
      $validatorSize$
Result: $role$
1  begin
2      $startTime$ = localtime(); $buf$ = "";
3      $block$ = ""; $voteCounts$ = 0; $validatorList$ = "";
4      while ( localtime() - $startTime \leq leaderTerm$) do
5          switch $phase$ do
6              case "2": /* Phase 2
7                  $buf$= getValidatorCandidate($trustThrehold$);
8                  broadcastToConsortiumForVote($buf$);
9                  for $m$ = 1, m++, $m \leq consortiumSize$ -1 do
10                     $memberVote$ = receiveVoteMsg($m$);
11                     $voteCounts$ = countVotes($memberVote$);
12                 $validatorList$ = countMajority($voteCounts$);
13                 if (sizeof($validatorList$)) $\leq validatorSize$ then
14                     broadcastToGateway($validatorList$);
15                     break
16             case "4": / * Phase 4
17                 $transactionList[validatorSize]$ = "";
18                 for $k$ =0 , $k$++, $k \leq validatorSize$ -1 do
19                     for $m$ = 1, $m$++, $m \leq consortiumSize$ / 2 + 1 do
20                         $buf$ = $buf$ + getGatewayShare($m$);
21                     $transactionList[k]$ =
                       recoverTransactionListUsingSSSS($buf$);
22                     broadcastTransactionListToConsortium($transactionList$[]);
23                 $transactionList$[] = "";
24                 for $m$ = 1, $m$++, $m \leq consortiumSize$ -1 do
25                     $transactionList[m]$ =
                       getConsortiumVoteTransactions($m$);
26                 $buf$ = selectMajorityTransactionVotes($transactionList$[]);
27                 $block$ = generateNewBlock($buf$);
28                 broadcastNewBlockToConsortium($block$);
29                 break;
30     return "Follower";
31 end
```

4.4 The Trust Model

Our approach can support any appropriate trust model. We here use the naive Bayes model as an example to illustrate the process. Assume that there are n crowdsourcing nodes, b out of n are Byzantine nodes, and v nodes are unfaithful nodes who may collude with others when there is an opportunity for a monetary gain. Here we assume $n \geq 3f + 1$, $n \geq 5u + 1$. The probability of picking a Byzantine node randomly is $P_b = b/n$, and picking an unfaithful node randomly is $P_v = v/n$. So $P_h = 1 - P_b - P_v$, which is the probability of picking an honest node randomly.

Assume that a user has an eigenvector $X(x_1, ..., x_n)$ that contributes to an honest or dishonest classification, where $x_1, ..., x_n$

Algorithm 2: Consortium Chain Follower Votes Validator

```
Data: phase $phase$ timeout $timeOut$, majorityNumber $MNumber$
Result: void
1  begin
2      switch $phase$ do
3          case "1": /* Phase 1
4              if $readMessage()$ == "Vote_Request" then
5                  $sendMessage()$ == "Vote_Ack";
6              break;
7          case "2": /* Phase 2
8              $buf$ = castVote(readMessageFromLeader());
9              sendMessageToLeader($buf$);
10             break:
11         case "4": /* Phase 3
12             $transactionList[validatorSize]$ = "";
13             for $k$ =0 , $k$++, $k \leq validatorSize$ -1 do
14                 for $m$ = 1, $m$++, $mleqconsortiumSize$ / 2 + 1 do
15                     $buf$ = $buf$ + getGatewayShare($m$);
16                 $transactionList[k]$ =
                   recoverTransactionListUsingSSSS($buf$);
17                 $transactionList$[] =
                   castTransactionVotes($transactionList$[]);
18                 broadcastTransactionListToConsortium($transactionList$[]);
19             break;
20 end
```

Algorithm 3: Consortium Chain Gateway Node

```
Data: phase $phase$ majorityNumber $MNumber$
Result: void
1  begin
2      switch $phase$ do
3          case "3": /* Phase 3
4              $validator$[] = ""; $h$ = 0; /* h is the current height of blockchain
5              $h$ = readMessageFromLeader($validator$[]);
6              for $i$ =1, $i$++, $i \leq sizeof(validator$[] do
7                  $msg$ = $msg$ + encryptWithValidatorPubKey($h$, $i$);
8              broadcastToCrowdsourceSite($msg$);
9              break;
10         case "4": /* Phase 4
11             for $k$ =0 , $k$++, $k \leq validatorSize$ -1 do
12                 readMessageFromValidator($msg$);
13                 $transactionShare$ = decryptMsgShare($msg$);
14                 broadcastTransactionShareToConsortium($transactionShare$);
15             break;
16 end
```

Algorithm 4: Crowdservice Validation Group Member

```
Data: phase $phase$, gatewayNumber $GNumber$
Result: void
1  begin
2      switch $phase$ do
3          case "3" /* Phase 3
4              $transactionShare[GNumber$ ] = "";
5              $msg$ = readMessageFromGateway();
6              if decryptMsg($msg$) then
7                  $transactionList$ = getTransactionList();
8                  $transactionShare$[] =
                   splitTransactionToShares($transactionList$, $GNumber$);
9                  $gatewayPubkey$[] = getGatewayPubkey($Gnumber$);
10                 encryptWithSSSS($transactionShare$[],
                   $gatewayPubkey$[]);
11                 broadcastToGateway($transactionShare$[]) ;
12             break:
13         case "4": /* Phase 4
14             $block$ = getBlockMsgFromGateway();
15             updateLocalBlockchain($block$);
16             break:
17 end
```

are properties that are independent to each other. We use naive Bayes to calculate the posteriori honest probability of each node based on a set of training data with the prior honest probability.

$$P(h|X) = \begin{cases} \frac{(P_h * \prod_{k=1}^{n} P(x_k|h))}{P(X)} & P(X) \neq 0 \\ 0 & P(X) = 0 \end{cases} \quad (1)$$

Then we only pick the nodes with $P(h|X) > P_h$. An example of X can be: ($totalTransactionAmount$, $validationTimes$, $complaintTime$), where $totalTransactionAmount$ is the total amount of all the transactions that the user has conducted on the crowdsourcing platform; $validationTime$ is the number of times when the user has participated in the validation process; and $complaintTime$ is the number of times when other users have complained the user. These three variables are independent and can be used to compute a trust value.

5 Protocol Analysis

We analyze the PoT consensus protocol based on the properties of *validity*, *agreement*, *liveness*, *performance*, *scalability*, *fairness* and *security* in general. We also discuss a number of possible attack scenarios.

5.1 Validity

Our consensus process has four phases where the first two phases are the preparation phases. Therefore, the *validity* property is only relevant to the last two phases.

In Phase 3, each validator selects the transactions in its buffer. Through Shamir's Secret Sharing Scheme, and the validator's selected transactions are passed to the consortium ledger management nodes in the end, without revealing to peer validators. Each ledger management node also checks each transaction against its transaction buffer. If it finds a match, it puts a vote on the transaction. Then it broadcasts the transactions and the votes to the consortium leader. The leader counts the transaction votes from the validators first. It first chooses the transactions that are voted by the majority of the validators, which is more than the $\frac{2}{3}$rd of the total validators. Secondly it will filter out those transactions that are not voted by at least $2f + 1$ of the total m ledger management nodes, where f is the limit of the Byzantine nodes in the consortium.

Assume that the faulty nodes (including Byzantine nodes and unfaithful nodes) are less than 50% of the total n nodes in the open crowdsourcing network. As the validators are selected based on their trust values, which is higher than the probability of randomly picking a non-faulty node of p_h, the majority of votes from the validator group can ensure the correctness of the selected transactions with an overwhelming probability. In the extreme case of selecting a validator group with majority faulty nodes, the transactions returned by the validator group will be filtered again by at least $2f + 1$ consortium members in Phase 4. Therefore it is not possible for an incorrect transaction to be selected by the leader.

Theorem 1. The PoT protocol can achieve *validity* for both the public crowdsourcing network P and the consortium network C.

Proof:

We assume that $m \geq 3f + 1$, $m \geq 5u + 1$, $u < f$, where m is the number of the ledger management nodes, f and u are the number of the maximum Byzantine nodes and the number of the maximum unfaithful nodes in the consortium respectively. We have: $2f + 1 > f + u + 1$. We consider an extreme case where the consortium has $f + u$ Byzantine and unfaithful nodes, and the transactions returned by the validator group include incorrect transactions. Since the leader chooses transactions voted by at least $2f + 1$ nodes, an incorrect transaction can at most get $f + u$ votes, which is less than $2f + 1$. Therefore, the transactions selected by the leader will always be correct.

Consider the case when there are Byzantine or unfaithful nodes in the consortium gateway nodes. As the consortium gateway nodes only act as a message passing gateway between the open crowdsourcing network and the private consortium network, they cannot determine the transactions finally selected in the new block. Also as long as the k shares used in the SSSS scheme satisfy $k \leq m - f - u$, the transactions can be reconstructed correctly by the consortium ledger management node.

Another extreme case is when the leader is either a Byzantine node or an unfaithful node. As the transactions voted by the ledger management notes are broadcast in the consortium, each member can verify the block broadcast by the leader as follows. If the block contains incorrect transactions, a consortium member can broadcast a view change request to change the leader. The new leader will count the votes again and generate a new block. The gateway nodes will only relay the new block to the open crowdsourcing network after receiving at least $2f + 1$ consortium ledger management's new block confirmation messages. Therefore, *validity* is always guaranteed.

□

5.2 Agreement

There are four different roles in our consensus process: the consortium leader node, the normal ledger management node, the consortium gateway node and the validator in the open crowdsourcing network. The *agreement* property requires all non-faulty nodes to agree on the same set of transactions, which means that the blockchain will not permanently fork.

Theorem 2. The PoT protocol can achieve *agreement* for both the public crowdsourcing network P and the consortium network C.

Proof:

The consortium leader selects the validator group member candidates, and posts to the consortium ledger management to vote. The leader also selects the transactions in a new block based on the voting results from both the validator group and the consortium ledger management nodes. In the end, the leader broadcasts the new block to the consortium network. After $2f + 1$ consortium members respond with a new block confirmation message, the new block is treated as finally confirmed in the blockchain. Therefore, there is no chance that the consortium blockchain will fork. One extreme case is when there are no transactions which fail to pass the majority validator group members' vote or the $2f+1$ consortium ledger management nodes' vote, so, the current consensus process is not successful. But this is not a problem as a new cycle of the consensus can be started. Therefore, our consensus protocol can always guarantee the *agreement* property.

□

5.3 Liveness

We assume that the communication is asynchronous, the timeout is bounded at time t_1 and t_2 in the consortium network C and the open crowdsourcing network P respectively.

For Phase 1 consensus stage, the timeout value for leader selection can be set as the same as t_1. If within the timeout value, no message is received from the leader, then the other ledger management nodes can initiate a leader election message by self-nomination. Thus the leader election can always be eventually successful. So, Phase 1 guarantees *liveness*.

In Phase 2 consensus stage, the leader chooses the validator candidates and passes to the consortium ledger management nodes to vote. Since the consortium network is a close and small network, the leader should normally expect to receive most of the responses from the consortium ledger management nodes within a reasonable timeout setting. However, in theory there is still a very rare chance that the leader continues to fail to receive enough votes from the ledger management nodes to form the validator group due to network latency or outage. This means that in theory liveness cannot be guaranteed, although this would not normally occur in a real situation, and as long as a timeout value is set properly, QoS can be guaranteed in a close, private small consortium network.

In Phase 3 consensus stage, gateway nodes inform the validator nodes, which send the transactions to the consortium network, using an SSSS encryption scheme. The message will be more likely to timeout than that of the consortium network. Along with Phase 4, it can be observed that *liveness* cannot be guaranteed. So overall, the PoT protocol cannot guarantee *liveness* if the network QoS cannot be guaranteed.

In a real application environment, depending on the network condition, an optimal setting of the size of the validator group and the message timeout value need to be worked out to ensure a smooth consensus process.

5.4 Performance

The essence of the PoT consensus protocol is to use the consensus amongst a subgroup of the nodes of the network with higher trust values to represent the consensus of the nodes in the whole network. Thus intuitively the performance of the PoT will be better than the traditional consensus protocol that involves all the nodes of the network in the consensus process. From the perspective of message exchange, assuming that the size of the validator group is v, there are equal numbers of the gateway nodes and the consortium ledger management nodes, which is m, and then the total number of the messages in a consensus process cycle is computed as follows:

Phase 1: The maximum number of exchanged messages in leader selection is: m * (m-1). This caters for the extreme case when every node sends out a request vote message to other peer nodes for self-nominating as the consortium leader.

Phase 2: This mainly involves the leader nominating validator candidates for the rest of the ledger management nodes to vote, and broadcast the voting results to the gateway nodes. Therefore, the maximum number of exchanged messages is 2 * (m-1) + m = 3 * m - 2.

Phase 3: The number of messages exchanged between the validator nodes and the gateway nodes is: 2 * v * m.

Phase 4: The number of messages exchanged between the validator nodes and the gateway nodes is 2 * $v * m$. The number of messages exchanged in the consortium network is m * m + m + $m - 1$. So, the total number of messages in Phase 4 is m^2 + 2 * $v * m$ + 2 * m-1.

In total, the number of messages exchanged is $2 * m^2 + 4 * v * m + 4 * m - 3$. This shows that the process of message exchange has $\mathcal{O}(m^2)$ complexity. Normally in a consortium network, the number of nodes should be relatively small, at around 10-20 ledger management nodes as well as gateway nodes. The number of the nodes in the validation group in the crowdsourcing network can also be limited to a reasonable size, *i.e.*, 1% of the total network nodes. Therefore the performance of the PoT protocol is better than those of most of the public blockchain platforms.

Selecting validators based on their trust values can reduce the chances of getting Byzantine nodes and the unfaithful nodes, and thus improve the efficiency of the consensus process and reduce the chance of consensus failure caused by the lack of *liveness* in the protocol.

5.5 Scalability

Traditional Paxos-based or BFT-based consensus protocols have poor scalability, and their performance degrades quickly while the number of the network nodes increases. In our protocol, since we use a hybrid blockchain architecture, the size of the consortium group and the validator group will remain relatively steady even when the number of nodes in the open network increases rapidly to well over millions of nodes. This means that the performance can remain steady even when the size of the open network increases. Therefore, the scalability of the PoT protocol is better than the consensus protocols of the pure consortium or private block chain platforms.

5.6 Fairness

The *fairness* property of a consensus protocol means that the consensus process can be assessed in three aspects. The first aspect is to check whether the power of determining the consensus outcome is concentrated within a few parties. The second is to check if the consensus process is neutral and impartial. The third aspect is to see if the protocol is resistent to collusions among the participants in the consensus process.

Unlike the most of the other consensus protocols which do not separate the roles of transaction validation and ledger management, our PoT protocol achieves the separation of powers by delegating the basic transaction validation work to a validator group in the public network, which has the power to vote on which transactions are to be included in the new block. Whereas the consortium ledger management nodes can only vote against the transaction lists that are voted by the majority of the validator group members, without the ability of adding extra transactions. In the end, the leader includes those transactions that receive more than $2f + 1$ votes from the consortium members into the new block. In this way, the PoT protocol adds more checks and balances into the consensus process, and passes the test of separation of powers in the first aspect of the fairness assessment.

Our PoT protocol achieves *fairness* by selecting the validators with higher trust values, and those who are not the direct participants in the current transactions in the buffer. Also the validators are nominated by the consortium leader based on their trust values in the leader's database, and then need to receive $2f + 1$ votes from the rest of the consortium ledger management nodes. Thus our PoT protocol passes the neutrality and impartiality test in the second aspect of fairness assessment.

Our PoT protocol also has the anti-collusion capabilities built-in as the SSSS cryptographic scheme is used in the validation process. In this way, the validators initially do not know the identities of the other peer validators, because the transaction lists are encrypted with the SSSS scheme without revealing to other validators. This can reduce the chances of collusion amongst the validators. Further collusion attack scenarios are analyzed in the next section.

Overall, our PoT protocol sustains well for all three aspects of the fairness test.

5.7 Security and Attack Scenarios

The security aspect of a consensus protocol is assessed through the commonly seen attack scenarios. We analyze some of the typical attack scenarios in the blockchain as well as the crowdsourcing context.

(1) Sybil Attack

This is a typical security attack that frequently occurs on the public blockchain platform. The attacker establishes a large number of nodes in the blockchain network and tries to influence the consensus result. In our protocol, since the validators are chosen based on their trust values, the chance for such an attack to succeed is very remote. Even when there are attacker nodes selected in the validator group, the transactions voted by the validator group will be voted again by the consortium ledger management nodes. Therefore, our PoT protocol can defend against the Sybil attack with an overwhelming probability.

(2) Collusion Attack

We separate the collusion attacks into two different classes. The first one is a collusion between the leader and some of the validators. Basically the leader nominates all of his favorite validators. However, he needs to get at least $2f + 1$ votes from the peer ledger management nodes. If the leader sends out a validator group member list without passing $2f + 1$ required votes, any ledger management node can challenge the leader by initiating a view change message to replace the leader. Even if, in the end, the leader forms a validator group with most of his allies, the transactions voted by the validators are still needed to be voted by the ledger management nodes. Thus the collusion cannot do any harm to the consensus outcome.

Also there is an incentive and punishment mechanism in our PoT protocol to encourage honest behaviors while discouraging unfaithful behaviors. The validators who honestly participate in the validation process will receive some service coins as a transaction fee. The validators who are engaged in collusion activities will be punished with a low trust value.

The second class is the collusion between the leader and the ledger management nodes. Here, our assumption is that the numbers of the Byzantine nodes and unfaithful nodes do not exceed $f + u$, where $f < (m-1)/3$ and $u < (m-1)/5$. So, even if a collusion occurs amongst the leader and the ledger management nodes, as long as the number of Byzantine nodes and unfaithful nodes is confined in the $f+u$ limit, they cannot get enough votes to include incorrect transactions. If the leader does include transactions that failed to get $2f+1$ votes, the honest ledger management nodes can initiate a view change request to change the leader.

(3) Distributed Denial of Service (DDoS) Attack

There are no effective ways to prevent a pure DDoS attack targeting the network, other than using traffic cleansing devices with large bandwidth support. However, there are some specific DDoS attacks targeting the blockchain infrastructure and the weakness of the consensus protocol. For example, some adversaries may broadcast some transactions using the SSSS scheme to conduct a DDoS attack against the gateway nodes in the DMZ. The gateway nodes need to filter out such messages based on the signature of the message senders. Only the messages signed by the members of the validator group are received by the gateway nodes.

Another DDoS attack scenario is that the adversaries may broadcast a lot of transactions in order to flood the network. The transactions may be easily conducted amongst the adversaries. The defense to this attack is to mandate a transaction fee on each transaction, thus raising the cost barrier for implementing such an attack. Also the trust evaluation process can identify such adversaries, and move them into a blacklist and block their message broadcasting accordingly.

6 Experiments

In order to evaluate whether our proposed PoT-based consensus protocol is able to satisfy the requirements **Req5-Req8**, we simulate the consensus processes of the proposed consensus protocol on a Windows 8 system with an I7 Intel Core CPU and 32GB RAM. Also, we compare the PoT-based consensus protocol with three other consensus protocols discussed below.

Ripple liked Sub-Network based Consensus: A transaction is permitted to be written into a block if the majority of the members in a randomly selected sub-network agree with the transaction.

Consortium based Consensus: A transaction is permitted to be written into a block if the majority of the members in the consortium agree with the transaction.

Joint Consensus: A transaction is permitted to be written into a block only when both the majority of the members in a randomly selected sub-network and the majority of the members in the consortium agree with the transaction.

PoT based Consensus: A transaction is permitted to be written into a block only when both the majority members in a selected trustworthy sub-network and the majority of the members in the consortium agree with the transaction.

6.1 Experiment Settings

Based on the assumptions in **Section 4**, we conduct simulations to verify whether PoT-based consensus protocol enables an accurate and efficient consensus on crowdsourcing transactions under two types of typical attacks: (1) Sybil attacks and (2) collusion attacks.

6.1.1 Settings for Sybil Attacks

First, regarding the number of Byzantine nodes, we consider the worst case where the proportion of the Byzantine nodes in each of the public crowdsourcing network and the consortium network equals $\frac{1}{3}$. Then, taking the Sybil attacks into account, we simulate the worst case where the $\frac{1}{5}$th of the nodes in the public crowdsourcing network are unfaithful and all of these nodes mount Sybil attacks by uniformly voting for a fake transaction. In addition, as consortium ledger management nodes are registered under a strict security accreditation, it is reasonable to set the proportion of Sybil nodes among them to be less than $\frac{1}{5}$. It is set as $\frac{1}{10}$ in Experiment 1 below. Based on the settings, we further evaluate our proposed POT based consensus protocol in different situations where the number of the concurrent transactions increases from 50 to 250 with a step of 50 and the number of the public crowdsourcing members increases from 1000 to 21000 with a step of 5000.

6.1.2 Settings for Collusion Attacks

Regarding the collusion attacks, we consider the worst case where the proportions of the Byzantine nodes and the unfaithful nodes in both the public crowdsourcing network and the consortium network are $\frac{1}{3}$ and $\frac{1}{5}$, respectively. As the first class of collusion attacks (i.e., a leader colludes with some of validators) and the second class of collusion attacks (i.e., a leader colludes with some ledger management nodes) co-exist in a crowdsourcing site, for each unfaithful node in the consortium network who may become a leader, we randomly select the unfaithful nodes from both the public crowdsourcing network and the consortium network to become its allies. Based on this scenario, we further test the performance of our protocol when the number of the concurrent transactions increases from 50 to 250 with a step of 50 and the number of the public crowdsourcing members increases from 1000 to 21000 with a step of 5000.

6.1.3 Other Settings

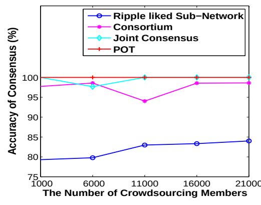
To reflect a real crowdsourcing network as much as possible, fake transactions are randomly generated and then broadcast to the public crowdsourcing network. In addition, in each round of the consensus process, 1% of the public crowdsourcing network nodes are selected as validators.

6.2 Experiment Results

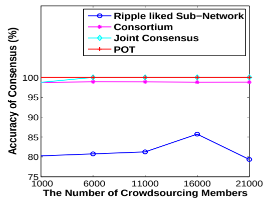
6.2.1 Experiment 1 (Performance Comparison of Different Consensus Protocols under Sybil Attacks)

Accuracy Comparison in Blocking True Transactions. Figs 5(a)-(e) plot the accuracies of different consensus protocols in blocking true transactions under Sybil attacks. We can observe that at least 3.41% and 1.18% of fake transactions are blocked by the sub-network based consensus protocol and the consortium based consensus protocol, respectively. Though the joint consensus protocol achieves relatively high accuracies in most cases, it still cannot guarantee 100% accuracy. By contrast, among all the four consensus protocols, only the PoT-based consensus protocol can always write the true transactions into the block chain, i.e., 100% accuracy in achieving consensus on any transaction. The superior performance of the POT-based consensus protocol is due to two reasons. First, the adoption of the trust model effectively prevents most of the Sybil nodes from becoming validators whose votes are broadcast to the consortium chain via the gateway nodes. Second, the consortium network requires members to register under strict security accreditation, which ensures only a few Sybil nodes can successfully register as consortium members. As such, the second-stage POT-based consensus can strongly defend against Sybil attacks mounted by the Sybil nodes from the public crowdsourcing network.

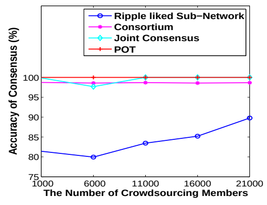
Efficiency Comparison of Different Consensus Protocols. Figs 6(a)-(e) plot the average time needed by the four consensus protocols in successfully generating a block under Sybil attacks. Among all the four consensus protocols, the consortium based consensus protocol delivers the best validity property because it generates a block once all the consortium members achieve consensus, which is an efficient procedure without extra processes. However, as we discussed in the accuracy comparison, the consortium based



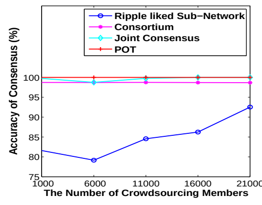
(a) Accuracy Comparison (the number of concurrent transactions is 50)



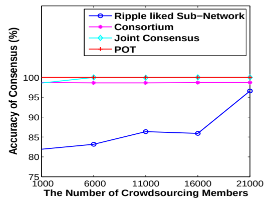
(b) Accuracy Comparison (the number of concurrent transactions is 100)



(c) Accuracy Comparison (the number of concurrent transactions is 150)

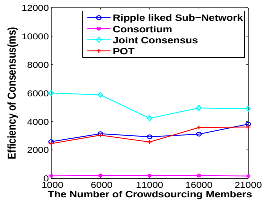


(d) Accuracy Comparison (the number of concurrent transactions is 200)

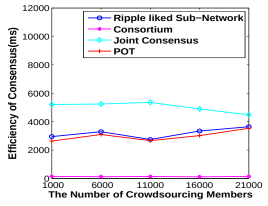


(e) Accuracy Comparison (the number of concurrent transactions is 250)

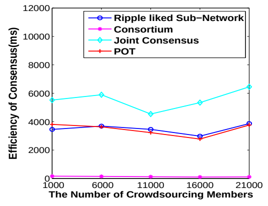
Fig. 5. The Accuracy Comparison of Different Consensus Protocols under Sybil Attacks



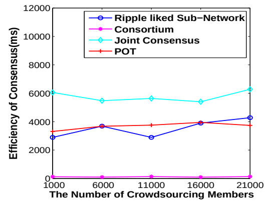
(a) Efficiency Comparison (the number of concurrent transactions is 50)



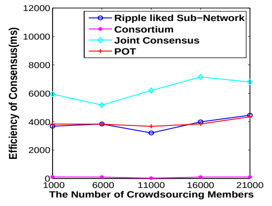
(b) Efficiency Comparison (the number of concurrent transactions is 100)



(c) Efficiency Comparison (the number of concurrent transactions is 150)

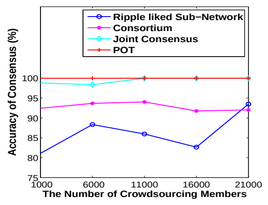


(d) Efficiency Comparison (the number of concurrent transactions is 200)

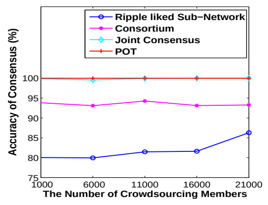


(e) Efficiency Comparison (the number of concurrent transactions is 250)

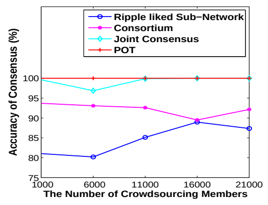
Fig. 6. The Efficiency Comparison of Different Consensus Protocols under Sybil Attacks



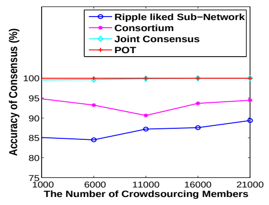
(a) Accuracy Comparison (the number of concurrent transactions is 50)



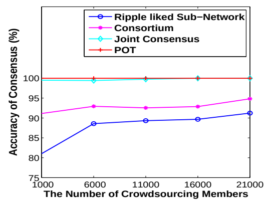
(b) Accuracy Comparison (the number of concurrent transactions is 100)



(c) Accuracy Comparison (the number of concurrent transactions is 150)

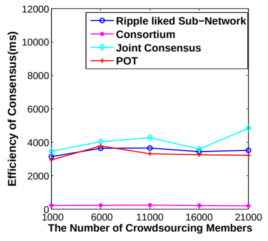


(d) Accuracy Comparison (the number of concurrent transactions is 200)

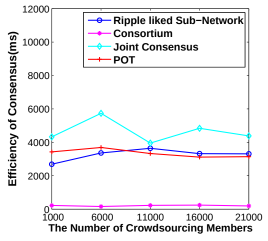


(e) Accuracy Comparison (the number of concurrent transactions is 250)

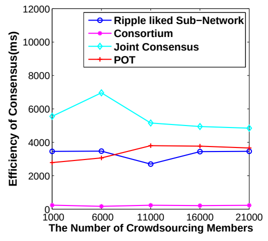
Fig. 7. The Accuracy Comparison of Different Consensus Protocols under Collusion Attacks



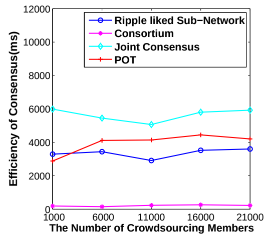
(a) Efficiency Comparison (the number of concurrent transactions is 50)



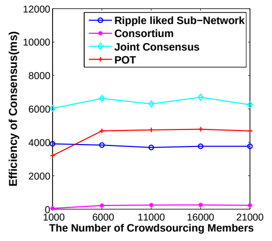
(b) Efficiency Comparison (the number of concurrent transactions is 100)



(c) Efficiency Comparison (the number of concurrent transactions is 150)



(d) Efficiency Comparison (the number of concurrent transactions is 200)



(e) Efficiency Comparison (the number of concurrent transactions is 250)

Fig. 8. The Efficiency Comparison of Different Consensus Protocols under Collusion Attacks

consensus sacrifices the validity property. Regarding the joint consensus based protocol, it is obviously not an efficient one among all the four protocols. In particular, on average, it consumes 5499ms in generating a block, which is 61.2% more than that needed by the POT-based consensus. The reason is that the joint consensus without applying a trust model wastes much time in dealing with those votes from many Sybil nodes. By contrast, under Sybil attacks, the sub-network based consensus consumes 3439.5ms to form a block on average, which is very close to that of the POT based consensus protocol, i.e., 3412ms. However, the sub-network based consensus cannot accurately form blocks where all transactions are true.

6.2.2 Experiment 2 (Performance Comparison of Different Consensus Protocols under Collusion Attacks)

Accuracy Comparison in Blocking True Transactions. Figs 7(a)-(e) plot the changes of accuracies of different consensus protocols in blocking transactions under collusion attacks. Among the four consensus protocols, only the PoT-based consensus protocol can guarantee 100% accuracy in blocking true transactions in all cases. This is because the trust model based validator selection helps prevent the unfaithful nodes from being the validators. Consequently, the votes from most of the unfaithful nodes cannot be broadcast to the consortium chain and thus cannot influence the result of consensus. Concretely, the adoption of a trust model dramatically decreases the probability that a fake transaction is broadcast to the consortium chain via the gateway nodes. Moreover, the members in the consortium chain provide the second defense that prevents the rare fake transactions bypassing the first defense from being voted for consensus. As such, the PoT-based protocol can guarantee the validity of consensus. By contrast, from Figs 7(a)-(e), we can observe that there are at least 5% fake transactions in the block chains of the sub-network based consensus and the consortium based consensus protocols. Interestingly, the joint consensus protocol achieves 100% accuracy when the size of the public crowdsourcing network grows to, *e.g.*, 16000 and 21000 public crowdsourcing nodes. This is because that the probability of the randomly selected validators can achieve consensus on a fake transaction dramatically decreases with an increasing number of the public crowdosourcing nodes.

Efficiency Comparison of Different Consensus Protocols. Figs 8(a)-(e) plot the average time needed by different consensus protocols in successfully generating a block. When facing collusion attacks, as both the sub-network based consensus protocol and the consortium based consensus protocol sacrifice the critical validity property, they can generate blocks more efficiently than the joint consensus protocol and the PoT based consensus protocol. The joint consensus protocol and PoT based consensus protocol consume a little more time in generating a block because they need to execute intermediate processes (e.g., the leader election, the validator selection, the SSSS based message broadcast and the double-stage voting process) to prevent the transactions voted by the colluding allies from being written into the block. In particular, the average time needed by the joint consensus based protocol and the PoT based consensus protocol for generating a block is 3688.5ms and 5243.2ms, respectively.

Compared to the joint consensus based protocol, the PoT-based consensus protocol saves 29.652% time in generating a block because the trust model based validator selection helps reduce the proportion of colluding public crowdsourcing network nodes in the validators, thus saving the effort of dealing with fault nodes' misleading messages and therefore increasing the efficiency of the consensus process. In addition, the double stage consensus process prevents invalid transactions from writing into the final block and thus guarantee the validity property of the consensus protocol. Moreover, from Figs 8(a)-(e), we can observe that the time needed by the POT-based consensus protocol in generating a block stays at a stable level, i.e., between 3142.8ms and 4785.4ms, though the number of public crowdsourcing nodes increases to a large number (i.e., 21000). Compared to the 15s confirmation time of the protocol in [20], our proposed PoT-based Consensus is scalable and can efficiently support millions of public crowdsourcing nodes.

Summary: Regardless of whether there are Sybil attacks or collusion attacks, our proposed POT based consensus protocol is the only one that can accurately add true transactions into the block chain in a huge public crowdsourcing network where many concurrent transactions exist (i.e., 250 concurrent transactions). Moreover, in all the cases, the POT based consensus protocol only needs around 4 seconds to successfully generate an accurate block. Thus, we claim that the POT based consensus protocol is scalable and efficient to support millions of public crowdsourcing nodes in practice. All of these further demonstrate that the PoT-based consensus protocol satisfy all the requirements **Req5-Req8**.

7 Summary and Future Work

The tamper-proof, immutable features of the blockchain technology offer great hopes in building an accountability infrastructure for the online service industry. However, most of the existing consensus protocols of public blockchain suffer either performance issues, resource overheads, security weaknesses or fairness pitfalls. On the other hand, the most of the consensus protocols of consortium/private blockchain cannot support large scale networks due to scalability limitations. In this paper, we have proposed a novel consensus protocol called Proof-of-Trust (PoT) consensus, which leverages a trust mechanism that is widely used in the online service industry to address the pitfalls and limitations of existing consensus protocols in either public or consortium/private blockchain platforms.

Our proposed PoT consensus is a hybrid blockchain architecture that integrates a consortium blockchain in an open, public service network, which brings in distributed governance and accountability to the online service industry. In our PoT consensus, we have integrated a trust component to meet the practical requirements in service industry, that is, together with the incentive measures, to tolerate Byzantine faults and address the unfaithful behaviors that quite often occur in the open, public service network. e.g., a crowdsourcing network. The PoT protocol splits the consensus process into four phases, each of which is conducted by different roles, which ensures performance and consistency of the consensus process while greatly improving scalability. By separating the powers of participants in the consensus process, our PoT protocol also promotes fairness and security. Our protocol does not provide liveness guarantee due to the FLP impossible theory. Future improvement of the protocol will add mechanism to work around the rare situation of consensus deadlock. We have conducted experiments to compare and contrast our PoT consensus protocol to three other consensus protocols. The results demonstrate that our proposed PoT consensus protocol outperforms those consensus protocols, achieving 100% accuracy with sound performance and scalability.

Overall, using the crowdsourcing environment as a motivating example, we have shown that our proposed PoT protocol can achieve *agreement*, *validity*, *performance*, *scalability*, *fairness* and *security*, thus meets the four requirements (**Req5-Req8**) listed in Section 1. To the best of our knowledge, our approach is the first one that provides a practical consensus solution for online service networks. This provides a foundation for further enhancements in the liveness guarantee of the consensus protocol and potential application of the game theory, which are the subjects of future work.

References

[1] A. Ailijiang, A. Charapko, and M. Demirbas. Consensus in the cloud: Paxos systems demystified. In *25th International Conference on Computer Communication and Networks, ICCCN 2016, Waikoloa, HI, USA, August 1-4, 2016*, pages 1–10, 2016.

[2] A. M. Antonopoulos. *Mastering Bitcoin*. OReilly, 2015.

[3] W. A. Brown, R. Laird, C. Gee, and T. Mitra. *SOA Governance: Achieving and Sustaining Business and IT Agility*. Pearson Education, 2008.

[4] V. Buterin. On public and private blockchains, 08 2015.

[5] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999*, pages 173–186, 1999.

[6] S. Deetman. Bitcoin could consume as much electricity as denmark by 2020. *Available at: http://motherboard.vice.com/read/bitcoin-could-consume-as-much-electricity-as-denmark-by-2020*, 03 2016.

[7] Y. Duan, Y. Cao, and X. Sun. Various "aas" of everything as a service. In *Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2015 16th IEEE/ACIS International Conference*, pages 1–6, June 2015.

[8] W. W. Eckerson. *Performance Dashboards: Measuring, Monitoring, and Managing Your Business*. John Wiley & Sons, 2010.

[9] I. Eyal, A. E. Gencer, E. G. Sirer, and R. van Renesse. Bitcoin-ng: A scalable blockchain protocol. In *13th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2016, Santa Clara, CA, USA, March 16-18, 2016*, pages 45–59, 2016.

[10] J. Famaey, J. Donders, T. Wauters, F. Iterbeke, N. Sluijs, B. De Vleeschauwer, F. De Turck, P. Demeester, and R. Stoop. Comparative study of peer-to-peer architectures for scalable resource discovery. In *2009 First International Conference on Advances in P2P Systems*, pages 27–33, 2009.

[11] M. J. Fischer, N. A. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[12] J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. *Advances in Cryptology - EUROCRYPT 2015 Volume 9057 of the series Lecture Notes in Computer Science*, pages 281–310, 2015.

[13] R. Kotla, A. Clement, E. L. Wong, L. Alvisi, and M. Dahlin. Zyzzyva: speculative byzantine fault tolerance. *Commun. ACM*, 51(11):86–95, 2008.

[14] L. Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, 1998.

[15] L. Lamport, R. E. Shostak, and M. C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[16] K.-J. Lin, J. Zou, and Y. Wang. Accountability computing for e-society. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 34–41. IEEE, 2010.

[17] S. Liu, C. Cachin, V. Quéma, and M. Vukolic. XFT: practical fault tolerance beyond crashes. *CoRR*, abs/1502.05831, 2015.

[18] J. Martin and L. Alvisi. Fast byzantine consensus. *IEEE Trans. Dependable Sec. Comput.*, 3(3):202–215, 2006.

[19] A. Michlmayr, F. Rosenberg, P. Leitner, and S. Dustdar. Comprehensive qos monitoring of web services and event-based sla violation detection. In *4th Middleware for Service Oriented Computing Workshop of the 10th International Middleware Conference 2009*, pages 1–6. ACM, 2009.

[20] M. Milutinovic, H. W. He, and M. Kanwal. Proof of luck: an efficient blockchain consensus protocol. *CoRR*, abs/1703.05435, 2017.

[21] X. Min, Q. Li, L. Liu, and L. Cui. A permissioned blockchain framework for supporting instant transaction and dynamic block size. In *2016 IEEE Trustcom/BigDataSE/ISPA, Tianjin, China, August 23-26, 2016*, pages 90–96, 2016.

[22] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder. *Bitcoin and cryptocurrency technologies.* Princeton University Press., 2016.

[23] C. Natoli and V. Gramoli. The blockchain anomaly. In *15th IEEE International Symposium on Network Computing and Applications, NCA 2016, Cambridge, Boston, MA, USA, October 31 - November 2, 2016*, pages 310–317, 2016.

[24] D. Ongaro and J. K. Ousterhout. In search of an understandable consensus algorithm. In *2014 USENIX Annual Technical Conference, USENIX ATC '14, Philadelphia, PA, USA, June 19-20, 2014.*, pages 305–319, 2014.

[25] J. Pengelly. ITIL service level management. *GTSLearning*, 2005.

[26] A. Poelstra. Distributed consensus from proof of stake is impossible. Technical report, 2014.

[27] K. Saito and H. Yamada. What's so different about blockchain? - blockchain is a probabilistic state machine. In *2016 IEEE 36th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 168–175, June 2016.

[28] N. Satoshi. Bitcoin: A peer-to-peer electronic cash system. *Available on: http://bitcoin.org/bitcoin.pdf*, 2012.

[29] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[30] M. Sharples and J. Domingue. The blockchain and kudos: A distributed system for educational record, reputation and reward. In *Adaptive and Adaptable Learning - 11th European Conference on Technology Enhanced Learning, EC-TEL 2016, Lyon, France, September 13-16, 2016, Proceedings*, pages 490–496, 2016.

[31] Y. Sompolinsky and A. Zohar. Secure high-rate transaction processing in bitcoin. In *Financial Cryptography and Data Security - 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers*, pages 507–527, 2015.

[32] M. Treaster. A survey of fault-tolerance and fault-recovery techniques in parallel systems. *CoRR*, abs/cs/0501002, 2005.

[33] F. Tschorsch and B. Scheuermann. Bitcoin and beyond: A technical survey on decentralized digital currencies. *IEEE Communications Surveys and Tutorials*, 18(3):2084–2123, 2016.

[34] W. van Grembergen. *Implementing Information Technology Governance: Models, Practices and Cases: Models, Practices and Cases*. IGI Global, 2007.

[35] A. van Wirdum. Ethereums dao forking crisis: The bitcoin perspective, 07 2016.

[36] G. Wood and J. Steiner. *Trustless Computing—The What Not the How*, pages 133–144. Springer International Publishing, Cham, 2016.

[37] B. Ye, Y. Wang, and L. Liu. Crowd trust: A context-aware trust model for worker selection in crowdsourcing environments. In *Web Services (ICWS), 2015 IEEE International Conference on*, pages 121–128. IEEE, 2015.

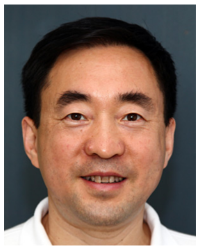
[38] J. Zou, Y. Wang, and M. Orgun. Modeling accountable cloud services based on dynamic logic for accountability. *International Journal of Web Services Research (IJWSR)*, 12(3):48–77, 2015.

[39] J. Zou, Y. Wang, and M. A. Orgun. A dispute arbitration protocol based on a peer-to-peer service contract management scheme. In *IEEE International Conference on Web Services, ICWS 2016, San Francisco, CA, USA, June 27 - July 2, 2016*, pages 41–48, 2016.

Jun Zou is currently CTO of HainaCloud Ltd. in China. He received the PhD degree in computing from Macquarie University in 2016, the BSc degree in computer science and engineering from South China University of Technology in 1988 and the MBA degree in business administration from Macquarie Graduate School of Management, Australia in 2007; He is the co-recipient of the Best Paper Award at IEEE ICEBE2006 and the Best Student Paper Award at IEEE ICWS2016.

Bin Ye is currently a PhD candidate in computer science at Macquarie University, Australia. He received his BMse degree in information management and information system from the Nanjing Tech University, P. R. China in 2011, and MMse degree in management science and engineering from the Nanjing Tech Universityg, P. R. China in 2014. His current research interests include crowdsourcing, spam recognition and learning-based recommendation. He is a student member of IEEE.

Qu Lie is currently a post doctorate research fellow in the Department of Computing, Sydney University, Sydney, NSW, Australia. He received the PhD degree in computing from Macquarie University in 2016, and MSc degree in computer science at University of Wollongong, Australia, in 2012, and BEng degree in computer science and technology at Beijing Institute of Technology, China, in 2005. His current research interests include trust computing, machine learning, blockchain and information security.



Yan Wang is currently an Associate Professor in the Department of Computing, Macquarie University, Sydney, NSW, Australia. His current research interests include trust management, web services, e-commerce and social networks. He is the co-recipient of the Best Paper Award at IEEE SCC2010 and IEEE TrustCom2012. Dr. Yan Wang serves on the Editorial Board of several international journals, including IEEE Trans. on Services Computing and Service-Oriented Computing & Applications. He is a Senior Member of the IEEE.

Mehmet A. Orgun is currently a professor at Macquarie University, Sydney, Australia. He received the BSc and MSc degrees in computer science and engineering from Hacettepe University, Ankara, Turkey in 1982 and 1985; and the PhD degree in computer science from the University of Victoria, Canada in 1991. His research interests include knowledge discovery, multi-agent systems, trusted systems and temporal reasoning. He is a Senior Member of the IEEE.

Lei Li is currently an Associate Professor of computer science and technology at Hefei University of Technology, Hefei, China. His research interests include data mining, social computing and graph computing. He has received his Ph.D. degree in computing from Macquarie University, Sydney, Australia in 2012. He is a Senior Member of the IEEE.