# ConTrib: Universal and Decentralized Accounting in Shared-Resource Systems

Martijn de Vos and Johan Pouwelse

Delft University of Technology

**Abstract**

Preventing the abuse of resources is a crucial requirement in shared-resource systems. This concern can be addressed through a centralized gatekeeper, yet it enables manipulation by the gatekeeper itself. We present ConTrib, a decentralized mechanism for tracking resource usage across different shared-resource systems. In ConTrib, participants maintain a *personal ledger* with tamper-proof *records*. A record describes a resource consumption or contribution and links to other records. Fraud, maintaining multiple copies of a personal ledger, is detected by users themselves through the continuous exchange of records and by validating their consistency against known ones. We implement ConTrib and run experiments. Our evaluation with up to 1'000 instances reveals that fraud can be detected within 22 seconds and with moderate bandwidth usage. To demonstrate the applicability of our work, we deploy ConTrib in a Tor-like overlay and show how resource abuse by free-riders is effectively deterred. This longitudinal, large-scale trial has resulted in over 137 million records, created by more than 86'000 volunteers.

## 1 Introduction

Preventing the abuse of user-volunteered resources is a key requirement in shared-resource systems [10]. Often, such systems are safeguarded from abuse by having a single operator controlling every aspect of the system, essentially acting as a *gatekeeper*. Companies like Uber and AirBnb are prime examples of gatekeepers to markets where user resources (cars and houses) are offered and consumed on a global scale.

Even though the practice of acting as a central gatekeeper is widely adopted, it is a concerning development. Recently, "big tech" companies have obtained significant dominance in the market for digital services [11]. Companies like Google, Amazon, Facebook and Apple are omnipresent in our current society and even have the means of acting as small states, inhabited by billions of users worldwide [1]. Unfortunately, this tremendous concentration of power enables the abuse of resources by the operator itself, potentially affecting a large portion

of users. For example, it has been demonstrated that Uber actively manipulates their ride-hailing market for commercial interests, undermining platform fairness [5].

Decentralized solutions are increasingly considered as an alternative to shared-resource systems with centralized gatekeeping. In contrast to centralized architectures, decentralized networks are fully maintained by participating members with near-equal authority. Compared to centralized systems, decentralized architectures tend to be more resilient against large-scale abuse of its resources [24]. However, peers in a sustainable decentralized system are required to coordinate the exchange of resources themselves. Therefore, preventing abuse of shared resources is a non-trivial and largely unsolved challenge in decentralized networks.

Adopting an *accounting mechanism* to record all resource consumption and contributions by individuals is a viable approach to address abuse in decentralized shared-resource systems [18]. Several accounting solutions have been introduced; however, many of these systems make assumptions on the application domain and are not universal enough to be reused across different shared-resource systems [19, 13, 20]. Blockchain technology also provides accounting capabilities and empowers users with a distributed ledger to securely store their interactions [25]. However, we consider blockchain unsuitable for accounting purposes since bulk storage on the distributed ledger is prohibitively expensive.

In this work, we introduce ConTrib, a universal and decentralized accounting mechanism to prevent abuse in shared-resource systems. Shared-resource systems can leverage ConTrib to securely record resource contributions and consumptions, or *interactions*, within tamper-proof *records*. Records are linked together in a personal ledger and can point to other records by including their hashes. Users continuously exchange random records with others and verify the consistency of hash pointers included in incoming records. This simple, yet effective approach enables quick detection of fraud, the situation where an adversary operates multiple versions of its personal ledger in secret. We implement ConTrib and evaluate the effectiveness and bandwidth overhead of fraud detection. Real-world experiments with up to 1'000 instances reveal that fraud can be detected with 22 seconds under a conservative strategy for record exchange.

To show the practicality and matureness of ConTrib, we address the abuse (overuse) of anonymous bandwidth by free-riders in Tribler. Tribler is our decentralized file-sharing software, downloaded by more than 1 million users [23]. We leverage ConTrib to account bandwidth exchanges in Tribler and refuse services to free-riders during periods of congestion. Our 36-months deployment trial has resulted in over 137 million records, created by over 86'000 volunteers.

The main contribution of this work is three-fold:

1. The universal ConTrib mechanism, enabling accountability in shared-resource systems (Section 2).

2. An implementation and evaluation of ConTrib, revealing fraud detection times within 22 seconds (Section 3).

3. A large-scale trial of ConTrib with 86'000 volunteers, addressing anonymous bandwidth abuse (Section 4).

# 2 ConTrib Design

ConTrib is an accounting mechanism for decentralized systems in which users have to manage access to shared resources, e.g., CPU power, bandwidth or files. A key challenge in accountable shared-resource systems is that users have a natural incentive to misrepresent their prior interactions to hide abuse, e.g., by strategically withholding or modifying information [18]. ConTrib must ensure the correctness of accounted interactions, and address fraud targeted at the data structure. We define fraud in the context of this work as *the manipulation or hiding of accounted interactions*. We consider the accounting of untruthful interactions, that is, the recording of interactions that have not actually occurred in the system, outside the scope of this work.

In this section, we present the design of ConTrib. The ConTrib design is inspired by the tamper-proof properties of the blockchain data structure, but avoids the need for network-wide consensus, minimizing computational overhead and bandwidth usage [26]. We first outline our system and threat model. We then show how interactions are recorded in ConTrib. Finally, we elaborate on the detection of fraud.

## 2.1 System and Threat Model

ConTrib builds upon a peer-to-peer network overlay. We assume that each peer knows the network address of a subset of all peers. The communication channels between peers can be unreliable and unordered, i.e., the arrival time on network messages is not upper-bounded. Each peer owns a cryptographic keypair with a public and private key. Their public key uniquely identifies the peer in the network, and the private key is used to sign records and outgoing network messages. We consider threats in the network layer, e.g., the Sybil and Eclipse Attack, beyond the scope of this work.

Our threat model involves malicious users that manipulate the records in their personal ledger. This manipulation manifests by users operating multiple copies of their personal ledger, possibly by sending different copies to distinct users or withholding records. We assume that the computational capabilities of adversaries are bounded and that cryptographic primitives (e.g., hashing) are secure.

## 2.2 Recording Interactions

We now outline how an interaction between users $a$ and $b$ is recorded. In ConTrib, this interaction is accounted using two distinct records: one *proposal* created by $a$ and one *confirmation* created by $b$. This process works as follows. First, user $a$ creates a proposal, say $P$. $P$ contains the public key of $a$ and $b$, a

type, a payload, and a digital signature over the record content by $a$. The type field is a short string identifier that identifies the application in which the interaction has occurred. The payload is an arbitrary blob of data that describes the interaction. This field is provided by the applications using ConTrib. After $a$ includes all fields in the proposal, the record is persisted to $a$'s database, sent to interaction partner $b$, and disseminated to $f$ random users in the ConTrib network. We refer to $f$ as the *fanout* value.

When user $b$ receives $P$ from $a$, $b$ verifies its validity (this process is discussed in Section 2.3). If $P$ is deemed valid, $b$ determines if the payload in $P$ truthfully describes the interaction. If not, $b$ ignores the incoming proposal. Otherwise, $b$ creates a confirmation $C$ that confirms $P$. This confirmation contains the same fields as $P$, and also includes the hash of $P$. We call this hash in $C$ the *confirmation pointer*. After the creation of $C$, $b$ persists $C$ to its database, sends it to $a$, and disseminates both $C$ and $P$ to $f$ random users. Upon reception of $C$ by $a$, $a$ validates $C$. Now, both parties own the digitally signed proposal and confirmation that together account a bilateral interaction. The process of recording interactions is lightweight since it requires minimal computational power and data exchange. We also note that users can engage in the recording of multiple interactions simultaneously.

To ensure that the illegitimate modification of records after their creation can be detected, we organize all records of the same user in a tamper-evident *personal ledger*. To index the records in a personal ledger, each record includes a sequence number $s \in \mathbb{Z}$ that starts from 1 and is incremented for subsequent records. Each record now also includes the hash of the prior record in the personal ledger, and $n$ additional hashes of $n$ prior, random records. We refer to the hashes that point to prior records in the same personal ledger as *prior pointers*. The inclusion of multiple prior pointers speeds up the detection of illegitimate modifications. The required prior records pointed to by some record $R$ are deterministically given by a pseudo-random function $\sigma$ that takes the public key of the record creator and the sequence number of $R$ as input. $\sigma$

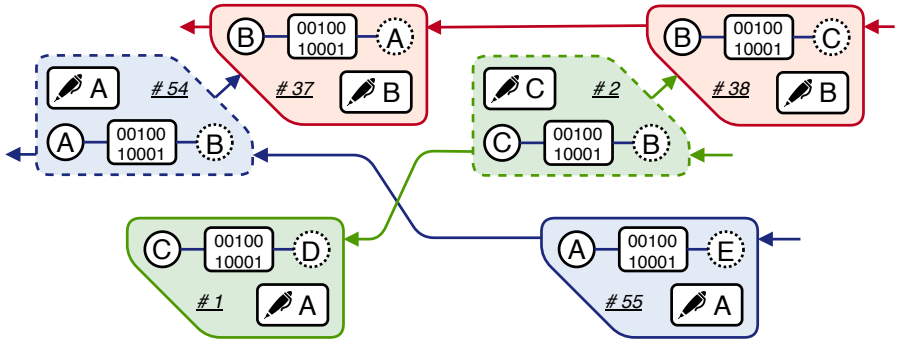
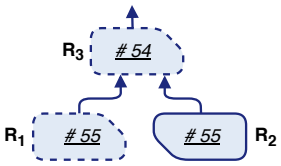
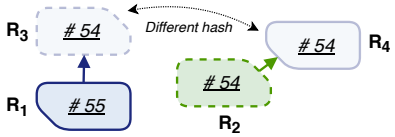


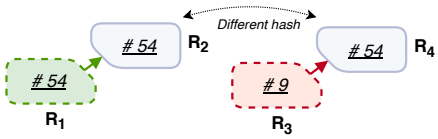

Figure 1: A part of the ConTrib data structure with five users and six records: four proposals and two confirmations (indicated by dashed borders).




(a) *Scenario I*: Records $R_1$ and $R_2$ have the same creator and sequence number. Together, they provide an irrefutable proof of fraud.




(b) *Scenario II*: $R_1$ and $R_2$ point to a record with the same sequence number and creator, but a different hash. This reveals an inconsistency.




(c) *Scenario III*: $R_1$ and $R_3$ confirm a record with the same sequence number and creator, but a different hash. This reveals an inconsistency.

Figure 2: Different scenarios which allows a user to either expose a fraudster, or to detect an inconsistency (without assigning the blame). Same-coloured records belong to the a single creator (blue for $a$, green for $b$ and red for $c$). Solid and dashed records indicate proposals, respectively confirmations. Opaque records are not in possession by the user.

returns a set with at most $n$ sequence numbers of prior records that should be pointed to. All ConTrib instances must use the same implementation of $\sigma$, which we achieve by bundling the implementation of $\sigma$ in the ConTrib software. The modification of a record in the personal ledger now changes the hash of the record and therefore invalidates hash pointers in subsequent records.

Accounting interactions within proposals and confirmations records yields the graph structure in Figure 1. Figure 1 shows a part of the ConTrib graph with six records, created by three distinct users. We show required fields (e.g., the signature and payload) in each record. Same-coloured records are part of a single personal ledger, and confirmations have a dashed border. For presentation clarity, we only show the hash pointer to the prior record in one's personal ledger.

## 2.3 Detecting Fraud

Fraud in ConTrib proceeds when an adversary forks its personal ledger and operates multiple personal ledgers. This would result in pairs of records with the same sequence number and creator, but with a different hash. A key objective of ConTrib is to detect such conflicting records. We remark that ConTrib is built

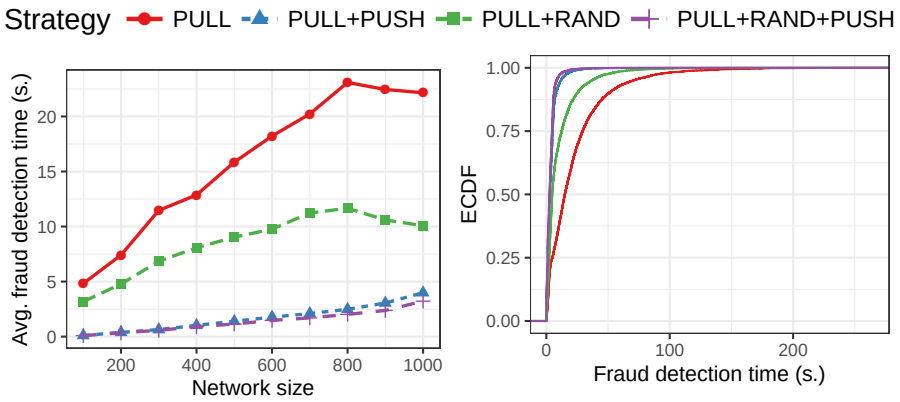
around fraud *detection* instead of prevention. We argue this is reasonable for two reasons. First, shared-resource systems can usually tolerate low amounts of fraud for short periods [17]. Second, the prevention of fraud is usually a resource-intensive process that requires users to reach a network-wide consensus on all created records, e.g., using classical BFT algorithms or Proof-of-Work [26]. We assume that the punishment of detected fraud attempts is realised by the applications using ConTrib, i.e., by not contributing resources to the fraudster for some time.

Fraud in ConTrib is detected by sharing records with random users, and by verifying the consistency of included hash pointers in incoming records against known ones. Users exchange records with others in two ways. First, as explained in Section 2.2, records are sent to $f$ random users upon creation (*push-based exchange*). Second, users in ConTrib continuously requesting random records in the personal ledger of other users (*pull-based exchange*). The collective effort of users enables quick fraud detection in ConTrib.

Fraud is detected by the validation procedure of incoming records. This procedure first verifies the validity of the record itself, e.g., by verifying the digital signature and structure. A critical step during validation is the verification of the (hash) pointers included in a record. Each user keeps track of all encountered pointers in a dictionary, which is queried and updated when encountering new pointers. An inconsistency between hashes, introduced by the modification and sharing of a record in a personal ledger, can now be detected by querying stored hashes in the dictionary.

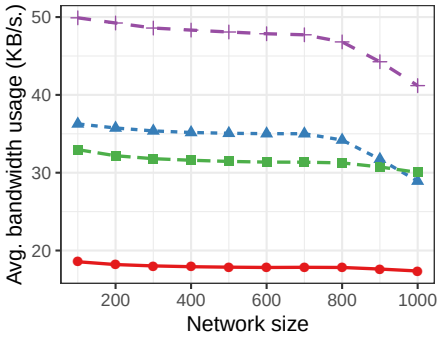
We further elaborate on fraud detection through the validation of hash pointers. Figure 2 highlights three scenarios in which a user can either expose an adversarial user (scenario I) or detect an inconsistency without assigning the blame yet (scenario II and III). Each scenario shows a subset of records stored in or missing from the database of a user (faded records are missing). In Figure 2, the record colour identities its creator. Scenario I (Figure 2a) shows a part of the personal ledger of user $a$. User $a$ forked its personal ledger, and thus committed fraud, since records $R_1$ and $R_2$ have the same creator and sequence number. When another user, say $b$, receives $R_1$ while already having $R_2$, or vice versa, the record pair $(R_1, R_2)$ is sufficient evidence to prove that $a$ has deliberately committed fraud and forked its personal ledger. We refer to this record pair as a *fraud proof*.

Figure 2b shows the scenario where a user receives proposal $R_1$ and already has confirmation $R_2$, or receives confirmation $R_2$ while already having proposal $R_1$. The user does not have $R_3$ and $R_4$. The included prior pointer in $R_1$ differs from the confirmation pointer in $R_2$. This indicates an inconsistency that is either introduced by user $a$ forking its personal ledger (at height 54) or by $b$ having included an invalid hash pointer in $R_2$. A user that encounters this scenario during record validation sends the record pair $R_1$ and $R_2$ to $f$ other random users in the hope that one of them possesses $R_3$ or $R_4$. Figure 2c highlights another scenario where a user encounters two confirmations, $R_1$ and $R_3$, created by different users, that point to a record with the same public key and sequence number, but a differing hash. Again, this indicates a fork of the




(a) Fraud detection times

(b) Fraud detection times for 1'000 instances




(c) Bandwidth usage

Figure 3: The fraud detection times and bandwidth usage of ConTrib, for different record exchange strategies, and while scaling the record creation rate with the network size. We fix the fanout ($f$) to 10.

personal ledger of $a$, or it can be the result of an invalid pointer in one of the confirmations. The validating user now shares the pair ($R_1$, $R_3$) with others.

# 3 Implementation and Evaluation

We implement ConTrib in the Python 3 programming language. We leverage our existing networking library to build a peer-to-peer overlay. We adopt an event-driven programming model using the `asyncio` library and use the UDP protocol for data exchange between peers. Each record contains at most ten pointers to prior records in the same personal ledger. The full implementation

of ConTrib, including tests and documentation, is published on GitHub.[1]

**Experiment Setup.** We evaluate ConTrib on our nation-wide university cluster which hardware specifications can be found online [2]. Precisely, we assess the efficiency of detecting forks in ConTrib by measuring the time between committing fraud and its initial detection. During our experiments, every user records an (artificial) interaction with another random user. For network sizes from 100 to 1'000 users, each user forks its personal ledger with a probability of 10% by removing the last record in its personal ledger and re-using its sequence number for the next record. Each user commits this fraud at most once. All users start with an empty personal ledger and database. We run each experiment ten times and average all results.

We explore the effect of combinations of three different record exchange strategies on fraud detection time and bandwidth usage. With the `PULL` strategy, each user requests five contiguous records at a random index in the personal ledger of another (random) user every half a second. The `PUSH` strategy pushes new records to $f$ random users upon creation. We fix the fanout $f$ to ten during all experiments. With the `RAND` strategy, users also return five random records sampled from their database upon a record request. Users forking their personal ledger refrain from sending the newly created record to $f$ random users when the `PUSH` strategy is active, to avoid detection.

Figure 3 shows the fraud detection times and bandwidth usage, for different record exchange strategies and network sizes ($n$). Figure 3a shows the average fraud detection times in second for increasing network sizes. The `PULL+PUSH` and `PULL+RAND+PUSH` strategies show fraud detection times under five seconds, even for $n = 1'000$. We explain this effect as follows: pushing created records to random users leads to a high probability of at least one user receiving conflicting records. The `PUSH` strategy is thus an effective strategy for fraud detection. The average detection times increase roughly linearly when increasing the network size for the `PULL+PUSH` and `PULL+RAND+PUSH` strategies. Fraud detection times increase when only using a `PULL` strategy. However, under this strategy and for $n = 1'000$, it still only takes 22 seconds on average to detect fraud.

Figure 3b shows an ECDF of the detection times for different strategy and $n = 1'000$. We observe that some fraud instances are only detected after a few minutes. We attribute this to the randomness involved in the fraud detection process. For the `PULL` strategy, 50% of all fraud attempts are detected within 20 seconds. This median decreases to just 2.5 seconds for the `PULL+RAND+PUSH` strategy.

Figure 3c shows the average bandwidth usage while varying the network size. Overall, bandwidth usage remains roughly constant when increasing the network size. This is because the record creation rate scales with the network size, keeping the average bandwidth usage constant. The overhead of the `PUSH` strategy decreases for higher values of $n$ since $f$ is fixed. With $n = 1'000$, the `PULL+RAND+PUSH` strategy requires 41.2 KB/s on average whereas the `PULL` strategy only requires 17.4 KB/s. When deploying ConTrib in a bandwidth-

[1] See `https://github.com/Tribler/py-ipv8/tree/master/ipv8/attestation/trustchain`

constrained environment (eg., IoT), one can reduce the fanout or increase the interval at which records are requested, at the cost of increased fraud detection times.

# 4 Addressing Resource Abuse at Scale

We now present a large-scale deployment trial of ConTrib to address free-riding behaviour in our academic file-sharing system named Tribler. Tribler is downloaded by over 1.5 million users and features a Tor-like overlay that anonymously onion-routes BitTorrent traffic. A downloader in Tribler uses one-hop circuits to download content. Currently, this overlay suffers from an undersupply of exit nodes which are gateways that fetch (unencrypted) content from BitTorrent swarms and forward data to downloaders. This undersupply results in frequent network congestions and overall degradation of download speeds for all users. We leverage ConTrib to account all bandwidth contributions as an exit node, and consumptions as a downloader. We then give preferential treatment to users that have substantially contributed to the network by running an exit node.

**Bandwidth Accounting.** With ConTrib, each peer can earn *bandwidth tokens* by operating an exit node. Downloaders remunerate exit nodes for their services by transferring bandwidth tokens. Each interaction recorded with ConTrib contains the token amount transferred, and the current token balance of the involved users. We plan on addressing linkability concerns by having each node aggregate and delay payouts, a technique introduced in the work of Palmieri et al. [22]. Still, bandwidth accounting with ConTrib does not leak the identity of a downloader to others, nor reveals any data being exchanged between users.

We grant preferential treatment to downloaders with higher token balances during congestion at an exit node. To this end, we modify exit nodes such that each circuit consumes an available slot at their side. We distinguish between *random* and *competitive* slots. When a request for a circuit arrives at an exit node, Tribler first determines if there is a random slot available and if so, assigns the new circuit to it. If no random slot is available, Tribler queries the bandwidth token balance of the circuit initiator $i$ by requesting the latest record in its personal ledger. Upon receiving this balance, Tribler checks eligibility for a competitive slot. If there is an unoccupied competitive slot, it assigns the new circuit to it. If all competitive slots are filled, the circuit of the initiator with the lowest amount of bandwidth tokens, say $p$, is destroyed if the token balance of $i$ is higher than the token balance of $p$. This pre-emptive approach frees up the competitive slot for the circuit of $i$. As a result, peers with a higher token balance have more chance to claim a competitive slot in periods of congestion, compared to free-riders, and thus they experience higher and more stable download speeds.

**Refusing Services to Free-Riders.** We implement the bandwidth accounting logic and slot mechanism in Tribler, and release a new version of our software. We also deploy a dedicated crawler that builds a dataset by fetching ConTrib records from random users in the network. This crawler selects a random user every two seconds, and requests missing records. This has resulted

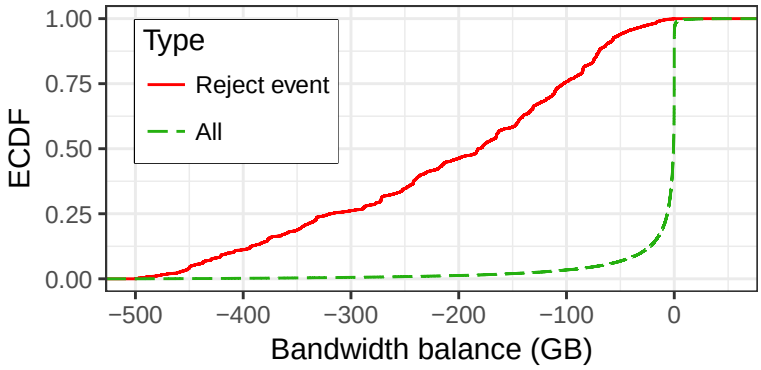



Figure 4: ECDF showing the distribution of bandwidth token balances users and individual rejects events at exit nodes.

in more than 137 million records, created by over 86'000 individuals during 36 months. In addition, our crawler found 127'135 instances of fraud in ConTrib.

To evaluate the effectiveness of ConTrib accounting, we deploy 48 additional exit nodes in the Tribler network. Each exit node has a total of 10 random slots and 20 competitive slots, resulting in a total of 1'440 slots. We log the bandwidth token balance when a circuit initiator is unable to claim a slot at one of our exit nodes. In total, we observe over 1.2 million reject events during a three-week period.

Figure 4 shows an ECDF with the bandwidth token balances of all users (dotted green line) and the balances of reject events (solid red line). We filter out all users and reject events with balances higher than 50GB or lower than -500GB. The median token balance of all users is -713MB and that of reject events -181.4GB, demonstrating that our mechanism targets users with low balances. This deployment trial shows that ConTrib is effective at detecting and addressing free-riding behaviour in Tribler.

# 5 Related Work

There have been various proposals to enhance decentralized networks with accounting capabilities to deter faulty nodes and prevent resource abuse. PeerReview and FullReview are accountability mechanisms that record message exchange between peers, and use dedicated witness sets to detect whether a peer deviates from the protocol [14, 8]. In contrast to our work, these solutions are designed for the low-level logging of all messages exchanged in the network. LiFTinG and AcTinG are protocols for the tracking of free-riding behaviour in gossip-based systems, but they cannot easily be reused in a different context [13, 19]. Similarly, Osipkov et al., devise a distributed system for the accounting of storage activities in file storage networks [20].

Otte et al. present TrustChain, a Sybil-resistant reputation algorithm and distributed ledger [21]. Their data structure resembles ConTrib, however, we identity that peers in TrustChain cannot engage in the recording of multiple interactions simultaneously, limiting throughput and applicability. Crosby et al. present a tree-based data structure for tamper-evident logging [6]. Their data structure is designed around the logging of unilateral events, whereas ConTrib is optimized to account bilateral interactions.

In line with our deployment trial (Section 4), there has been considerable effort to incentivize relay and exit node operators in the Tor network. One of the earlier approaches is Gold Star where directory servers keep track of users providing good services to the community [9]. Other approaches like BRAIDS and LIRA, reward relay and exit nodes with credits that can be redeemed for prioritized traffic [15, 16]. These solutions assume a centralized bank or a group of semi-trusted nodes for credit management. TorCoin proposes a mechanism where relay and exit nodes "mine" a Bitcoin-derived cryptocurrency. TorCoin, however, requires centralized circuit management. [12]. Finally, some shared-resource systems leverage blockchain technology and use monetary incentives to provide communal services, e.g., storage (Filecoin [3]) and BitTorrent bandwidth (BitTorrent token [4]).

# 6 Conclusion and Future Directions

We have presented ConTrib, a universal accounting mechanism to address abuse in shared-resource systems. The ConTrib data structure uses records and hash pointers to capture bilateral interactions. Each user maintains a personal ledger with tamper-evident records. Forking of a personal ledger is detected by the exchange and validation of records. We have implemented ConTrib and have demonstrated with experiments that our mechanism detects forking within seconds. A large-scale deployment trial, involving over 86'000 users, demonstrated how ConTrib addresses free-riding in our peer-to-peer software.

We envision and encourage the deployment of ConTrib for use-cases beyond shared-resource applications. We are currently exploring the accounting capabilities of ConTrib in the context of decentralized trading, order matchmaking and self-sovereign identity [7].

# References


[1] Federico Ast. The new federalism: blockchain will decentralise big tech's power on the internet. *LSE Business Review*, 2018.

[2] Henri Bal et al. A medium-scale distributed system for computer science research: Infrastructure for the long term. *Computer*, 49(5):54–63, 2016.

[3] J Benet and N Greco. Filecoin: A decentralized storage network. Technical report, 2018.

[4] Inc. BitTorrent. Bittorrent token (btt): Tokenizing decentralized file sharing.

[5] Ryan Calo and Alex Rosenblat. The taking economy: Uber, information, and power. *Colum. L. Rev.*, 117:1623, 2017.

[6] Scott Crosby et al. Efficient data structures for tamper-evident logging. In *USENIX Security*, pages 317–334, 2009.

[7] Martijn de Vos et al. Xchange: A blockchain-based mechanism for generic asset trading in resource-constrained environments. *arXiv preprint arXiv:2004.05046*, 2020.

[8] Amadou Diarra and othersn. Fullreview: Practical accountability in presence of selfish nodes. In *SRDS*, pages 271–280. IEEE, 2014.

[9] Roger Dingledine et al. Building incentives into tor. In *FC*, pages 238–256. Springer, 2010.

[10] Brett M Frischmann. Two enduring lessons from elinor ostrom. *Journal of institutional economics*, 2013.

[11] Jon Frost et al. Bigtech and the changing structure of financial intermediation. *Economic Policy*, 2019.

[12] Mainak Ghosh et al. A torpath to torcoin: Proof-of-bandwidth altcoins for compensating relays. Technical report, Naval Research, 2014.

[13] Rachid Guerraoui et al. Lifting: lightweight freerider-tracking in gossip. In *Middleware*, pages 313–333. Springer, 2010.

[14] Andreas Haeberlen et al. Peerreview: Practical accountability for distributed systems. *SIGOPS*, 41(6):175–188, 2007.

[15] Rob Jansen et al. Recruiting new tor relays with braids. In *CCS*, pages 319–328, 2010.

[16] Rob Jansen et al. Lira: Lightweight incentivized routing for anonymity. Technical report, Naval Research, 2013.

[17] Ramayya Krishnan et al. The virtual commons: Why free-riding can be tolerated in file sharing networks. *ICIS*, page 82, 2002.

[18] Michel Meulpolder et al. Bartercast: A practical approach to prevent lazy freeriding in p2p networks. In *IPDPS*, pages 1–8. IEEE, 2009.

[19] Sonia Ben Mokhtar et al. Acting: Accurate freerider tracking in gossip. In *SRDS*, pages 291–300. IEEE, 2014.

[20] Ivan Osipkov et al. Robust accounting in decentralized p2p storage systems. In *ICDCS*, pages 14–14. IEEE, 2006.

[21] Pim Otte et al. Trustchain: A sybil-resistant scalable blockchain. *Future Generation Computer Systems*, 107:770–780, 2020.

[22] Paolo Palmieri and Johan Pouwelse. Paying the guard: an entry-guard-based payment system for tor. In *FC*, pages 437–444. Springer, 2015.

[23] Johan Pouwelse et al. Tribler: a social-based peer-to-peer system. *Concurrency and computation: Practice and experience*, 2008.

[24] Wesley W Terpstra et al. Bubblestorm: resilient, probabilistic, and exhaustive peer-to-peer search. In *SIGCOMM*, pages 49–60, 2007.

[25] Sarah Underwood. Blockchain beyond bitcoin, 2016.

[26] Marko Vukolić. The quest for scalable blockchain fabric: Proof-of-work vs. bft replication. In *iNetSec*, pages 112–125. Springer, 2015.