TriSAS: Toward Dependable Inter-SAS Coordination with Auditability

Shanghao Shi
Virginia Tech
Arlington, Virginia, USA
shanghaos@vt.edu

Yang Xiao
University of Kentucky
Lexington, Kentucky, USA
xiaoy@uky.edu

Changlai Du
Virginia Tech
Arlington, Virginia, USA
cdu@vt.edu

Yi Shi
Virginia Tech
Arlington, Virginia, USA
yshi@vt.edu

Chonggang Wang
InterDigital
Princeton, New Jersey, USA
Chonggang.Wang@InterDigital.com

Robert Gazda
InterDigital
Princeton, New Jersey, USA
robert.gazda@interdigital.com

Y. Thomas Hou
Virginia Tech
Blacksburg, Virginia, USA
thou@vt.edu

Eric Burger
Virginia Tech
Arlington, Virginia, USA
ewburger@vt.edu

Luiz DaSilva
Virginia Tech
Arlington, Virginia, USA
ldasilva@vt.edu

Wenjing Lou
Virginia Tech
Arlington, Virginia, USA
wjlou@vt.edu

ABSTRACT

To facilitate dynamic spectrum sharing, the FCC has designated certified SAS administrators to implement their own spectrum access systems (SASs) that manage the shared spectrum usage in the novel CBRS band. As a premise, different SAS servers must conduct periodic inter-SAS coordination to synchronize service states and avoid allocation conflicts. However, SAS servers may inevitably stop service for regular upgrades, crash down, or even perform maliciously that deviate from the normal routines, posing a fundamental operation security problem — the system shall be robust against these faults to guarantee secure and efficient spectrum sharing service. Unfortunately, the incumbent inter-SAS coordination mechanism, CPAS, is prone to SAS failures and does not support real-time allocation. Recent proposals that rely on blockchain smart contracts or state machine replication mechanisms to realize fault-tolerant inter-SAS coordination require all SASs to follow a unified allocation algorithm. They however face performance bottlenecks and cannot accommodate the current fact that different SASs hold their own proprietary allocation algorithms.

In this work, we propose TriSAS—a novel inter-SAS coordination mechanism to facilitate secure, efficient, and dependable spectrum allocation that is fully compatible with the existing SAS infrastructure. TriSAS decomposes the coordination process into two phases including input synchronization and decision finalization. The first phase ensures participants share a common input set while the second one fulfills a fair and verifiable spectrum allocation selection, which is generated efficiently via SAS proposers' proprietary allocation algorithms and evaluated by a customized designed allocation evaluation algorithm (AEA), in the face of no more than one-third of malicious participants. We implemented a prototype of TriSAS on the AWS cloud computing platform and evaluated its throughput and latency performance. The results show that TriSAS achieves high transaction throughput and low latency under various practical settings.

CCS CONCEPTS

• **Security and privacy → Mobile and wireless security**; • **Computer systems organization → Dependable and fault-tolerant systems and networks**.

KEYWORDS

Spectrum Access System, Spectrum Sharing, Operation Security, Inter-Server Coordination.

ACM Reference Format:
Shanghao Shi, Yang Xiao, Changlai Du, Yi Shi, Chonggang Wang, Robert Gazda, Y. Thomas Hou, Eric Burger, Luiz DaSilva, and Wenjing Lou. 2024. TriSAS: Toward Dependable Inter-SAS Coordination with Auditability. In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '24), July 1–5, 2024, Singapore, Singapore.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3634737.3645005

1 INTRODUCTION

The radio spectrum is an important and scarce resource for wireless communications. To accommodate the ever-increasing volume and variety of spectrum users in the commercial setting, spectrum regulatory agencies in the US, notably the Federal Communications Commission (FCC) and National Telecommunications and Information Administration (NTIA) have opened up previously exclusive spectrum bands for more flexible shared use by the public [18], culminating in a paradigm of dynamic spectrum sharing (DSS). The Spectrum Access System (SAS), mandated by the FCC for managing spectrum sharing in the 3.55- 3.7 GHz Citizens Broadband Radio Service (CBRS) band [24], is currently the most promising DSS-enabling technology and has started commercial deployment in 2020. At its core, SASs are implemented and operated by certified service providers, called *SAS administrators*, to allocate spectrum usage resources (location, time, bands, power) to commercial spectrum users per request while ensuring the pre-emptive access right of incumbent users (e.g., military radars, satellite stations, and other federal users). Multiple SAS administrators have been approved by the FCC for the CBRS ecosystem, including Google, Federated Wireless, Sony, Amdocs, and CommScope.

1.1 Inter-SAS Coordination Problem

The SAS administrators have been certified to manage the CBRS band usage of their customers (spectrum users) through proprietary SAS servers. They overlap in geographical service areas, a situation similar to that of current mobile network operators (MNOs). However, unlike the current MNOs where each MNO operates in its own licensed spectrum bands, all SAS administrators serve their customers in the *same* CBRS band. This brings a fundamental inter-SAS coordination problem—the SAS administrators should be coordinated so that their customers do not receive the same or significantly overlapping allocations (location, time, bands, power). Failing to address this problem leads to service de-synchronization and allocation conflicts, which could result in harmful interference between spectrum users. Unfortunately, this is strictly prohibited by the current WInnForum Standards [29] because any interference that exceeds the pre-defined threshold can disrupt the spectrum sharing service of the whole area.

However, during the operation of the SAS, some SAS servers may inevitably deviate from normal routines because of server crashes, regular system upgrades, or even malicious hacking. As a result, these SAS servers exhibit arbitrary behaviors and can be treated as Byzantine nodes. For example, SAS servers may become offline and stop service for a certain period because of regular maintenance. Malicious SAS servers may purposely modify or lie for some shared information to gain excessive spectrum resources in the shared CBRS band. To provide a sound and sustainable spectrum sharing service, the SAS system shall be able to overcome these issues and ensure that the system operates smoothly even if a portion of the server fails. More specifically, the already committed spectrum allocation transactions shall not be revoked or modified even if their proposed servers crash down, and the malicious servers shall not jeopardize the coordination procedure of the honest servers. Moreover, when these crashed or malicious servers go online, they shall be able to recover their operation status and retrieve the records from the honest servers.

The Coordinated Periodic Activities for SASs (CPAS) procedure, as specified in the Wireless Innovation Forum (WInnForum) standard [10], is the only incumbent inter-SAS coordination mechanism used in practice. This mechanism requires SAS administrators to perform a full-dump-style synchronization of SAS service states on a daily basis, from 3 AM to 6 AM U.S. Eastern Time, so that they can provide non-conflicting spectrum allocations to users the next day. However, this overnight message-exchange protocol neither considers the possible presence of malicious SASs nor meets with real-time performance expectations of the dynamic spectrum sharing service.

To tackle these issues, recent literature has proposed using the blockchain and distributed ledger technology to construct fault-tolerant SASs and inter-SAS coordination schemes [3, 12, 31, 34]. They either use the blockchain to bootstrap a distributed spectrum database [3, 12], or leverage smart contracts' flexible programming interfaces to implement spectrum allocation and other business logic [31, 32, 34]. The blockchain-based solutions can also provide auditability by maintaining an immutable ledger of spectrum operations across all participants. In [31, 32] particularly, the fault-tolerant inter-SAS coordination is fulfilled by the blockchain system's built-in Byzantine fault-tolerant (BFT) consensus, which essentially realizes a state machine replication (SMR) in that all SAS servers follow the same spectrum allocation algorithm and curate a unified ledger for all spectrum users. In this way, the non-faulty servers can synchronize and process new requests sequentially, instead of exchanging all service data in a full-dump style (as in the case of CPAS).

Limitations of Blockchain/SMR-based SAS: While providing consensus-driven security and immutable records of spectrum access, the above blockchain-based SASs and SMR-based inter-SAS coordination schemes do not represent a practical solution in the recently deployed SAS ecosystem. First and foremost, the SAS administrators are competitive commercial entities and manage their own service subscribers and adopt proprietary allocation algorithms which could be commercial secrets. As a result, there is no unified allocation algorithm among different SAS administrators, and their managed SAS servers cannot be treated as replicated state machines. Moreover, implementing complex allocation algorithms in blockchain smart contracts and executing them could face prohibitive on-chain costs, since the spectrum allocation algorithm can be highly complex, such as using graph coloring-based [33], reinforcement learning-based [1] or optimization-based approach [14] to derive a fair and effective spectrum allocation. Therefore, instead of replicating the same spectrum sharing service across different entities, it is imperative for the SASs to adopt a coordination mechanism that can ensure a unified and fair spectrum allocation among all participating SASs without having all parties pre-agree on any allocation algorithm. To avoid situations in that malicious or selfish SAS plays favoritism towards its own customers, security properties including fault-tolerance, immutability, and auditability of spectrum allocations should also be observed.

1.2 Our Solution

In this work, we propose TriSAS: a trustworthy, robust, and efficient inter-SAS coordination mechanism that achieves the above goals while being backward compatible with the existing SAS framework. TriSAS's core mission is to facilitate independent SAS servers to generate fair and auditable spectrum allocations for their users in an efficient manner while ensuring correctness against selfish and even malicious SAS servers. Catering to the heterogeneity of spectrum allocation algorithms among different SAS servers, we decompose the inter-SAS coordination into a two-phase process—the input data synchronization phase and the decision finalization phase. We use a blockchain database as a key component for the secure and efficient handling of spectrum requests from all users.

For the input synchronization phase, we require all servers to store their input requests in the blockchain database proactively. The blockchain database automatically converts the requests to input transactions and uses a reliable broadcast mechanism to deliver the transactions to all other SAS servers securely. This phase ensures that all participants share a common input set. In the decision finalization phase, each SAS server first pulls the common input set from the blockchain database and computes its allocation proposal locally via its proprietary allocation algorithm, which is stored in the blockchain database and reliably broadcasted to other parties. To avoid any SAS from playing favoritism towards its own users, we design a voting sub-phase to allow all SAS servers to agree on the best allocation proposal. We propose an allocation evaluation algorithm (AEA), which balance between the spectrum utilization rate and allocation fairness, for the SASs to evaluate different allocations following pre-agreed evaluation criteria. An allocation receiving a high score needs to boost spectrum usage efficiency and ensure allocation fairness at the same time. The servers vote for their preferred allocation according to the scores obtained from AEA. The allocation that is voted by the majority of the servers is elected as the final allocation result. TriSAS stores all the transactions in an immutable ledger for potential audits. In general, TriSAS can tolerate one-third of SAS servers being Byzantine.

We implemented TriSAS on the AWS cloud computing platform with the Bigchaindb [22] serving as the blockchain database. We deployed a network of servers in different areas across the U.S. to simulate real-world deployments of the SAS. We evaluated the performance of TriSAS with respect to different input rates and pulling intervals. We focused on the throughput and latency performance of TriSAS and the experiment results show that TriSAS can achieve scalable performance under practical settings.

In summary, this paper makes the following contributions:

- We identify the inter-SAS coordination problem and its unique challenges of achieving operation security while accommodating the diversity of allocation algorithms across distributed SAS servers in the current CBRS ecosystem.
- We propose TriSAS, a new inter-SAS coordination mechanism to address these challenges. TriSAS is resilient to Byzantine SAS servers when their number does not exceed one-third of the total population and at the same time keeps high throughput and low latency. TriSAS extends the current SAS service model and essentially ensures the safety, fairness, auditability, and efficiency of spectrum allocations and SAS operation.
- We propose a novel allocation evaluation algorithm (AEA) to evaluate the fairness and spectrum utilization efficiency of spectrum allocations from different SAS servers and help the system decide on a final selection. This algorithm can also serve as a benchmarking tool for general SAS performance evaluation and may be of independent interest.
- We implemented a prototype of TriSAS. Extensive experiment results show that TriSAS is a practical inter-SAS coordination solution in handling large input volumes with a small processing latency.

2 BACKGROUND

2.1 Spectrum Access System

The FCC has designated the Citizens Broadband Radio Service (CBRS) band [24], a 150 MHz frequency band between 3.55 GHz to 3.7 GHz, to accommodate the shared spectrum usage between federal and non-federal users [24]. The users get access to this spectrum resource according to a three-tiered framework approved by the FCC, with the incumbent federal users being positioned at the highest tier, the non-federal Priority Access License (PAL) holders at the middle tier and the non-federal General Authorized Access (GAA) users at the lowest tier. At the heart of this new shared spectrum usage paradigm is an automatic spectrum coordinator, named the Spectrum Access System (SAS), which dynamically allocates spectrum to users at various tiers and controls the operation and management of the spectrum usage. The SAS ensures that lower tier users cannot cause harmful interference to higher tier users, and at the same time boosts the spectrum utilization efficiency. SASs are implemented by different SAS administrators designated by the FCC and deployed on their SAS servers. Currently, different SAS administrators manage their own subscribers and allocation algorithms. They take spectrum access requests from their spectrum users and respond with transmission grants indicating whether the requests have been approved as the result of their spectrum allocation algorithms.

2.2 Blockchain-based SAS

The unique properties of blockchain, i.e., decentralization, transparency, and consensus-based security, make it a potential enabler to future spectrum management [28]. Recent works have proposed several blockchain-based, decentralized SASs to realize secure and verifiable dynamic spectrum access.

Ariyarathna et al. [3] propose a smart contract-based SAS that mainly focuses on creating and trading "spectrum tokens" based on Ethereum. This work adopts the SAS admins as a centralized party and does not consider the problems (coordination, fault tolerance, etc.) introduced by the current decentralized SAS settings. Zhang et al. [34] propose an enhanced smart contract-based dynamic spectrum sharing system, as well as a novel consensus mechanism. This work also incorporates the privacy-preserving consideration of users into their design. However, although this explores the use of a smart contract to implement the allocation, which results in a fault-tolerant allocation among participants, it still adopts the single SAS administrator model and fails to address the challenges introduced by decentralized SAS settings. Grissa et al. [12] introduce TrustSAS, a secure and privacy-preserving SAS that combines state-of-the-art cryptography with the blockchain technique. However, for inter-SAS coordination, TrustSAS only uses the blockchain (the global chain in their design) as a record-keeping board to accept any allocation proposals proposed by a cluster of users, who are usually

Table 1: A comparison between different blockchain-based SAS.

Scheme	Blockchain's role	Who curate blockchain	Fault-tolerant allocation	CBRS-SAS compatibility	Inter-SAS coordination	Allocation generation	Algorithm diversity
CPAS [10]	N/A	N/A	No	Yes	Yes (Full dump)	Off-chain	Yes
Ariyarathna et al. [3]	Rule/Record Keeping	Third Party	No	No	No	On-chain	No
Zhang et al. [34]	Allocation/Record Keeping	Prior Users	Yes	No	No	On-chain	No
TrustSAS [12]	Record Keeping	GAA Users	No	Yes	Yes (Classic BFT)	Off-chain	Yes
BDSAS [31, 32]	Allocation/Record Keeping	SAS Servers	Yes	Yes	Yes (Classic BFT)	On-chain	No
TriSAS	Record Keeping	SAS Servers	Yes	Yes	Yes	Off-chain	Yes

governed by a single SAS administrator. In a situation where different clusters of users (SAS administrator) have overlapped serving areas, which unfortunately is the current fact, cannot prevent a selfish cluster from proposing unfair or even fake allocations in favor of himself. Xiao et al. [32] propose BD-SAS, a fault-tolerant, decentralized SAS that uses a two-layer blockchain system where the global chain is used for spectrum regulation compliance and smart contract-based local chains are used in individual spectrum zones for automating spectrum allocation. However, BD-SAS still relies on smart contracts to perform on-chain spectrum allocation, which is neither efficient nor enables allocation algorithm diversity.

2.3 Blockchain Database

The blockchain database is an emerging technology that combines blockchain's decentralization, Byzantine fault tolerance, and data immutability properties, with distributed databases' query, high throughput, and low latency properties. Blockchain databases are mostly deployed in small-scale private networks consisting of multiple or tens of peers who know each other's identities. These networks enforce strict access control policies and only authorized entities can participate in the network. Examples include Bigchaindb [22] and Couchdb [2]. Compared to classical cryptocurrency and smart contract systems, blockchain databases are not purposed for realizing a full-fledged payment system. Instead, they are customized to handle a larger volume of data transmission and storage across different nodes, while at the same time maintaining the fault-tolerance property for common database operations. Whenever a data processing operation (e.g., insert, modify, and query) is received by a peer, it leverages the underlying reliable broadcast mechanism to forward it to all peers to ensure that every node maintains the same immutable ledger.

3 SYSTEM MODEL

3.1 System Architecture

Geographical Concepts: Following the definition of the WInnForum Standard [29] and previous works [31, 32], we consider spectrum sharing zone as a unit spectrum management geo-location areas in which SAS administrators can realize all spectrum sharing functions including spectrum request, response, allocation, and inter-SAS coordination. In practice, a zone usually refers to a US county. Different spectrum sharing zones operate independently and spectrum allocations in one zone will not affect the allocations of others. But all spectrum zones are subject to global regulations where the global refers to the entire spectrum jurisdiction such as the US continent. In this paper, we focus on the spectrum sharing system within one zone as spectrum sharing is conducted on a per-zone basis. We define five types of participants in the spectrum sharing system, as shown in Figure 1.

Spectrum Users operate one or more CBSDs (base stations) in the system. They register themselves to their SAS administrators when they first join the system and later send spectrum usage requests to get access to spectrum resources. They are expected to receive transmission grants, i.e. the decision about whether their requests are granted or not. Currently, one spectrum user only subscribes to and receives service from one SAS administrator. Within one zone, there are hundreds of CBSDs, denoted by $CBSD_k, (k = 1, 2, \cdots, m)$.

ESC Sensors is a set of radio sensors deployed by either the SAS administrators or other appointed entities that aim to detect the presence of incumbent spectrum users such as naval radars. After they detect the presence of incumbent users, they send ESC notifications to SAS servers and the servers suspend the transmission grants of CBSDs that might cause interference to the incumbents. ESC notifications usually have strict latency requirements and are processed by each SAS server locally.

SAS Servers are deployed by the SAS administrators to coordinate the spectrum usage of spectrum users. We assume there is only one server in charge within one spectrum sharing zone for each SAS administrator, denoted by $SAS_i, (i = 1, 2, \cdots, n)$, where n refers to the number of SAS administrators in the zone. In one zone, there are typically about 4-7 SAS servers/administrators and each SAS server has its own list of subscribers. The SAS servers are the core components of the spectrum access systems. They contain several key functions including database functions, coordination functions, and proprietary allocation algorithms. Within one server, the database function stores all the essential records including system inputs, intermediate results, and final results. For a blockchain database, the database function stores all the transactions in an immutable ledger. The allocation algorithm, which is considered to be owned by a SAS administrator who is unwilling to expose its details because of commercial and intellectual property reasons, is in charge of generating spectrum allocations by taking input sets from the database function. The coordination function is in charge of keeping state synchronization across SAS servers. It interacts with the other two functions within its server and the other coordination functions within other servers. It is the core component of our inter-SAS coordination mechanism.

Regulatory Entities include the NTIA and FCC. They are the trusted entities in the system and regulate the operation of SAS

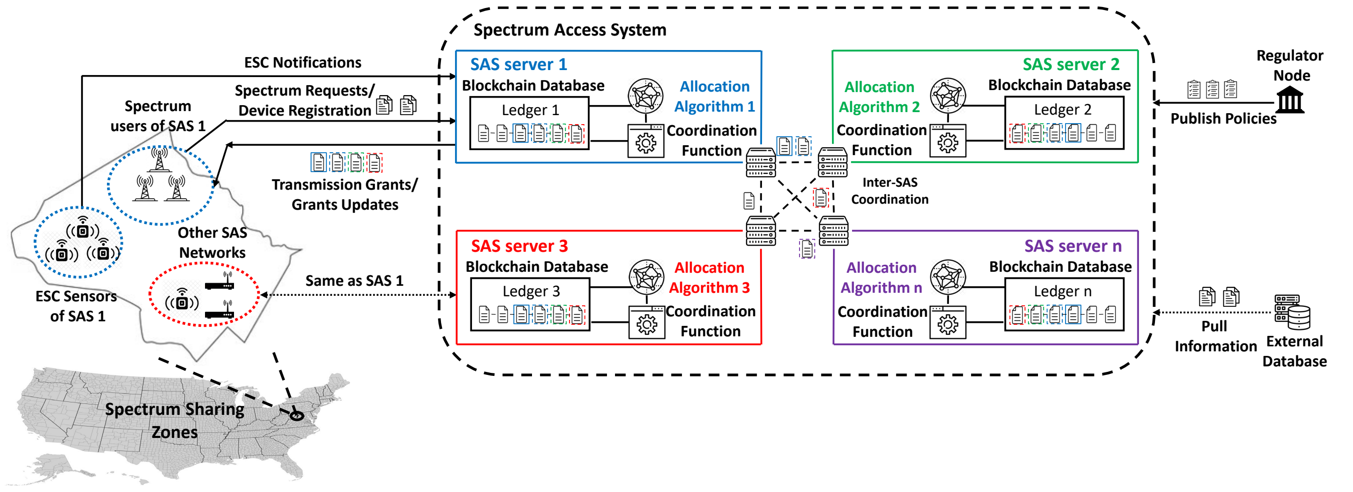


Figure 1: TriSAS system model.

administrators. They do not participate in the spectrum sharing service operations and only publish and enforce regulation rules in the system. The frequency of policy upgrades is relatively low, in the order of days and months.

External Databases provide supplemental information to SAS servers, such as geographic information. The data size of the pulled records from them is usually very large and the frequency of updates is very low. For some of them, the SAS server may only pull the records once to its cloud for the whole life cycle unless there are significant changes. Therefore, we do not consider these records necessary for our real-time coordination procedure.

3.2 Threat Model

The WInnForum standard stipulates the 5G spectrum sharing system to establish a CBRS public key infrastructure (PKI) [30]. As a result, each node in the network, including a spectrum user and a SAS server, can sign its messages with its own private key. The other nodes can verify the signatures with the corresponding public key retrieved from the CBRS PKI. This prevents a network-level attacker from eavesdropping, modifying, and forging messages (or transactions) in the system. Unfortunately, this built-in security mechanism cannot distinguish the messages sent by malicious SAS servers, who are considered insider attackers and can sign the malicious messages with their own secret keys to bypass the cryptographic checks.

Because of crashes down, regular maintenance, and even malicious hacking, we consider a portion of the SAS server may deviate from the normal operation routine for a certain period of time. But we assume they can eventually be fixed. Furthermore, some selfish SAS administrators may purposely sabotage the system by having its SAS servers send conflicting information to other SAS servers under the current framework for their own benefit. For example, the attacker can lie about the status of specific spectrum transmission grants to cause a de-synchronization of the service state between different SAS servers and further trigger allocation conflicts, ultimately jeopardizing shared spectrum use. The attacker when acting as a selfish administrator may configure its SAS server to propose unfair allocations via its proprietary algorithm to give its own clients extra spectrum resources when it is in turn.

In this work, we use "Byzantine behavior" to cover all the possible SAS server behaviors that deviate from the normal routine (i.e., arbitrary behaviors that do not follow the correct coordination procedure). We assume the Byzantine servers are fewer than one-third of the total population, which follows the convention in classical Byzantine fault tolerance problems in distributed systems [5, 6, 21]. Moreover, in current SAS ecosystems, the SAS administrators and their managed SAS servers are certified entities. They make a profit in the spectrum sharing market and are incentivized to follow the FCC rules for the operation of SAS administrators, i.e., 47 CFR Part 96 [24]; failure to comply can have serious legal ramifications. The certified and commercial nature of SAS administrators makes the 2/3 honest majority assumption achievable in practice.

Spectrum users may also behave in certain malicious ways. They can launch DoS attacks or send falsified spectrum requests to acquire excessive allocations. This requires the SAS servers to enforce strict access control. For this work, we regard this as the responsibility of individual SAS administrators and out of the scope of the inter-SAS coordination problem.

3.3 System Goal

The inter-SAS coordination mechanism takes input sets $\mathcal{R}_i(t)(i = 1, 2, \cdots, n)$ from all SAS servers that contain all active requests and can ensure the following properties:

- **State Synchronization:** The inter-SAS coordination mechanism is conducted periodically and after each round the states $\mathcal{S}_i(t)$ of SAS servers shall be synchronized.
- **Byzantine Resilience (Operation Security):** The coordination mechanism needs to counter Byzantine servers. It shall ensure that the honest servers function normally and will not be affected by Byzantine servers when they constitute less than 1/3 of the total population. Once the crashed or malicious servers are recovered, they can synchronize themselves with the honest ones easily.

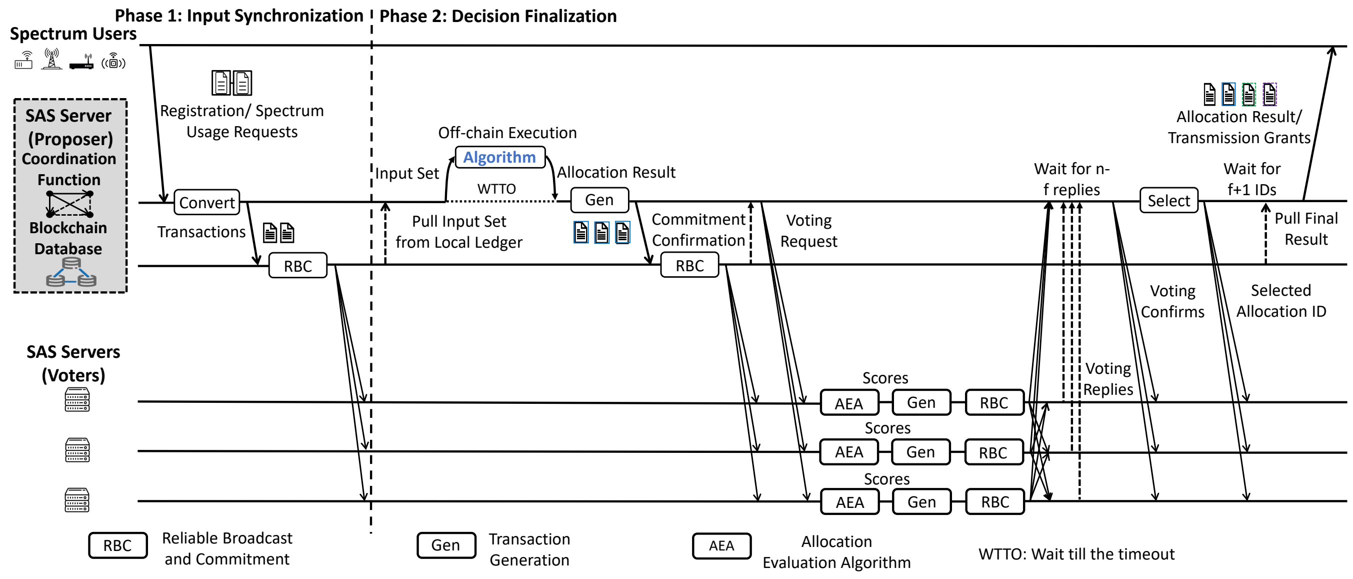


Figure 2: TriSAS workflow.

- **Uniform Allocation and Record Audibility:** The distributed SAS servers should agree on a uniform allocation for each user's spectrum request. The input sets and spectrum allocations (along with how the agreement is reached) need to be stored in a public immutable ledger. Every node can get access to this ledger for audibility purposes. As a result, any malicious behaviors can be identified and tracked in the immutable ledger and the corresponding malicious party cannot deny doing so because all records are cryptographically signed and verifiable.
- **Fairness and Efficiency:** The coordination mechanism needs to be performed in real-time without adding too much overhead and latency. The processing latency shall be smaller than the periodical communication interval between CBSDs and servers (300 seconds) under practical input rates (about 30 requests per minute). The coordination mechanism also needs to ensure that the final allocations are voted by a 2/3 majority of servers to avoid potential unfair allocations. To avoid imposing a large on-chain execution overhead, the proprietary allocation algorithms shall be executed locally.

4 COORDINATION MECHANISM

4.1 Mechanism Overview

TriSAS is a two-phase mechanism including the input synchronization and decision finalization phases, as shown in Figure 2. For each SAS server, the input synchronization phase is triggered whenever there are new device registration requests or spectrum usage requests. The input synchronization phase needs to ensure that all SAS servers share a common input set $\mathcal{R}(t)$, which is the union of all the SAS servers' individual input sets, i.e. $\mathcal{R}(t) = \bigcup_{i=1}^{n} \mathcal{R}_i(t)(i = 1, 2, \cdots, n)$. This set is stored in an immutable ledger in the blockchain database and can be used in the decision-finalization phase to generate spectrum allocations.

The decision finalization phase is in charge of generating spectrum allocations and finalizing one of them as the valid allocation that is favored by a 2/3 majority of SAS servers. This phase requires the coordination function within the servers to periodically (with interval T) check whether there are new transactions in the common input set $\mathcal{R}(t)$. The decision finalization phase is triggered when these proposers find new input transactions in the common input set. Within one interval, there can be one or more proposers, depending on how many SAS administrators are proposing spectrum allocations. The decision finalization phase needs to be finished within the interval T and can accomplish periodical synchronizations of SAS server states $\mathcal{S}_i(t)$.

SAS server's state $\mathcal{S}_i(t)$ contains three types of records, the common input set $\mathcal{R}(t)$, the proposed allocation set $\mathcal{A}(t)$, and the vote set $\mathcal{V}(t)$. These are stored in the immutable ledger of the blockchain database in the form of transactions and data blocks. All participants can get access to the ledger for audit purposes. Each data block contains a hash value of previous blocks and any modification of already committed blocks can be detected.

4.2 Input Synchronization

The SAS servers take the requests from the spectrum users as the inputs to the system. Different SAS administrators have their own input set $\mathcal{R}_i(t)$. The input set consists of requests signed by CBSDs, denoted by $r_{CBSD_k}^{t_{gen}}$, where t_{gen} refers to request generation time. The number of requests increases monotonically with respect to time t, i.e. $|Input_i(t)| \propto t$.

For each server, when the coordination function receives a request from spectrum users, it first transfers the request to the corresponding input transaction by formatting the request into the blockchain database's standard and signing the transaction with its own private key. As a result, the device registration requests are transferred to registration transactions that contain the device identity, antenna characteristics, power level, and location information. The transactions are signed by their origin CBSDs and the enrolled SAS server, denoted by $Reg_{tx} =< id_{device}, antenna, power, loc, sig_{CBSD_k}, sig_{SAS_i} >$.

Similarly, the spectrum usage requests are transferred to request transactions that contain device identity, request spectrum ranges, request time and duration, and are also signed by their origin CBSDs and the enrolled SAS server. We denote them as $Req_{tx} =< id_{device}, range, t_{req}, duration, sig_{CBSD_k}, sig_{SAS_i} >$.

After the transactions are generated, the coordination function sends the transactions to the blockchain database. The coordination function can either send the transactions immediately whenever new ones arrive or can buffer the transactions into batches before sending them. The blockchain database, upon receiving the transactions, organizes the messages into data blocks. The underlying consensus mechanisms of the blockchain database, such as PBFT [6] or Tendermint [5], ensure that these blocks are reliably broadcasted and stored by all honest SAS servers, i.e. the blocks are committed. The coordination function receives a commit confirmation message when all data blocks are committed, indicating that the input synchronization phase is finalized and the requests have been stored in an immutable ledger. Notably, the SAS servers cannot modify or forge the registrations and spectrum usage transactions because they cannot forge the signatures of the CBSDs. Detailed step-by-step algorithm of this phase is summarized in algorithm 1.

Algorithm 1 TriSAS-Phase 1: Input Synchronization

```
Requirement: The number of servers is $n$, and the desired number
    of faults to counter $f$ satisfies $f \leq \lfloor \frac{n}{3} \rfloor$.
Ensure: All SAS administrators have the same input set.
1: procedure Input Synchronization
2:     function Request Conversion and Broadcast
3:         for $i$ in $\{1, 2, \cdots, n\}$, each server $SAS_i$ do
4:             Get $Req_{tx}/Reg_{tx} = Convert(r^{t_{gen}}_{CBSD_k})$.
5:             Broadcast $Req_{tx}$ or $Reg_{tx}$ to others.
6:             Store received $Req_{tx}$ or $Reg_{tx}$ to the ledger.
7:         end for
8:     end function
9: end procedure
```

4.3 Decision Finalization

The decision finalization phase is launched with interval T by SAS servers (proposers) denoted by SAS_j where $j \in \{1, 2, \cdots, p\}$ and p is the number of proposers. This phase is asynchronous with the input synchronization phase. This phase has three sub-phases: allocation generation, allocation voting, and allocation decision.

4.3.1 Allocation Generation. The coordination functions of SAS proposers periodically pull a common input set $\mathcal{R}(t_{pull})$ from the blockchain database to launch the allocation generation process, where t_{pull} refers to the record pulling time. The pulled input set, $\mathcal{R}(t_{pull})$, contains all the active requests before the pulling time t_{pull}. With this common input set, the proposers generate spectrum allocations according to their own proprietary algorithms Alg_j. We define the set of allocations of a SAS proposer as $\mathcal{A}_j(t)$ and the newest allocations at t_{pull} is $\mathcal{A}_j(t_{pull})$. By definition, we have $\mathcal{A}_j(t_{pull}) = Alg_j(\mathcal{R}(t_{pull}))$. $\mathcal{A}_j(t_{pull})$ contains the server's replies to spectrum usage requests, indicating whether they are granted. If a spectrum usage request is not granted, the allocation may provide recommended operation parameters including the power level and spectrum band(s).

There are many allocation algorithms customized for the spectrum allocation problem including simple first-come-first-served (FCFS) allocation, greedy approaches, graph coloring [11], reinforcement learning [1], and optimization-based algorithms [14].

Within each proposer, the allocation algorithm is usually a separate function from the coordination function. This enables the SAS servers to execute complex allocation algorithms offline without the need to consume on-chain computation resources. The coordination function obtains allocation result $\mathcal{A}_j(t_{pull})$ from the algorithm and generates the allocation transactions $Alloc_{tx}$ accordingly. Each allocation transaction is bound with a request transaction, containing the pulling time, its generation time, the id of the request transaction it replies to, the decision (whether the spectrum usage request is granted and if not, some recommended alternative spectrum resources), the id of the allocation result it belongs to, and the signature of its generator. The allocation transaction can be expressed as $Alloc_{tx} =< t_{pull}, t_{gen}, id_{Req_{tx}}, decision, id_{\mathcal{A}_j(t_{pull})}, sig_{SAS_j} >$. The coordination function within each server sends the allocation transactions to the blockchain database in a batch and waits for the commit confirmations from it to conclude this sub-phase.

4.3.2 Allocation Voting. Upon receiving the commit confirmations of the proposed allocation transactions, the coordination function broadcasts voting requests to all SAS servers. The voting request messages do not contain any operation parameters and only serve as notifications. They are signed by the proposer's private key and the recipients can verify their authenticity. The other SAS servers, upon receiving the voting requests, serve as the voters and start the allocation voting sub-phase to generate replies.

Allocation Evaluation Algorithm. The Allocation Evaluation Algorithm (AEA) evaluates the performance of spectrum allocations. It takes the request set $\mathcal{R}(t_{pull})$ and allocations $\mathcal{A}_j(t_{pull})$ as the input and produces scores indicating the performance of different allocations, i.e. $s = \textbf{AEA}(\mathcal{A}_j(t_{pull}), \mathcal{R}(t_{pull}))$.

For each spectrum user, $\mathcal{A}_j(t_{pull})$ contains the feedback about whether its request is granted. There could be spectrum usage conflicts between different spectrum requests, and AEA first checks whether $\mathcal{A}_j(t_{pull})$ resolves these conflicts, i.e. whether $\mathcal{A}_j(t_{pull})$ satisfies the interference and priority requirements. A proper allocation shall ensure that the lower tier users cause no harmful interference to higher tier users. If an allocation cannot pass this check, it will get a zero score. For the allocations that have passed the checking process, AEA evaluates their performance in terms of spectrum utilization rate and fairness. We assume the numbers of spectrum usage requests from SAS servers are $l_1, l_2, \cdots, l_n$. The numbers of granted spectrum usage requests in allocation $\mathcal{A}_j(t_{pull})$ are $g_1, g_2, \cdots, g_n$. Note that $g_i \leq l_i$ because not all the requests are necessarily granted.

We define the spectrum utilization rate as the ratio between granted requests and original requests:

$$u = \frac{\sum_{i=1}^{n} g_i}{\sum_{i=1}^{n} l_i} \tag{1}$$

For fairness measurement, we choose the well-established Jain's fairness index [15]. Considering $\mathbf{H} = (\frac{g_1}{l_1}, \frac{g_2}{l_2}, \cdots, \frac{g_n}{l_n})$ as the request grant rate of all SAS servers. We define the fairness metric as:

$$v = \frac{(\sum_{i=1}^{n} \frac{g_i}{l_i})^2}{n \sum_{i=1}^{n} (\frac{g_i}{l_i})^2} \tag{2}$$

In the ideal situation if the allocation is totally fair, the fairness metric $v = 1$. In the worse case if the allocation is totally unfair, the fairness metric $v = \frac{1}{n}$.

The final evaluation score s can be expressed as a function of u and v, i.e. $s = f(u, v)$. Different application scenarios may apply different $f()$ to satisfy their unique requirements. For example, s can be the multiplication of the utilization rate u and fairness score v, i.e. $s = uv$, indicating the area the allocation covers in the (u, v) space, as well as serving as a metric to evaluate an allocation. Or the users can set a fairness threshold to only accept allocations that obtain a higher fairness score than the threshold th. In this case, $f(u,v) = \begin{cases} u & \text{if } v \geq th \\ 0 & \text{if } v < th \end{cases}$.

The voters generate voting transactions based on their local calculation of scores through the AEA. Each voting transaction contains the id of the allocation it votes for, the score, its generation time, and the signature from its generator. The transaction can be expressed as: $Vote_{tx} =< id_{\mathcal{A}_j(t_{pull})}, s, t_{gen}, sig_{SAS_i} >$. The coordination function within each voter sends the voting transactions to the blockchain database, which are stored in the ledger and reliably broadcasted to others. After the coordination function gets commit confirmations from the blockchain database, it replies to the proposer with a voting reply message. The proposer waits for $n - f$ replies and upon receipt, broadcasts a voting confirmation message to conclude this sub-phase.

Within each server, the allocation generation sub-phase and the allocation voting sub-phase take place concurrently. This means a server can both serve as a proposer to generate new allocation proposals and serve as a voter to vote for others' allocation proposals at the same time. The number of proposers p is a known parameter for every SAS server. If the number p equals n, all SAS servers propose allocations to compete with each other; and if equals one, there is only one proposer and all the others serve as voters to approve the allocations proposed by it.

4.3.3 *Allocation Decision.*

The final allocation decision step waits for the voting confirmation replies from all committed ($\leq p$) allocations, i.e. those that had sent a voting request message before, until a timeout threshold. The SAS servers select the valid allocation with the highest score $\mathcal{A}_{final}(t_{pull})$ as the final selected allocation. In this context, valid means there are at least $f + 1$ identical votes from different servers stored in the ledger. The servers broadcast the id of the final selected allocation $id_{\mathcal{A}_{final}(t_{pull})}$ in the voting decision message and upon receiving $f + 1$ identical results signed by different servers, pull $\mathcal{A}_{final}(t_{pull})$ from the database and reply to the users, which conclude the whole coordination process. Detailed step-by-step algorithm of this phase can be found in algorithm 2.

Algorithm 2 TriSAS-Phase 2: Decision Finalization

Requirement: The number of servers is n, the number of proposers is p, and the desired number of faults to counter f satisfies $f \leq \lfloor \frac{n}{3} \rfloor$.
Ensure: The SAS system is robust against f Byzantine faults and all SAS admins obtain a unified, fair, and verifiable spectrum allocation plan.

```
1:  procedure DECISION FINALIZATION
2:      function ALLOCATION GENERATION
3:          for $j$ in $\{1, 2, \cdots, p\}$, each server $SAS_j$ do
4:              Pull common input set $R(t_{pull})$.
5:              Get $A_j(t_{pull}) = Alg_j(R(t_{pull}))$.
6:              Get $Alloc_{tx} = Convert(A_j(t_{pull}))$.
7:              Broadcast $Alloc_{tx}$ to others.
8:              Store received $Alloc_{tx}$ to the ledger.
9:          end for
10:     end function
11:     function ALLOCATION VOTING—PROPOSERS
12:         for $j$ in $\{1, 2, \cdots, p\}$, each server $SAS_j$ do
13:             Broadcast voting requests to voters.
14:             Wait for $n - f$ voting transactions $Vote_{tx}$.
15:             Broadcast voting confirmations to voters.
16:         end for
17:     end function
18:     function ALLOCATION VOTING—VOTERS
19:         for $i$ in $\{1, 2, \cdots, n\}$, each server $SAS_i$ do
20:             Calculate score $s = \mathbf{AEA}(\mathcal{A}_j(t_{pull}), \mathcal{R}(t_{pull}))$ upon
                receiving voting request.
21:             Broadcast the voting scores to others.
22:         end for
23:     end function
24:     function ALLOCATION DECISION
25:         for $i$ in $\{1, 2, \cdots, n\}$, each server $SAS_i$ do
26:             Wait for $p$ confirmations from all proposers.
27:             Select the best allocation $\mathcal{A}_{final}(t_{pull})$ from $p$ pro-
                posed allocations.
28:             Broadcast the identity $id_{\mathcal{A}_{final}(t_{pull})}$.
29:             Wait for $f + 1$ identical identities.
30:             Pull $\mathcal{A}_{final}(t_{pull})$ from the ledger and replies final
                results to the clients.
31:         end for
32:     end function
33: end procedure
```

In summary, from the spectrum users' perspective, a valid transmission grant corresponds to the following list of transactions: a registration transaction Reg_{tx} containing device and physical operation information, a request transaction Req_{tx} containing the spectrum usage request, an allocation transaction $Alloc_{tx}$ showing the spectrum usage decision, at least $f + 1$ identical vote transactions $Vote_{tx}$ showing the validity of the allocation, and at least $f + 1$ identical voting decision messages showing SAS servers' endorsements. Note that the voting decision messages do not necessarily

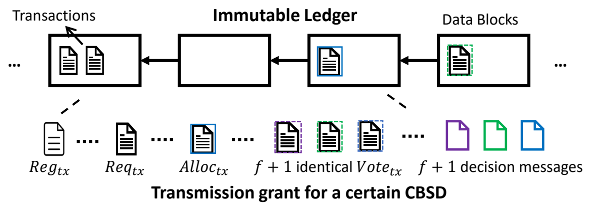


Figure 3: The data structure of a valid transmission grant.

need to be stored in the immutable ledger because the stored allocation and voting transactions are enough for checking the integrity and correctness of the voting process. The honest servers will get correct results even if the last id broadcasting and verification is not there i.e. they directly pull the final results back to clients after the selection step. We keep it in the coordination process to prevent Byzantine servers from forcing their subscribers to transmit in the unauthorized band(s) to jeopardize the operation of others because with this step the clients shall only consider a transmission grant to be valid if and only if there are at least $f + 1$ endorsements from the SAS servers which Byzantine nodes cannot accomplish.

With all these transactions and messages, the users know that the spectrum usage decisions they have received are reliable and agreed upon by the majority of the servers. The users can verify the transactions and decisions in the blockchain ledger. We demonstrate the data structure of a valid transmission grant in Figure 3.

4.4 Analysis

State Synchronization. The coordination function waits for the blockchain database to reply with a commit confirmation each time it sends a transaction. Because the blockchain database itself has an underlying Byzantine fault-tolerant consensus mechanism, which has been well investigated and proved such as PBFT and Tendermint, all the transactions are stored in distributed immutable ledgers across all servers. The state $\mathcal{S}_i(t)$, including the input records $R_i(T)$, allocation records, and vote records are synchronized when the coordination process is accomplished.

Decision Correctness. Each proposed allocation receives $n - f$ replies from the voters (including the proposer himself) before it proceeds. Among them at most f votes are malicious. When $n \geq 3f + 1$, the remaining $n - 2f$ honest votes are identical and take up the majority. Therefore, by majority voting, the Byzantine servers cannot affect voting scores of honest allocations. However, the proposers themselves can be Byzantine and propose allocations. But a Byzantine allocation proposed by a Byzantine server cannot have a higher score s than normal allocations, otherwise, it is a correct allocation. As a result, Byzantine allocations cannot be selected as the final allocation and the decision correctness is ensured. Our mechanism has a timeout threshold, which means different servers need to be synchronized. We consider this assumption to be practical because current telecommunication networks usually have stringent time synchronization requirements and many protocols such as the network time protocol (NTP) [23] and precision time protocol (PTP) [9] are used for this purpose.

Complexity. We denote the number of servers as n and the number of CBSDs as m. The message complexity of our coordination mechanism is $O(n^2)$. The memory consumption is $(n + 2)m + n^2$, or considering $m >> n$ is $O(mn)$. More specifically, within one coordination round the ledger stores m registration transactions Reg_{tx}, m request transactions Req_{tx}, n proposed allocations with each one containing m replies ($alloc_{tx}$) to the requests and n^2 vote transactions $Vote_{tx}$.

Table 2: Round trip times (in milliseconds) among nodes in different areas.

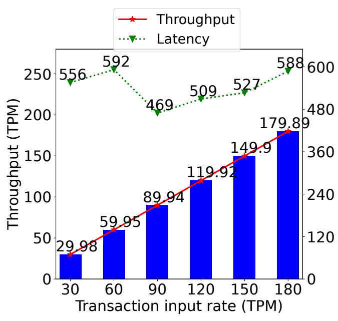
Node	Location	1	2	3	4	5
1	Lab (N.Virginia)	-	-	-	-	-
2	AWS (N.Virginia)	7	-	-	-	-
3	AWS (Ohio)	21.5	11.2	-	-	-
4	AWS (California)	84	60.6	51.4	-	-
5	AWS (Oregon)	72	63.5	48	20.2	-

5 EXPERIMENTAL RESULTS

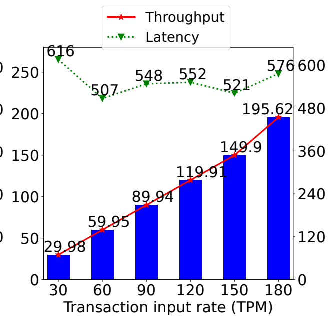
5.1 Experiment Setting

We implemented a prototype of TriSAS on the Amazon AWS cloud computing platform. Our system consisted of {4, 7, 10, 13} EC2 instances serving as SAS servers and one desktop in our lab serving as spectrum users. Each server instance was an EC2 T2.Large node that had two vCPUs, one 8GB memory, and one 32GB disk. They were located in four different areas across the U.S., including northern Virginia, Ohio, Oregon, and northern California. This simulated the real-life deployment of SAS servers because in reality servers of different SAS administrators are deployed in their own data centers across the country. The network latency between the desktop in our lab and cloud servers is shown in Table 2. We chose Bigchandb as the blockchain database. Each server installed a Bigchaindb implementation and we configured them together to form a Bigchaindb network. Bigchaindb is a representative and well-documented blockchain database. We leveraged the Python programming interface of Bigchaindb, i.e. the Python driver of Bigchaindb to implement our coordination mechanism. Bigchaindb provides two types of transaction templates including the *create* transaction template and *transfer* transaction template. We used the *create* template for the registration transaction Reg_{tx} and the *transfer* template for the other types of transactions, i.e. Req_{tx}, $Alloc_{tx}$ and $Vote_{tx}$. The contents of these transactions were included in the *asset* field and *metadata* field of the templates. Bigchaindb uses the Tendermint consensus mechanism as the underlying consensus protocol. Upon initialization, the Tendermint consensus mechanism generates a pair of elliptic curve-based public and private keys for each participant. We used this key pair to identify nodes and sign transactions.

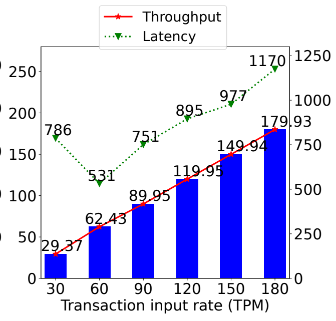
The key evaluation metrics are the system's throughput and latency under a certain volume of input traffic. Within a real-life spectrum sharing zone, which is typically a county, there are about 400 CBSDs and each of them can send a request to the server per heartbeat interval, which by the WInnForum standard is a 300-second interval [29]. Together this contributes to 80 transactions per minute (TPM) input rate. This is the maximum input rate because most of the time the CBSDs just send a heartbeat message without any meaningful content. We can expect the usual traffic volume to



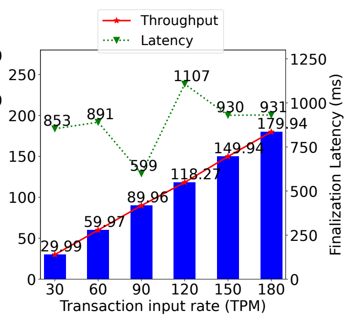
(a) $n = 4$: 4 servers in the network.



(b) $n = 7$: 7 servers in the network.

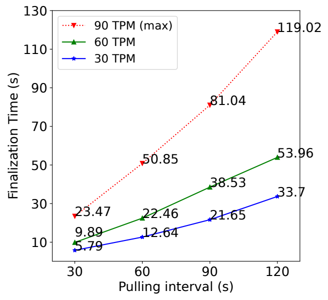


(c) $n = 10$: 10 servers in the network.

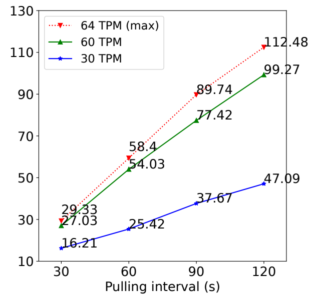


(d) $n = 13$: 13 servers in the network.

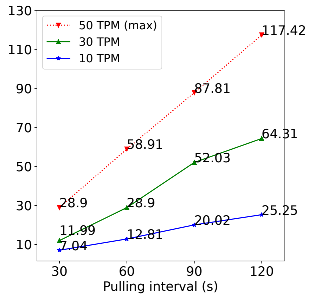
Figure 4: Input synchronization throughput and latency performance. When n={4, 7, 10, 13}, the system can tolerate {1, 2, 3, 4} Byzantine faults.



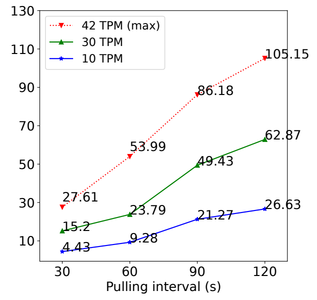
(a) $n = 4$: 4 servers in the network.



(b) $n = 7$: 7 servers in the network.



(c) $n = 10$: 10 servers in the network.



(d) $n = 13$: 13 servers in the network.

Figure 5: Decision finalization latency performance. When n={4, 7, 10, 13}, the system can tolerate {1, 2, 3, 4} Byzantine faults.

be on a scale of 10 TPM. In our experiment, we conducted stress tests for the system and chose the following input rates: {30, 60, 90, 120, 150, 180} TPM. Each request contained its desired spectrum bands and location. We assumed there are 100 locations and 15 spectrum bands (one band is 10 MHz), and each time a request randomly asked for three to five bands and one location.

Latency is another important evaluation metric, especially for the second decision finalization phase. This is because the second phase is conducted periodically and must be finished within the interval T, otherwise, the coordination mechanism fails. The decision finalization interval is expected to be shorter than the heartbeat interval (300s) because if this is guaranteed, the users can get real-time replies in the next heartbeat communication with servers. In our experiment, we chose the interval as {30, 60, 90, 120} seconds under a fixed input rate.

5.2 Input Synchronization Performance

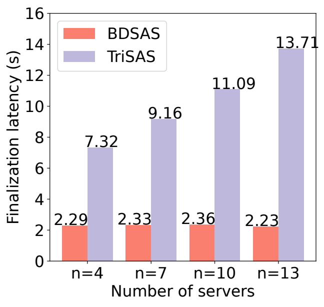
In Figure 4 we demonstrate the experiment results of the input synchronization phase. The figures show the throughput and latency performance when the network size increases from $n = 4$ to $n = 13$, which can tolerate 1 to 4 Byzantine faults. We can observe that their output throughput rates are monotonically increasing and follow a linear relationship with respect to the input rates. This implies that the system throughput has not yet reached its bottleneck and may increase even with larger input rates. We can also observe that, for all settings, the system throughput is almost identical to the input rate, showing that the request messages can be processed in real time without buffering.

The latency of the input synchronization phase, i.e. the time interval between the transactions proposal and commitment, is very stable when the network size $n = 4$ and $n = 7$. It is about 550 milliseconds for every input rate. But when the network becomes larger, the latency increases to 800 milliseconds to 1 second when $n = 10$ and $n = 13$. We consider this to be because larger networks impose more communication overhead for the system, resulting in longer processing delays.

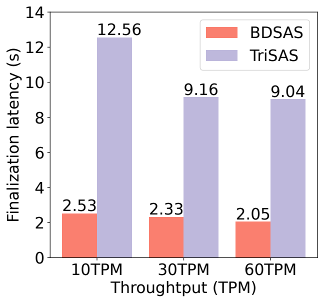
5.3 Decision Finalization Performance

In Figure 5 we demonstrate the performance of the decision finalization phase. In the experiment, we mainly focused on the latency performance of this phase. We changed the size of the network from $n = 4$ to $n = 13$. For each network size, we checked the system's performance with respect to a normal input rate (10 or 30 TPM) to its maximum stable input rate, which was obtained as the maximum input rate that keeps the processing latency smaller than the interval T, meaning that the decision finalization phase can be finished within the interval.

For each case, we can observe that the processing latency is significantly increased when the input rate changes from the normal rate to the maximum rate for a fixed polling interval. This is



(a) Performance verses server numbers.



(b) Performance verses throughput.

Figure 6: Latency performance of TriSAS and BD-SAS under different network sizes and input rates.

because a higher input rate contributes to a larger number of request messages $\mathcal{R}(t)$, which consumes more processing time for the transaction commitments and allocation generations. For a fixed input rate, we can observe that the latency linearly increased with respect to the polling interval. This is because the system gets more requests for larger pulling intervals and needs more time to process them. When the network size becomes larger, we find that the performance decreases as the system has smaller maximum stable input rates and longer processing latency. For example, the maximum stable input rate decreases from 90 TPM to 64 TPM when n increases from 4 to 7; and if we fixed the input rate to 30 TPM, the processing latency when $n = 7$ is significantly larger than that of $n = 4$.

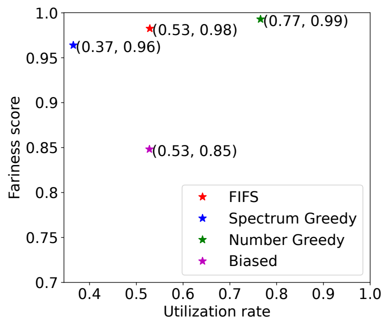
5.4 Benchmark Comparison

We compared the performance of our mechanism with BD-SAS [32], another blockchain-based decentralized SAS following the setting that the system processes the requests sequentially one by one instead of the current default periodical batch-based processing routine to make fair comparisons because BD-SAS does not support the second way. We demonstrate the performance of TriSAS and BD-SAS in figure 6. We focused on the average latency of both mechanisms to process one request.

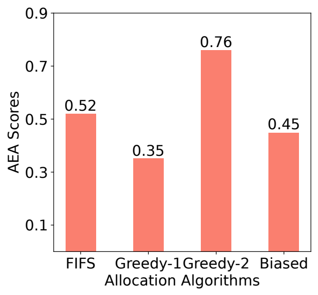
We observe that when the network size increases from $n = 4$ to $n = 13$, BD-SAS keeps a very stable processing latency at about 2.2 seconds, while TriSAS has a larger latency that increases from 7.32 seconds to 13.73 seconds under 30 TPM input rate. For a fixed $n = 7$ network, we find that BD-SAS's processing latency is still stable at about 2.3 seconds while TriSAS's latency varies from 9.04 seconds to 12.56 seconds. From the result, we find that TriSAS imposes a much larger overhead than BD-SAS. We consider this is because BD-SAS adopts a much simpler service model as it relies on smart contracts to fulfill the spectrum assignment and cannot implement complex allocation algorithms, execute them off-chain, or allow algorithm diversity, which is crucial for inter-SAS coordination.

5.5 Fairness Performance

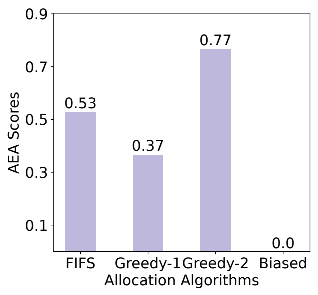
We conducted a simulation to evaluate the performance of several allocation algorithms including first-come-first-serve (FCFS), spectrum greedy, number greedy, and biased allocation algorithms with our AEA algorithm. FCFS means that if two requests ask for the usage of the same spectrum band in one location, the earlier request is approved and the latter one is rejected. Spectrum greedy means that if two requests have conflicts, the one with larger spectrum bands wins. Number greedy means that the algorithm tries to maximize the number of granted requests. The biased allocation algorithm tries to maximize the total number of grants for a certain server. We assumed there were 300 spectrum usage requests toward 100 locations with each one asking for a random amount of spectrum band(s). We repeated the simulation for 100 trials and within each trial, every allocation algorithm generated one allocation to obtain an AEA score. We considered the average performance of the 100 trails as the final performance of each allocation algorithm.



(a) U (utilization)- V (fairness) map.



(b) Performance when $s = uv$.



(c) Performance when $th = 0.9$.

Figure 7: Evaluation performance of different allocation algorithms.

We demonstrate the experiment result in figure 7. We first plot the (u, v) map, i.e. the utilization rate and fairness score map of the four allocation algorithms. For fairness performance, we can observe that the biased allocation algorithm achieves the lowest score, which is consistent with our intuition. For the utilization rate, the number greedy allocation algorithm achieves the best performance. To further provide overall unified and quantitative measurements for different algorithms, we calculate two different AEA scores as $s_1 = uv$ and $s_2 = \begin{cases} u & \text{if } v \geq th \\ 0 & \text{if } v < th \end{cases}$, with the first one indicating the area the algorithms covered in the (u, v) map, and the second one enforcing a fairness threshold. We can observe that the number greedy algorithm achieves the best performance for both two scores. Therefore, if the SAS administrators use the aforementioned allocation algorithms to generate spectrum allocations, the one employing the number greedy algorithm shall win the voting phase with the highest probability. We clarify that in reality, the

SAS administrators may use much more complex allocation algorithms than the four we mentioned. However, they can still use AEA to evaluate their performance and fulfill the decision-finalization phase in our design.

6 DISCUSSION

Scalability. In this work, we mainly focus on the spectrum sharing problem within one zone. However, in reality, the SAS administrators may conduct spectrum scheduling and inter-SAS coordination in many zones across different areas simultaneously. In this case, the number of spectrum users and input volume is significantly increased. The bottleneck of the current implementation is that we use the Amazon EC2 computing instances and these nodes have limited computation and memory resources. Their computation and memory power is only comparable to a daily-used desktop and certainly can not handle very large user numbers and input volume. On the bright side, all the SAS administrators are major corporate entities with access to high-performance computing clusters. Therefore, we can expect them to have enough resources to handle large-scale and complex inter-SAS coordination problems.

Complex allocation algorithm. For the spectrum allocation algorithm, our current implementation only considered linear-complexity allocation algorithms. We observe that there exist more complex allocation algorithms that utilize optimization techniques to achieve high allocation efficiency and fairness [14]. In this regard, TriSAS can incorporate these complex algorithms in a straightforward fashion since it enables different SAS servers to use proprietary algorithms. Nonetheless, how to accomplish the vote-and-selection process within a hard time limit, especially when different SAS servers finish allocation generation at disparate times, is an outstanding issue and we leave this problem to future work.

SAS client privacy. In our scheme, we assume each SAS server disseminates its local client requests to all other servers, allowing all honest SAS servers to receive the same super-set of client requests. In practice, however, different SAS providers may be reluctant to share their client data with each other, at least in its original form, to protect the competitive advantage of their algorithms and the privacy of locally subscribed clients. To address this privacy concern, we identify two potential extensions of TriSAS. First, a SAS server may anonymize or obfuscate sensitive information about client requests, such as device ID and location, while preserving a high level of allocation accuracy. Differential privacy techniques can be used to help each client establish a privacy budget [8]. Second, a SAS server may employ a server-level trusted execution environment (TEE) solution, such as Intel SGX [7] and AMD SEV [16], to compute over confidential client requests (to decrypt inside the TEE enclave with keys secretly provisioned from other SAS servers). A program integrity proof can be provided to other SAS servers through remote attestation. Meanwhile, how to manage the decryption keys among SAS servers would require a secure design.

7 RELATED WORK

State Machine Replication and BFT Consensus. When it comes to realizing a fault-tolerant distributed computing service that achieves uniform decision among participants, state machine replication (SMR) is heralded as the *de facto* paradigm [4, 6, 26], where a consortium of replicated servers provides consistent computation service in response to a sequence of client requests, despite a certain portion of faulty servers. Based on the type of faults they address, the SMR-based consensus protocols can be classified into crash fault-tolerant (CFT) consensus protocols and Byzantine fault-tolerant (BFT) consensus protocols. Paxos [20], Raft [13] and Zab [25] are typical CFT consensus protocols. They can tolerate crashed faults when the majority of the servers behave. Many industrial distributed computing systems such as Apache Hadoop [27] and Apache Kafka [19] use them as the fundamental consensus mechanisms. The BFT consensus protocols address Byzantine failures, which exhibit arbitrary behaviors due to malicious hacks, device crashes, and network failures. Compared to CFT consensus protocols requiring crash faults to comprise less than half of the total population, BFT consensus protocols counter Byzantine faults when their population is less than one-third. PBFT [6], Zyzzyva [17], and Tendermint [5] are well-known BFT consensus protocols.

Blockchain-based SAS. Prior works have explored using blockchain smart contract to implement a spectrum allocation mechanism directly [3, 31] or to aid the current SAS server in query aggregation and allocation publication [12, 34]. It is further explored in TrustSAS [12] and BD-SAS [31] that a two-layer blockchain framework may provide further scalability benefits. However, as we have discussed in Section 2, blockchain-based SASs have certain limitations and cannot fully address the inter-SAS coordination problem.

8 CONCLUSION

In this paper, we investigate the inter-SAS coordination problem in the 5G spectrum sharing system. We identify the drawbacks of the WInnForum CPAS standard as it is unable to provide secure and efficient inter-SAS coordination service. We further analyze the current blockchain-based SAS solutions and identify their limitations of being unable to allow different SAS servers to have their own proprietary allocation algorithms, as well as failing to enable efficient off-chain execution of them. To address this problem, we propose TriSAS, a two-phase coordination mechanism that not only provides security guarantees on inter-SAS coordination but also ensures high throughput and low processing latency in generating spectrum allocations to clients. We implemented a prototype of TriSAS on the Amazon cloud computing platform and conducted extensive experiments to evaluate its performance. The results show that TriSAS can be practically used in real-life systems. This work contributes to the state-of-the-art in inter-SAS coordination and communication, an important problem that is often ignored by the community. This problem may also involve commercial, public policy, and enforcement activities, which together contribute to a healthy spectrum-sharing market.

ACKNOWLEDGMENTS

This work was supported in part by the US National Science Foundation under grants 2331936, 2332675, 2247560, 2247561, 2154929, and 1916902, the US Army Research Office under grant W911NF-20-1-0141, the Virginia Commonwealth Cyber Initiative (CCI), and a gift from InterDigital.

REFERENCES

[1] Waseem Abbass, Riaz Hussain, Jaroslav Frnda, Irfan Latif Khan, Muhammad Awais Javed, and Shahzad A. Malik. 2022. Optimal Resource Allocation for GAA Users in Spectrum Access System Using Q-Learning Algorithm. *IEEE Access* 10 (2022), 60790–60804. https://doi.org/10.1109/ACCESS.2022.3180753

[2] J Chris Anderson, Jan Lehnardt, and Noah Slater. 2010. *CouchDB: the definitive guide: time to relax.* " O'Reilly Media, Inc.".

[3] Thirasara Ariyarathna, Prabodha Harankahadeniya, Saarrah Isthikar, Nethmi Pathirana, HMN Dilum Bandara, and Arjuna Madanayake. 2019. Dynamic Spectrum Access via Smart Contracts on Blockchain. In *2019 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 1–6.

[4] Alysson Bessani, João Sousa, and Eduardo EP Alchieri. 2014. State machine replication for the masses with BFT-SMART. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 355–362.

[5] Ethan Buchman. 2016. *Tendermint: Byzantine fault tolerance in the age of blockchains.* Ph.D. Dissertation. University of Guelph.

[6] Miguel Castro, Barbara Liskov, et al. 1999. Practical byzantine fault tolerance. In *OsDI*, Vol. 99. 173–186.

[7] Victor Costan and Srinivas Devadas. 2016. Intel SGX explained. *Cryptology ePrint Archive* (2016).

[8] Cynthia Dwork. 2008. Differential privacy: A survey of results. In *International conference on theory and applications of models of computation*. Springer, 1–19.

[9] John C Eidson, Mike Fischer, and Joe White. 2002. IEEE-1588™ Standard for a precision clock synchronization protocol for networked measurement and control systems. In *Proceedings of the 34th Annual Precise Time and Time Interval Systems and Applications Meeting*. 243–254.

[10] Wireless Innovation Forum. 2022. Spectrum Sharing Committee Policy and Procedure Coordinated Periodic Activities Policy. Available:https://winnf.memberclicks.net/assets/CBRS/WINNF-SSC-0008.pdf

[11] Weichao Gao and Anirudha Sahoo. 2021. Performance Impact of Coexistence Groups in a GAA-GAA Coexistence Scheme in the CBRS Band. *IEEE Transactions on Cognitive Communications and Networking* 7, 1 (2021), 184–196. https://doi.org/10.1109/TCCN.2020.3003027

[12] Mohamed Grissa, Attila A Yavuz, and Bechir Hamdaoui. 2019. Trustsas: a trustworthy spectrum access system for the 3.5 ghz cbrs band. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 1495–1503.

[13] Dongyan Huang, Xiaoli Ma, and Shengli Zhang. 2019. Performance analysis of the raft consensus algorithm for private blockchains. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 50, 1 (2019), 172–181.

[14] Naru Jai, Shaoran Li, Chengzhang Li, Y Thomas Hou, Wenjing Lou, Jeffrey H Reed, and Sastry Kompella. 2021. Optimal channel allocation in the cbrs band with shipborne radar incumbents. In *2021 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*. IEEE, 80–88.

[15] Raj Jain, Arjan Durresi, and Gojko Babic. 1999. Throughput fairness index: An explanation. In *ATM Forum contribution*, Vol. 99.

[16] David Kaplan, Jeremy Powell, and Tom Woller. 2016. AMD memory encryption. *White paper* (2016).

[17] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. 2007. Zyzzyva: speculative byzantine fault tolerance. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*. 45–58.

[18] Michael Kratsios. 2019. Emerging technologies and their expected impact on non-federal spectrum demand. *Executive Office of the President of the United States* (2019).

[19] Jay Kreps, Neha Narkhede, Jun Rao, et al. 2011. Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB*, Vol. 11. 1–7.

[20] Leslie Lamport. 2001. Paxos made simple. *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)* (2001), 51–58.

[21] Leslie Lamport, Robert Shostak, and Marshall Pease. 2019. The Byzantine generals problem. In *Concurrency: the works of leslie lamport*. 203–226.

[22] Trent McConaghy, Rodolphe Marques, Andreas Müller, Dimitri De Jonghe, Troy McConaghy, Greg McMullen, Ryan Henderson, Sylvain Bellemare, and Alberto Granzotto. 2016. Bigchaindb: a scalable blockchain database. *white paper, BigChainDB* (2016).

[23] David L Mills. 1991. Internet time synchronization: the network time protocol. *IEEE Transactions on communications* 39, 10 (1991), 1482–1493.

[24] The Office of the Federal Register (OFR) and the Government Publishing Office. 2016. OFR: Electronic Code of Federal Regulations, Title 47: Telecommunication, Part 96 - Citizens Broadband Radio Service. https://www.ecfr.gov/cgi-bin/text-idx?node=pt47.5.96

[25] Benjamin Reed and Flavio P Junqueira. 2008. A simple totally ordered broadcast protocol. In *proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware*. 1–6.

[26] Fred B Schneider. 1990. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)* 22, 4 (1990), 299–319.

[27] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. 2010. The hadoop distributed file system. In *2010 IEEE 26th symposium on mass storage systems and technologies (MSST)*. Ieee, 1–10.

[28] Martin BH Weiss, Kevin Werbach, Douglas C Sicker, and Carlos E Caicedo Bastidas. 2019. On the application of blockchains to spectrum management. *IEEE Transactions on Cognitive Communications and Networking* 5, 2 (2019), 193–205.

[29] Wireless Innovation Forum 2017. *Requirements for Commercial Operation in the U.S. 3550-3700 MHz Citizens Broadband Radio Service Band Working Document.* Wireless Innovation Forum. https://winnf.memberclicks.net/assets/CBRS/WINNF-TS-0112.pdf Version V2.0.0.

[30] Wireless Innovation Forum 2020. *CBRS Communications Security Technical Specification.* Wireless Innovation Forum. https://www.wirelessinnovation.org/assets/work_products/Specifications/winnf-15-s-0065-v1.0.0%20cbrs%20communications%20security%20technical%20specification.pdf Version V1.2.0.

[31] Yang Xiao, Shanghao Shi, Wenjing Lou, Chonggang Wang, Xu Li, Ning Zhang, Y Thomas Hou, and Jeffrey H Reed. 2022. Decentralized spectrum access system: Vision, challenges, and a blockchain solution. *IEEE Wireless Communications* 29, 1 (2022), 220–228.

[32] Yang Xiao, Shanghao Shi, Wenjing Lou, Chonggang Wang, Xu Li, Ning Zhang, Y. Thomas Hou, and Jeffrey H. Reed. 2023. BD-SAS: Enabling Dynamic Spectrum Sharing in Low-trust Environment. *IEEE Transactions on Cognitive Communications and Networking* (2023), 1–1. https://doi.org/10.1109/TCCN.2023.3270440

[33] Xuhang Ying, Milind M. Buddhikot, and Sumit Roy. 2017. Coexistence-aware dynamic channel allocation for 3.5 GHz shared spectrum systems. In *2017 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*. 1–2. https://doi.org/10.1109/DySPAN.2017.7920771

[34] Hanwen Zhang, Supeng Leng, and Haoye Chai. 2020. A Blockchain Enhanced Dynamic Spectrum Sharing Model Based on Proof-of-Strategy. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*. IEEE, 1–6.