AccountNet: Accountable Data Propagation Using Verifiable Peer Shuffling

Man-Ki Yoon
Department of Computer Science
North Carolina State University

***Abstract*—Collecting evidence of data that software systems produce and consume can provide critical information for reconstructing erratic behavior, tracing the origin of faults, and thus holding the responsible party accountable for particular consequences. However, when data propagates across trust boundaries with conflicting interests, they can be tempted to make evidence unprovable in order to avoid potential liability. Hence, we present a data propagation protocol that makes such attempts either detectable or ineffective by having data transfers witnessed by other network participants that collectively act as the prover for the data propagation process. The protocol builds an unstructured peer-to-peer overlay and is designed to disincentivize collusion among a malicious coalition of nodes by enforcing network participants to constantly exchange their partial views on the network in a random yet verifiable manner. A data producer or consumer who does not faithfully follow the protocol ends up having fewer witnesses from their side, making the network resistant to collusion with high probability. We derive the conditions under which this property holds and demonstrate the practicality and cost of our approach through the implementation of a distributed application built on top of the proposed protocol.**

***Index Terms*—accountability, peer-to-peer network, data distribution system**

I. Introduction

Modern software systems are increasingly driven by data; boosted by rapid development in artificial intelligence, software components make intelligent decisions using data produced by others. The advancement in communication technologies further enables these systems to integrate functions and resources distributed across wide-area networks (e.g., Machine Learning as a Service or MLaaS [1], [2]). However, as these applications become more inexplicable due to the high complexity and the data-driven nature of their decision-making processes, the issue of *accountability* arises – who should be held responsible for a faulty operation? This can be answered by collecting *irrefutable* evidence of *data propagation* among software components. This would not be an issue in traditional monolithic systems because the system developer/manufacturer is held solely responsible for any faulty/wrongful consequences. However, as these systems are increasingly integrating components from multiple parties (e.g., third-party vendors, outsourced services), it is becoming more necessary to hold each party accountable for its association with the data that it produces and consumes. A lack of accountability mechanism could lead to a situation where certain self-interested parties act unfaithfully to make their potentially faulty/wrongful behaviors unprovable; to avoid liability, hide unauthorized information flow, etc.

In this paper, we consider the problem of assigning accountability to data propagation across network nodes. Proving data propagation between network nodes is especially challenging unless there exists a trustworthy 'broker' that routes all data traffic for every pair of data producer and consumer. However, such a centralized entity can easily become a single point of failure and a performance bottleneck. Hence, we propose AccountNet, a decentralized protocol that is based on peer-to-peer overlay networking. In this protocol, a subset of the network participants, called *witness* nodes, act collectively as the prover for data propagation between a pair of data producer and consumer. They are selected randomly by the producer-consumer pair themselves from their (logical) neighborhoods and collect logs about what they relay between the pair.

The key challenge in this protocol lies in that nodes can be malicious or simply non-cooperative. More seriously, an ill-intentioned data producer or consumer is tempted to influence the selection of witness nodes to make the evidence collected by them unusable or favorable to him/her. This could be achieved by polluting its neighborhood (and the opposite side's) with as many colluders as possible using techniques such as Eclipse attacks [3], [4] in an attempt to have the colluders take up the majority of the witness group. Our solution to this challenge is to enforce the network nodes to shuffle their peers continuously with others in a *random* yet *verifiable* manner to prevent the neighborhoods from being manipulated. With this protocol, those who participate faithfully in the protocol are incentivized when forming a witness group. On the other hand, malicious nodes are either detectable or likely to form a separate overlay of their own. Our witness-forming algorithm takes advantage of this property – a malicious producer/consumer is likely to end up either (i) having to follow the protocol correctly or (ii) having fewer candidates from its side with high probability, making collusion attempts futile. We present an analysis for finding network parameters for a probabilistic tolerance to collusion, which we also validate experimentally. Lastly, as a case study, we implement a broker-less publish-subscribe communication layer on top of the AccountNet protocol, apply it to a cloud-based machine learning service, and discuss the cost of the approach as well as its practicality.

II. Problem Statement and Models

A. Motivating Scenario

Consider a robotic system that needs a vision-based inference task. Due to a lack of a proper hardware accelerator, such as a graphical processing unit, or a rigorous inference model, it decides to use an online machine learning service that returns inference results given images captured locally ('cloud robotics' [5], [6]). The ML service provider is aware of its subpar performance, and thus it is tempted *not* to leave any provable evidence of response to the service user to avoid any potential liability that could arise from an incorrect inference. Conversely, the service user, e.g., the robotic system, wants to leave provable evidence that can be used to assign culpability to

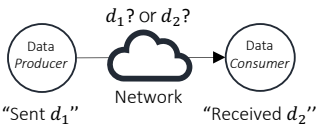


Fig. 1. Scenario when data producer and consumer report conflicting logs about data propagation between them.

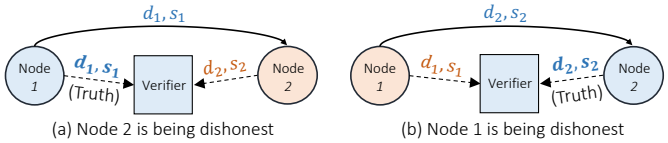


Fig. 2. Verifier cannot determine who is being dishonest given data-signature pairs (d, s) from the sender and the receiver.

the ML service provider in the event of an erroneous operation due to its incorrect inference. In these scenarios, a proper accounting protocol can help trace the origin of erratic behavior and thus resolve disputes among the parties involved, building transparency in data propagation into data-driven applications.

B. Problem Description

We consider a problem of assigning accountability for data production and consumption to network nodes. If there was a propagation of data D from a node to another, neither can deny the production/consumption of D. Similarly, neither can pretend to have produced/consumed data D if a propagation did not happen or $D' \neq D$ was propagated. Ill-intentioned data producers or consumers are motivated *not* to leave any provable evidence of these actions as aforementioned.

Suppose nodes are asked to write logs about data that they produce and consume, as depicted in Fig. 1. This scheme could allow a self-interested node to act *unfaithfully*. For instance, a malicious receiver may write to the log that it received d_2 when it actually received d_1. The receiver may even not enter any log at all as if no data transfer happened, accusing the producer. Data producers can have similar motivations. Especially when the nodes gain no benefit from being honest about their association with particular data propagation, they are tempted to fabricate/hide log entries. Of course, such behaviors (e.g., reporting fixed/random data or even nothing) would look obviously suspicious. However, there remains no evidence that can prove such malicious actions. Hence, when log entries conflict with one another, it is difficult to figure out whose log is correct unless the actual data transmissions are observable.

C. Can Digital Signatures Solve the Problem?

Suppose a digital signature is used for a data propagation from node 1 to node 2. That is, node 1 signs data d (or its hash) using its private key and sends both the data and the signature (which is simply a bitstring from other's perspective) to node 2. Now, both nodes are asked by a third-party verifier to present the data and signature pair that they sent/received, as illustrated in Fig. 2. Suppose that node 1 claims that it sent data d_1 (with a bitstring s_1) while node 2 claims that it received data $d_2 \neq d_1$ (with a bitstring s_2). Now, the verifier tries to determine who is correct given (d_1, s_1) and (d_2, s_2). The verifier possesses the public key of node 1, pk_1, and the signature verifying algorithm $V(pk, d, s)$ that outputs `valid` if s is a signature on d signed by the private key that corresponds to pk or `invalid` otherwise.

At a minimum, a malicious producer would never present an invalid signature. Hence, $V(pk_1, d_1, s_1)$ is always `valid`. Now, suppose the verifier sees $V(pk_1, d_2, s_2)$ = `invalid`. Does this necessarily indicate that node 2 is dishonest? We can consider the following cases (see Fig. 2):

(a) Node 2 is being dishonest (i.e., node 1 indeed sent d_1, s_1): Node 2 has fabricated s_2 by altering s_1 (given by node 1) or creating a completely-random one, and now it claims that node 1 sent d_2 with an invalid signature s_2 and that node 1 is 'pretending' that it sent d_1 and s_1.

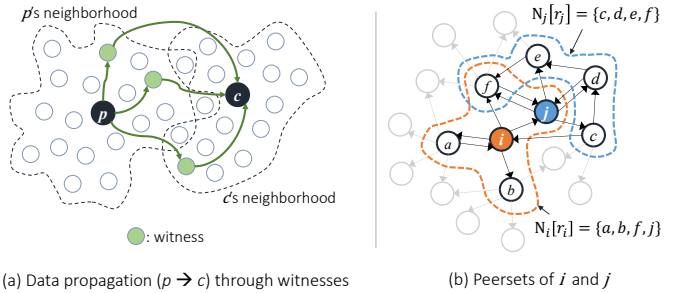
(b) Node 1 is being dishonest (i.e., node 1 actually sent d_2, s_2): Node 1 has given an invalid signature s_2 (e.g., a random bitstring) to node 2, and now it clams that node 2 is 'pretending' that it received d_2 and a fabricated signature s_2 in order to falsely accuse node 1.

Notice that (a) and (b) are *indistinguishable* from the verifier's perspective. That is, the verifier cannot determine whether the invalid signature s_2 is (a) fabricated by node 2 or (b) given by node 1. This ambiguity is caused because a digital signature no longer carries its non-repudiation property when it is not directly provided by the signer. This is why node 1 in Fig. 2 would never present an invalid signature to the verifier, whereas the verifier cannot verify the non-repudiation property of s_2 which is 'told by' node 2 who may or may not be honest. Therefore, digital signatures cannot solve the accountability problem stated above.

D. System and Adversary Models

We assume that nodes can join and leave the network at arbitrary times. Each node v_i is associated with a unique identifier such as IP (Internet Protocol) address with a port number, denoted by addr_i. Note that our use of peer-to-peer overlay is *not* for efficient search of data items. That is, the addresses of data sources are publicly known, and we do not concern about how nodes obtain the address information. We also assume that data is eventually delivered within a bounded time unless the connection is permanently lost and that there is no transmission error (e.g., data is delivered over TCP/IP).

Nodes are either benign or malicious. *Benign* nodes faithfully follow the data propagation protocol that will be presented in the rest of the paper. Nodes are motivated to act benign especially if they want their association with particular data production/consumption to be provable. *Malicious* nodes act in favor of the colluding nodes that share the same interest (e.g., from the same administrative domain). There can also be nodes that are arbitrarily Byzantine. However, from the security analysis point of view, we do not differentiate between malicious nodes and Byzantine nodes. Hence, we view both types as malicious nodes – they cannot be assumed to follow the protocol. There can be multiple collusion groups. However, we view all of them (regardless of where they belong) as a single, large coalition of colluding nodes for the worst-case security analysis. We assume that a network of size $|V|$ is expected to have about $p_m|V|$ malicious nodes (equivalently, a node can be malicious with a probability p_m), but they are unknown.



(a) Data propagation ($p \rightarrow c$) through witnesses (b) Peersets of i and j

Fig. 3. Peer-to-peer overlay of AccountNet.

Sybil attack [7] is a serious threat to peer-to-peer networking and it can only be mitigated, not prevented. Hence, we assume that a mitigation approach is taken; for instance, limiting IP mapping [8], [9], using a trusted authority to certify identity [3], [10], [11], or even using computational puzzles [12].

III. Overview of AccountNet

In AccountNet, all network participants serve as *witnesses* to data propagation between others as illustrated in Fig. 3(a). Due to the absence of a centralized entity that has a full view of the entire network, these witness nodes are selected by a producer and consumer pair when they establish a data propagation channel. The witnesses relay data between them and capture evidence about it which can be used later to resolve any dispute between the producer and consumer. However, some witnesses could be malicious, hence they could alter/drop the data being relayed or even fake the logs in the interest of a colluding producer/consumer. Furthermore, the data producer/consumer may influence the witness selection process to fill the witness group with as many colluding nodes as possible.

Our solution to these challenges is to let nodes construct a peer-to-peer overlay network and to incentivize those who learn about more peers. Specifically, nodes maintain information about a small set of peers, which we call the *peerset*. Fig. 3(b) shows examples. Nodes repeatedly perform a gossip-based peer sampling [13], [14] to update their partial view on the network, which is known to lead the network to behave uniform-randomly; by *shuffling* one's peerset with another, nodes are likely to have peers as if sampled uniform-randomly from the whole network. It is important to note that the overlay is *unstructured* because otherwise (i.e., structured overlay such as [15]–[17]), malicious nodes may take advantage of a structural property (e.g., by occupying particular addresses) to have a biased set of peers [3], [18].

The key idea of AccountNet is to enforce nodes to shuffle their peerset in a *verifiable* way. Nodes are required to keep records of the peerset shuffling, which can be verified by any other nodes. When a node performs a shuffling, the records are given to the counterpart and verified whether the node's peerset has been manipulated. When drawing witnesses between a data producer and consumer pair, a set of witnesses are selected randomly from their neighbors (in the overlay graph, not in a geographical sense) with weights proportional to the neighborhood size. This *disincentivizes* malicious nodes that form clusters among themselves. As will be seen later, nodes are likely to have larger neighborhoods if following the protocol faithfully.

In summary, (i) each node maintains a set of peers that are constantly updated by a verifiable peer shuffling (Sec. IV) and (ii) when establishing a data channel, a group of witnesses is sampled, also in a verifiable manner, from the neighborhoods of data producer and consumer (Sec. V).

IV. Verifiable Peer Shuffling

A. Protocol Detail

Nodes perform three types of protocol operations: *join*, *shuffle*, and *leave*. Each operation creates records of their interactions with others which can be verified by any others.

Peerset Update History: In order to prevent a malicious node from arbitrarily manipulating its peerset, nodes are required to keep records of changes in their peersets. These records, called *Peerset Update History*, are provided to other nodes (when shuffling peers or forming a witness group) to prove from whom and when each of the peers is learned of.

$\mathsf{N}_i[r_i]$ denotes the peerset of node v_i as of round r_i. Its peerset update history at round r_i is an ordered-list of entries:

$$\Omega_i[r_i] = (\omega_{i,0}, \omega_{i,1}, \ldots, \omega_{i,r_i}).$$

An entry is defined by $\omega_{i,r} = (v_j, \sigma_j(\mathsf{nonce}), \mathsf{nonce}, \mathsf{out}, \mathsf{in}, \mathsf{fill})$ where v_j is the node that v_i interacted with (e.g., shuffling counterpart), and $\sigma_j(\mathsf{nonce})$ is a message signed on an operation-specific nonce that prevents v_i from forging the entry. For example, when shuffling its peerset, the counterpart's round number is used as the nonce. The last three fields, i.e., out, in, and fill, specify how the node's peerset changed at the r^{th} round. For instance, out is the set of peers that are removed from v_i's peerset due to a shuffling. The detail about these fields will be explained shortly below.

Network join: Nodes can join the network through any other node that is already in the network, as in general peer-to-peer overlay networks [9], [13], [19], [20]. We do not assume a particular method of obtaining the address of a bootstrap node; it can be obtained through certain out-of-band means, e.g., using a directory service or by scanning a local network.

Let v_{bn} denote the bootstrap node through which v_i joins the network. v_{bn} gives v_i the list of its neighbor nodes with depth up to d (see Fig. 7). Upon receiving the list (and storing it internally for verification purposes), v_i randomly selects up to f nodes from it, using the same verifiable random sampling that is used for peer shuffling (which will be explained in 'Verifiable shuffling' below). The set of the selected nodes becomes v_i's initial peerset, $\mathsf{N}_i[0]$. The bootstrap node also creates an *entry stamp*, $\sigma_{bn}(\mathsf{addr_i})$, for v_i. Then, v_i creates the first entry of the peerset update history:

$$\omega_{i,0} = (v_{bn}, \sigma_{bn}(\mathsf{addr_i}), \mathsf{addr_i}, \mathsf{out} = \emptyset, \mathsf{in} = \mathsf{N}_i[0], \mathsf{fill} = \emptyset).$$

Any node who is given this entry can verify the origin of v_i using the entry stamp $\sigma_{bn}(\mathsf{addr_i})$.

Shuffling: Each node v_i repeatedly shuffles its peerset randomly in rounds by exchanging a random sample of the set with that of another peer's. Suppose it is v_i's $r_i{}^{th}$ round. The high-level shuffling protocol (adopted from [21]) is as follows:

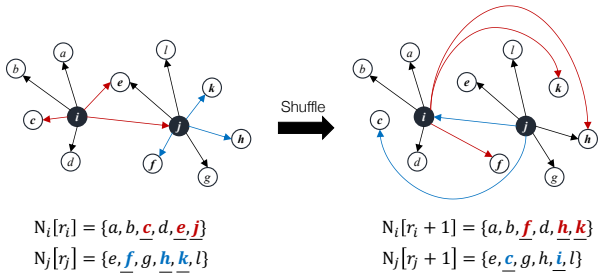


Fig. 4. Peer shuffling between v_i and v_j.

1) v_i first selects a node v_j randomly from its current peerset $\mathsf{N}_i[r_i]$ as the shuffling counterpart.
2) v_i selects $L-1$ more nodes, denoted by set $\mathcal{A}$, from its peerset and sends $\mathcal{A} \cup \{v_i\}$ to v_j.
3) v_j selects L nodes, denoted by set $\mathcal{B}$, from its current peerset $\mathsf{N}_j[r_j]$ and sends $\mathcal{B}$ to v_i.
4) v_i updates it peerset by removing $\mathcal{A} \cup \{v_j\}$ and then adding those in $\mathcal{B}$. Similarly, v_j updates it peerset by removing $\mathcal{B}$ and then adding those in $\mathcal{A} \cup \{v_i\}$.

Fig. 4 shows an example shuffling. In the last step of the protocol, space can be left after the update due to the possibility of duplication; e.g., some node $v_x \in \mathcal{B}$ already exists in $\mathsf{N}_i[r_i] - (\mathcal{A} \cup \{v_j\})$. In such a case, the space is filled with some of the peers sent to the other side (e.g., drawn from $\mathcal{A} \cup \{v_j\}$ for v_i or from $\mathcal{B}$ for v_j). Let us denote them by $\mathcal{C}_i$ and $\mathcal{C}_j$ for v_i and v_j respectively. This last step prevents the peerset size from diminishing to zero.

In AccountNet, a shuffling results in creating a peerset update entry for each side. First, v_i's entry is

$$\omega_{i,r_i} = (v_j, \sigma_j(r_j), r_j, \mathsf{out} = \mathcal{A} \cup \{v_j\}, \mathsf{in} = \mathcal{B}, \mathsf{fill} = \mathcal{C}_i).$$

The corresponding entry for v_j is

$$\omega_{j,r_j} = (v_i, \sigma_i(r_i), r_i, \mathsf{out} = \mathcal{B}, \mathsf{in} = \mathcal{A} \cup \{v_i\}, \mathsf{fill} = \mathcal{C}_j).$$

After this exchange, their peersets become

$$\mathsf{N}_i[r_i + 1] = \Big(\mathsf{N}_i[r_i] - (\mathcal{A} \cup \{v_j\})\Big) \cup \mathcal{B} \cup \mathcal{C}_i \text{ and}$$

$$\mathsf{N}_j[r_j + 1] = \Big(\mathsf{N}_j[r_j] - \mathcal{B}\Big) \cup \Big(\mathcal{A} \cup \{v_i\}\Big) \cup \mathcal{C}_j,$$

respectively. Notice that v_i, who initiated the shuffling, becomes a peer of v_j, if it was not, after this shuffling.

Example 1. *For the shuffling shown in Fig. 4, the following entries are created at v_i and v_j, respectively:*

$$\omega_{i,r_i} = (v_j, \sigma_j(r_j), r_j, \mathsf{out} = \{c, e, j\}, \mathsf{in} = \{f, h, k\}, \mathsf{fill} = \emptyset)$$
$$\omega_{j,r_j} = (v_i, \sigma_i(r_i), r_i, \mathsf{out} = \{f, h, k\}, \mathsf{in} = \{c, e, i\}, \mathsf{fill} = \{h\})$$

Fig. 5 shows how the peer shuffling explained above leads nodes to discover others. The color of each element in the heatmaps represents whether node i and j have shuffled their peersets with each other at least once. If a network was (roughly) partitioned, the heatmaps would show clusters. In a well-shuffled network, old nodes and newer nodes can discover each other as the figure shows.

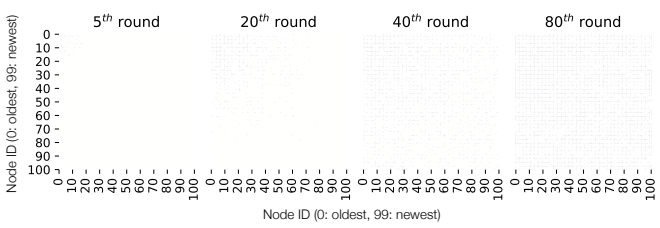


Fig. 5. Nodes discover others through a series of random peer shuffling.

Algorithm 1: `Shuffle`(v_i)

r_i: v_i's current round number
$h_0, \pi_0 \leftarrow \mathsf{vrf}_i(r_i)$
$v_j \leftarrow$ `select`$(\mathsf{N}_i[r_i], h_0)$ `// Draw a shuffle partner`
$r_j \leftarrow v_j$'s round number
$\mathcal{A} \leftarrow \emptyset$
$\mathcal{P}_i \leftarrow \emptyset$ `// Proofs of sampling`
$k \leftarrow 1$
while $|\mathcal{A}| \leq L - 1$ **do**
  $h_k, \pi_k \leftarrow \mathsf{vrf}_i(r_j || k)$
  $v_k \leftarrow$ `select`$(\mathsf{N}_i[r_i] - \{v_j\}, h_k)$
  if $v_k \notin \mathcal{A}$ **then**
    $\mathcal{A} \leftarrow \mathcal{A} \cup \{v_k\}$
    $\mathcal{P}_i \leftarrow \mathcal{P}_i \cup \{\pi_k\}$
  end
  $k \leftarrow k + 1$
end
Send v_j (i) $\mathcal{A} \cup \{v_i\}$, (ii) $\mathcal{P}_i \cup \{\pi_0\}$, (iii) $\mathsf{N}_i[r_i]$, (iv) $\Omega_i[r_i]$, (v) r_i and $\sigma_i(r_i)$
Receive from v_j (i) $\mathcal{B}$, (ii) $\mathcal{P}_j$, (iii) $\mathsf{N}_j[r_j]$, (iv) $\Omega_j[r_j]$, (v) r_j and $\sigma_j(r_j)$
if `Verify`$(\Omega_j[r_j], \mathsf{N}_j[r_j], \mathcal{P}_j, \mathcal{B}) =$ `True` **then**
  `Update`$(\mathsf{N}_i[r_i], v_j, \sigma_j(r_j), r_j, \mathcal{A}, \mathcal{B})$
end

Algorithm 2: `Select`(X, h)

X: ordered list of nodes
$Q \leftarrow \lceil \log_2 |X| \rceil$
$index \leftarrow h \mid (2^Q - 1)$ `// Bitwise-AND`
return $X[index]$ *or Null if* $index > |X|$

Verifiable shuffling: The shuffling protocol presented above can be exploited by a malicious node; it may shuffle only with colluding nodes or try to pollute other benign nodes's peersets with its colluders. To prevent such a node from arbitrarily manipulating peer shuffling, we make the procedure *verifiable* yet still *random*. The key idea is that given the current peerset, the next shuffling-counterpart is *deterministic*. However, the random sample cannot be determined until a nonce is provided by the counterpart (e.g., its round number). Algorithm 1 presents a pseudocode of the verifiable shuffling algorithm. It calls Algorithm 2 to select a node from list X given a hash value h. After node i receives a sample of peers from node j, it verifies the correctness of the sampling by

Algorithm 3: `Update`$(\mathsf{N}_i[r_i], v_j, \sigma_j(r_j), r_j, \mathcal{A}, \mathcal{B})$

$\mathsf{N}_i[r_i + 1] \leftarrow \mathsf{N}_i[r_i] - \{v_j\} - \mathcal{A}$
$\mathsf{N}_i[r_i + 1] \leftarrow \mathsf{N}_i[r_i + 1] \cup \mathcal{B}$
if $\mathsf{N}_i[r_i + 1]$ *is not full* **then**
  $\mathcal{C}_i \leftarrow$ select nodes randomly from $\mathcal{A} \cup \{v_j\}$ to fill $\mathsf{N}_i[r_i + 1]$
  $\mathsf{N}_i[r_i + 1] \leftarrow \mathsf{N}_i[r_i + 1] \cup \mathcal{C}_i$
else
  $\mathcal{C}_i \leftarrow \emptyset$
end
$\omega_{i,r_i} = (v_j, \sigma_j(r_j), r_j, \mathsf{out} = \mathcal{A}, \mathsf{in} = \mathcal{B}, \mathsf{fill} = \mathcal{C}_i)$
$\Omega_i[r_i] \leftarrow \Omega_i[r_i] \cdot \omega_{i,r_i}$ `// Add new entry`
$r_i \leftarrow r_i + 1$

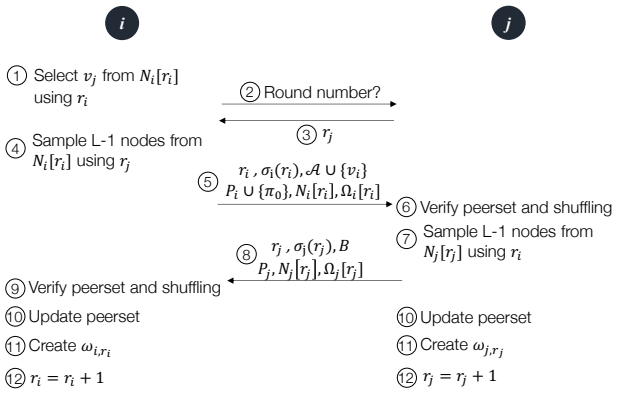


Fig. 6. Shuffling process between v_i and v_j.

calling `Verify`, which follows similar steps as in Algorithm 1 (hence not defined separately). If the verification is passed, Algorithm 3 is called to update the node i's peerset and to extend the peerset update history.

Fig. 6 summarizes the verifiable shuffling process. First, v_i, who initiates a shuffling, calls a verifiable random function [22], using its current round number r_i as the input, to select the next shuffling counterpart. Then, it asks the selected node, v_j, for its round number r_j. Now, using it as a nonce, v_i selects $L-1$ nodes from its own peerset. Specifically, for $k = 1, 2, \ldots$, it computes

$$h_k, \pi_k = \mathsf{vrf_i}(r_j || k),$$

where h_k is the hash output and π_k is the proof that h_k is correct. Because r_j, the round number of the counterpart, is used as a part of the nonce, v_i cannot pre-select peer samples (similarly, v_j uses v_i's round number when sampling nodes from its peerset). Using the hash output h_k as the index, v_i retrieves a node from the sorted list of peers. Because the size of the hash space (e.g., 512-bit) is much larger than that of the peer list, we use the first (or last) Q bits of the hash output as the index, where Q is the smallest integer such that $2^Q \geq |X|$ where X is a peer list. Note that the k^{th} selection may not pick a new peer, in which case h_k and π_k are discarded and a new hash is computed for $k+1$. This procedure repeats until $L-1$ distinct nodes are selected. v_i (resp. v_j) includes the proofs of the hash outputs as well as its current peerset when sending the random sample of peers to v_j (resp. v_i).

Now, given v_i's peerset, its round number, and the proofs generated by the verifiable random function above, v_j can verify if (i) itself should indeed be the shuffling counterpart for v_i's current round and (ii) v_i's random peer samples are correct, by performing the same sampling procedure as if it is v_i. Because v_j knows the input to the verifiable random function, i.e., its round number r_j, it can verify the peer samples from v_i. Hence, a malicious node would not try to make up a biased sample because it is easily detectable. However, it could still try to manipulate the random sampling by forging the peerset (where the sample is drawn from), not the sample; some nodes can be added (especially colluding ones) or removed (especially benign ones) as if they have existed or not. However, such attempts can be detected by *reconstructing* the counterpart's peerset from the peerset update history.

Peerset reconstruction: Suppose a node received the peerset update history of v_i, i.e., $\Omega_i[r_i] = (\omega_{i,a}, \omega_{i,a+1}, \ldots, \omega_{i,r_i})$ where $\omega_{i,a}$ is the oldest entry. Then, the node can reconstruct v_i's peerset, $\widehat{\mathsf{N}_i}[r_i]$, by iterating over the entries in the chronological order. Specifically, for each entry

$$\omega_{i,r} = (v_k, \sigma_k(\mathsf{nonce}), \mathsf{nonce}, \mathsf{out}, \mathsf{in}, \mathsf{fill})$$

we apply the following recursive operation:

$$\widehat{\mathsf{N}_i}[r] = \left(\widehat{\mathsf{N}_i}[r-1] - \mathsf{out}\right) \cup \mathsf{in} \cup \mathsf{fill}$$

where $\widehat{\mathsf{N}_i}[a-1] = \emptyset$.

Note that $\widehat{\mathsf{N}_i}[r] \subseteq \mathsf{N}_i[r]$ must hold. Hence, it is node v_i's responsibility to provide a sufficiently long peerset update history that is enough to reconstruct $\widehat{\mathsf{N}_i}[r] = \mathsf{N}_i[r]$. In fact, nodes do not need to send a long list of entries. Suppose we select L nodes from the peerset in each round. Letting f be the maximum size of the peerset, the probability that a node v_x stays in the peerset after one round of shuffling is $\frac{f-L}{f}$. Hence, the probability that the node stays in the peerset for m consecutive rounds is $(\frac{f-L}{f})^m$. If $f = 10$ and $L = 5$, the probability is 0.1% and 0.003% for $m = 10$ and 15, respectively.

When shuffling peerset, nodes can send $\Omega_i[r_i] = (\omega_{i,a}, \omega_{i,a+1}, \ldots, \omega_{i,r_i})$ where $\omega_{i,a}$ contains (in the 'in' field) the oldest node(s) in the current peerset, with which it is guaranteed that $\widehat{\mathsf{N}_i}[r] = \mathsf{N}_i[r]$. When requested by another node, v_i may provide an old-entry lookup service, which can be used for tracing back the origin of a particular node.

Peerset verification: Before node v_i updates its peerset using samples received from v_j, it verifies if v_j indeed performed an unbiased random sampling from its current peerset. For this, v_i first checks if $\widehat{\mathsf{N}_j}[r_j] = \mathsf{N}_j[r_j]$ by reconstructing $\widehat{\mathsf{N}_j}[r_j]$ from v_j's peerset update history $\Omega_j[r_j]$ as explained above. Furthermore, for each entry $\omega_{j,r}$ in $\Omega_j[r_j]$

$$\omega_{j,r} = (v_k, \sigma_k(r'), r', \mathsf{out}_j, \mathsf{in}_j, \mathsf{fill}_j),$$

v_i can ask v_k for the corresponding entry

$$\omega_{k,r'} = (v_j, \sigma_j(r), r, \mathsf{out}_k, \mathsf{in}_k, \mathsf{fill}_k)$$

which should be found in v_k's peerset update history if v_j and v_k performed the shuffling. Furthermore, these two entries should satisfy $\mathsf{out}_j = \mathsf{in}_k$ and $\mathsf{in}_j = \mathsf{out}_k$. Also, the following invariants must hold for each $\omega_{j,r}$:

- $v_k \in \widehat{\mathsf{N}_j}[r]$ if v_j initiated the shuffling: the counterpart, v_k, was selected from v_j's peerset at its r^{th} round;
- $\mathsf{out}_j \subseteq \widehat{\mathsf{N}_j}[r]$: a random subset of peers given to the counterpart was sampled from v_j's peerset at its r^{th} round.

These are verified by the same verification process used during the peer shuffling explained earlier.

Leaving network: Nodes may leave the network gracefully or abruptly. In case of a graceful leaving (of node v_x), the node may inform its peers of the leaving. Nevertheless, there can be nodes in the network that have v_x as a peer, but not vice versa. Hence, they will not be aware of v_x's leaving until it is selected as a shuffling counterpart (or a random ping test fails). Thus, in our protocol, we assume nodes leave the network ungracefully. Suppose a node v_i has detected that v_x is inactive. Then, v_i informs its own peers of v_x's leaving, sending a signed message

$\sigma_i(r_i)$. Each of the peers, say v_j, may individually check v_x's liveness (e.g., ping test [11]). Upon successful confirmation of v_x's leaving, v_j adds the following entry to its peerset update history regardless of v_x being in its current peerset:

$$\omega_{j,r_j} = (v_i, \sigma_i(r_i), r_i, \mathsf{out} = \{v_x\}, \mathsf{in} = \emptyset, \mathsf{fill} = \emptyset).$$

This entry should be found in the peerset update history of every $v_j \in \mathsf{N}_i[r_i]$.

B. Security Analysis

An adversary will try to maximize the footprint of malicious nodes in the network's peersets so that they are likely to be selected as witnesses for many (or particular) data channels. Hence, the adversary is motivated to target a particular shuffling counterpart and also to compose a peer sample set with malicious nodes by *not* sampling randomly from its peerset or even by faking it as well as the peerset update history. In what follows, we explain how such attempts are deterred or detectable and why a group of malicious nodes would have to either (i) follow the shuffling protocol correctly (thus acting as benign nodes during shuffling) or (ii) separate themselves from the network of benign nodes.

How difficult is it for a malicious node to manipulate its peerset? Suppose that a malicious node v_m wants to manipulate its peerset for the $r + 1^{th}$ round, i.e., $\mathsf{N}_m[r+1]$, so that certain peers are included/excluded. For this, it should be able to fabricate its peerset update history $\Omega_m[r] = (\omega_{m,0}, \omega_{m,1}, \ldots, \omega_{m,r-1}, \omega_{m,r})$ that can reconstruct the desired $\mathsf{N}_m[r+1]$. Let us first consider the latest shuffling:

$$\omega_{m,r} = (v_c, \sigma_c(r'), r', \mathsf{out}, \mathsf{in}, \mathsf{fill}),$$

which indicates that v_m performed a shuffling with a node v_c. First of all, for the entry to look valid, v_c must be in collusion with v_m. Otherwise, $\omega_{m,r}$ cannot be fabricated because v_i cannot forge $\sigma_c(r')$ unless it possesses v_c's private key. Now, fabricating a valid entry $\omega_{m,r}$ is equivalent to finding a solution to the following equation:

$$\mathsf{N}_m[r+1] = \Big(\mathsf{N}_m[r] - \mathsf{out}\Big) \cup \mathsf{in} \cup \mathsf{fill}. \tag{1}$$

A similar equation for the colluding node v_c also needs to be solved together. Because r is fixed, v_m and v_c must be able to find $\mathsf{N}_m[r]$, $\mathsf{N}_c[r']$, and r' that satisfy Eq. (1). If v_m initiated the shuffling, the following additional constraints must be satisfied:

- $v_c \in \mathsf{N}_m[r]$,
- $\mathtt{vrf}_{\mathtt{m}}(r)$'s hash output is mapped to the index of v_c in the 'sorted' list of $\mathsf{N}_m[r]$.

Notice from the verifiable shuffling procedure that there are dependencies among the variables:

- A change in r' causes a change in the random sample from v_m which is the out field in Eq. (1);
- The out field is also dependent on the v_m's peerset at the r^{th} round, i.e., $\mathsf{N}_m[r]$, that needs to be determined as well;
- A change in $\mathsf{N}_c[r']$ causes a change in the random sample from v_c which is the in field;
- The fill field is determined by $\mathsf{N}_m[r]$, out, and in.

If a combination of the four variables does not satisfy Eq. (1), one should backtrack and try a different combination. That is, each variable cannot be determined independently. Hence, v_m and v_c have to *enumerate* the combinations of the variables. Even if a solution to Eq. (1) was found, it becomes the base constraint for the next search for the solution to $\omega_{m,r-1}$, that is, $\mathsf{N}_m[r] = (\mathsf{N}_m[r-1] - \mathsf{out}) \cup \mathsf{in} \cup \mathsf{fill}$, and also for v_c's peerset update histories. Furthermore, as the length of the v_m's peerset update history that needs to be fabricated increases, more colluding nodes are involved, which makes the search space grow exponentially while further reducing the solution space.

Does a malicious node gain any benefit from having benign peers? As discussed above, malicious nodes can collude with each other to manipulate their peersets. Now, suppose a malicious node v_m has a benign node v_b as a peer, i.e., $v_b \in \mathsf{N}_m[r_m]$. Once v_m performs a shuffling with v_b, the former becomes a peer of the latter (see Fig. 4). Since then, v_m can be asked to do a shuffling by v_b or other benign nodes who learned of v_m from v_b through peerset shuffling, which cuts the chain of collusion mentioned above. Hence, v_m can no longer manipulate its peerset (and peerset update history) without being detected by benign nodes. In addition, having a benign neighbor would only decrease the chance of its colluding nodes being selected as witnesses. Also, as will be explained shortly, common neighbors are excluded from the witness selection, and hence malicious nodes would not want to be peers of benign nodes. Therefore, v_m, a malicious node, has no benefit of having a peer relationship (in both directions) with benign peers. Following this rationale, a group of colluding nodes would create a separate network in which nodes have no peer relationship with those in the benign network.

Malicious bootstrap node: If a new node v_i joins the network through a malicious bootstrap node v_{bn}, the latter may try to fill v_i's initial peerset with other malicious nodes. In the worst case, the neighbor list provided by v_{bn} can consist only of malicious nodes. However, the size of such a neighbor set is likely to be small compared to average cases (i.e., a neighbor of a benign node), as will be explained in the next section. Hence, the node v_i can decide whether to join the network through v_{bn} or try a different bootstrap node based on the size of the neighborhood given by v_{bn}. If a node becomes a part of a malicious cluster, it would encounter fewer and fewer new peers as it performs shuffling. This is a way to detect if one has joined the network through a malicious bootstrap node.

V. Witness Group Formation

As explained in Sec. III, a set of witness nodes are randomly drawn from the neighborhoods of data producer and consumer. Viewing the AccountNet network as a directed graph $G = (V, E)$ where V is the set of network participants and E is a set of directed edges $E = \{(v_a, v_b) | (v_a, v_b) \in V^2 \wedge v_b \in \mathsf{N}_a\}$ that represents peer relationships, the neighborhood of node v_i with *depth* up to d is defined by

$$\mathsf{N}_i^d = \{v_x | \mathsf{dist}(v_i, v_x) \leq d\}$$

where $\mathsf{dist}(v_a, v_b)$ is the length of the shortest path from v_a to v_b. (Note: in this section, we simplify the notations by omitting round numbers, e.g., $\mathsf{N}_i = \mathsf{N}_i[r_i]$, unless otherwise

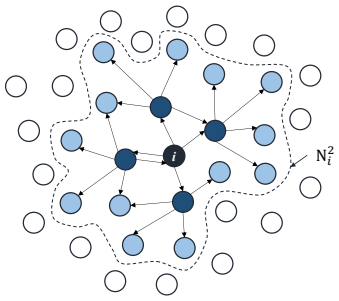


Fig. 7. The neighborhood of v_i with depth up to 2.

needed.) If v_b is a direct peer of v_a, $\mathsf{dist}(v_a, v_b) = 1$. By the definition, $\mathsf{N}_i = \mathsf{N}_i^1$. Hence, a witness may not be a direct peer of data producer/consumer. A node can find its neighborhood by flooding a query message, with the radius-limit set to d, to every peers. Fig. 7 shows an example of N_i^2.

Suppose a data channel between nodes v_i and v_j is to be created. Once N_i^d and N_j^d are found, v_i and v_j exchange the neighbor sets (and the up-to-date peerset update histories) for verification. Using the same procedure in Sec. IV, v_i can verify the correctness of N_j^d by traversing from v_j; for each node on a search path, v_i verifies the node's peerset by requesting its peerset update history. v_i can reduce search and verification efforts by random walking.

Upon successful verification, v_i and v_j compute the witness-allocation ratios proportional to their neighborhood sizes:

$$\alpha_i = |\mathsf{N}_i^d|/(|\mathsf{N}_i^d| + |\mathsf{N}_j^d|), \qquad \alpha_j = |\mathsf{N}_j^d|/(|\mathsf{N}_i^d| + |\mathsf{N}_j^d|).$$

Letting $|W|$ be the agreed-upon size of the witness group, v_i and v_j sample $\alpha_i|W|$ and $\alpha_j|W|$ nodes from their respective neighbor sets in the same way as in the peerset shuffling, which is then verified by the other side before finalizing the witness group. Here, we exclude those nodes on both sides because the chance of node $v_x \in \mathsf{N}_i^d \cap \mathsf{N}_j^d$ being selected is twice that of the others. This could be taken advantage of by a malicious side by polluting the opposite side's neighborhood with its colluders.

As aforementioned, the witnesses act as 1-hop relay between a producer and a consumer. The captured evidence can be used later by a third-party resolver using a simple majority scheme. The rest of this section answers the following question: given the probability of nodes being malicious, how the network parameters should be chosen to probabilistically guarantee that more benign witnesses are selected than malicious ones? The key is that neighborhoods should not be too small or too large.

A. Neighborhood Size

Given a depth limit d and the peerset size f, $|\mathsf{N}_i^d| \leq (f^{d+1} - f)/(f-1)$ holds for any node v_i. This is because a node's neighborhood is maximal when none of the neighbors share any peers. In this case, the neighborhood forms a perfect f-ary tree with a depth of d. Hence, the number of neighbors (excluding the root) in this case, denoted by $|\mathsf{N}^d|^*$, is

$$|\mathsf{N}^d|^* = \sum_{k=1}^{d} f^k = (f^{d+1}-1)/(f-1) - 1 = (f^{d+1}-f)/(f-1).$$

Now, the *expected* size of one's neighborhood, denoted by $\overline{|\mathsf{N}^d|}$, can be calculated by a combinatorial analysis. Algorithm 4

Algorithm 4: `expected_neighborhood_size`$(|V|, f, d)$

```
n ← 1, #iter ← (f^d − 1)/(f − 1)
for i = 0, 1, ..., #iter − 1 do
    for k = 0, 1, ..., f do
        Pr(X = k) ← C(n−1, k) C(|V|−n, f−k) / C(|V|−1, f)
    end
    Δn ← Σ_{k=0}^{f} (f − k) · Pr(X = k)
    n ← n + Δn
end
return n − 1
```

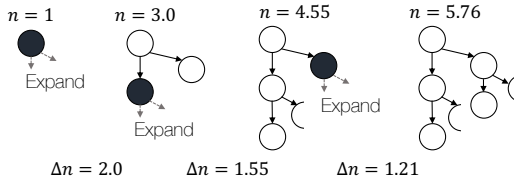


Fig. 8. Finding the expected neighborhood size for $|V| = 10$, $f = 2$, $d = 2$. Each step can be viewed as selecting f peers from the pool of $|V| - n$ nodes.

presents the pseudocode. Intuitively speaking, the algorithm can be viewed as forming an f-ary tree with depth up to d. Fig. 8 illustrates the algorithm. It starts with a single node ($n = 1$), i.e., the root of the tree. Then, its f peers are randomly drawn from the rest of the network and added to the tree as the children of the root node. This repeats for each of the newly-added nodes. Now, some of the randomly-chosen nodes may have already been added to the tree, which is especially the case when the network size is small. Thus, a random expansion may add fewer than f new nodes to the tree. Suppose that there are currently n nodes in the tree and that a random draw is to be performed. To find the expected increase in the number of distinct nodes (Δn) due to this sampling, we calculate the probability of picking k peers from the existing nodes (that have already been included in the neighbor set, i.e., the f-ary tree) and that of picking $f-k$ peers from the rest (i.e., $|V|-n$ nodes). For this, let X be a random variable that represents the number of peers selected from those already in the tree. Then, it follows the hypergeometric distribution:

$$\Pr(X = k) = \binom{n-1}{k}\binom{|V|-n}{f-k} \Big/ \binom{|V|-1}{f}. \qquad (2)$$

The denominator represents the number of ways to select f peers from the entire network (i.e., $|V| - 1$ nodes). The numerator represents: k of them are selected from those already added (i.e., $n-1$ nodes) *and* $f-k$ from the rest (i.e., $|V|-n$ nodes). Then, the neighborhood size is increased by $f - k$. Using Eq. (2), we calculate $\Pr(X = k)$ for $0 \leq k \leq f$, from which we obtain the expected increase Δn:

$$\Delta n = \sum_{k=0}^{f} (f-k) \cdot \Pr(X = k).$$

This process (i.e., $n = n + \Delta n$) repeats for $\frac{f^d-1}{f-1}$ times, the number of internal nodes in a perfect f-ary tree with a depth of d. The expected size of one's neighborhood is then $n - 1$ (because the starting node should be excluded). For a very large network, each expansion is likely to add $\Delta n = f$ new nodes to the neighbor set because $\Pr(X = 0) \approx 1$. Thus the expected neighborhood size is maximized to $(f^d-1)/(f-1) \cdot f = |\mathsf{N}^d|^*$.

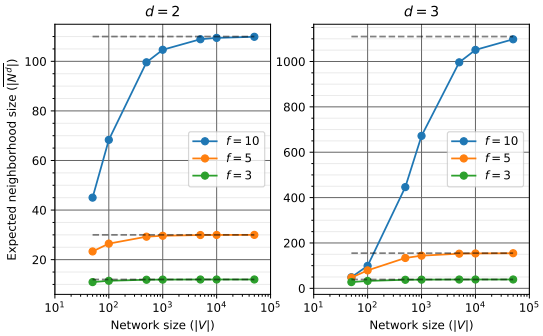


Fig. 9. Expected neighborhood size, $\overline{|\mathsf{N}^d|}$, for different combinations of $|V|$, f, and d. Notice the difference in the y-scales.

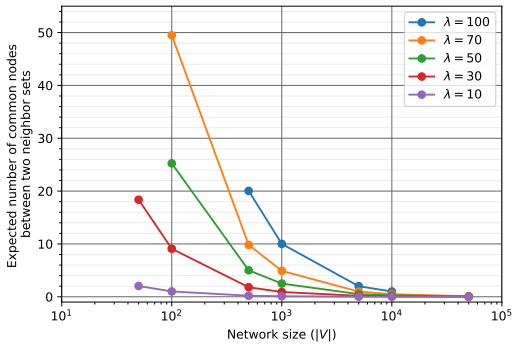


Fig. 10. Expected number of common nodes between two neighbor sets of same size $\lambda = |\mathsf{N}_i^d| = |\mathsf{N}_j^d|$).

Example 2. *Suppose the network consists of $|V| = 10$ nodes, the maximum peerset size is $f = 2$, and the depth limit is $d = 2$. Fig. 8 shows the steps to calculate the expected neighborhood size. The calculation starts with $n = 1$. The first $\Delta n = 2.0$ is because there is no way to select a peer from the existing nodes in this case (hence $\Pr(X = 1) = \Pr(X = 2) = 0$). Now, given three nodes ($n = 3.0$), we expand the neighbor set by calculating Eq. (2):*

$$\Pr(X = k) = \binom{2.0}{k}\binom{7.0}{2-k}/\binom{9}{2}$$

which is approximately 0.58, 0.39, 0.03 for $k = 0, 1, 2$, respectively. Thus, $\Delta n = 0.58 \cdot 2 + 0.39 \cdot 1 + 0.03 \cdot 0 = 1.55$. This step repeats once more, and finally $n = 5.76$. Therefore, the expected neighborhood size is $\overline{|\mathsf{N}^2|} = n - 1 = 4.76$.

Fig. 9 shows the expected neighborhood size for different combinations of $|V|$, f, and d (the dashed lines represent the maximum sizes, $|\mathsf{N}^d|^*$). For $f = 3$ and 5, it quickly converges to the maximum size as the network size increases. This indicates that nodes are unlikely to share neighbors because the network is large enough relative to one's neighborhood.

B. Finding Network Parameters

Let us consider a sampling of witnesses between two sides, v_i and v_j. Without loss of generality, let us assume that the former is benign and the latter is malicious. Then, the question is what values the network parameters d and f, i.e., the depth limit and maximum peerset size, should have so that there are more benign witnesses than malicious ones with a high probability.

Suppose a node is malicious with probability p_m. We can first consider the case when the malicious group follows the protocol faithfully, in which case v_i and v_j would have $|\mathsf{N}_i^d| \cdot p_m$ and $|\mathsf{N}_j^d| \cdot p_m$ malicious nodes in their neighborhoods respectively. However, because the nodes in $\mathsf{N}_i^d \cap \mathsf{N}_j^d$ are excluded, the effective probability of a node from each side (i.e., $\mathsf{N}_i^d - \mathsf{N}_j^d$ and $\mathsf{N}_j^d - \mathsf{N}_i^d$) being malicious changes; it increases if we assume that $\mathsf{N}_i^d \cap \mathsf{N}_j^d$ consists of as many benign neighbors as possible, which is the worst-case from the benign side's perspective.

Below we first find the expected number of *common nodes* in two neighborhoods.

Lemma 1. *The expected size of $\mathsf{N}_i^d \cap \mathsf{N}_j^d$ is*

$$\overline{|\mathsf{N}_i^d \cap \mathsf{N}_j^d|} = |\mathsf{N}_i^d||\mathsf{N}_j^d|/(|V| - 1). \tag{3}$$

Proof. Let $\lambda_x = |\mathsf{N}_x^d|$, i.e., neighborhood size, for simplicity of notation. We can view a node v_x having λ_x neighbor nodes as drawing λ_x nodes from $|V| - 1$ nodes. Hence, there are total $\binom{|V|-1}{\lambda_i}\binom{|V|-1}{\lambda_j}$ ways to have the two neighbor sets independently. Now, let Y be a random variable representing the number of common nodes in N_i^d and N_j^d. Suppose total $Y = y$ nodes are common between them. This can be viewed as (i) λ_j nodes are picked first for N_j^d from $|V| - 1$ nodes, (ii) among them y nodes are shared with N_i^d, and finally (iii) from the rest (i.e., $|V| - 1 - \lambda_j$), $\lambda_i - y$ nodes are drawn to fill N_i^d. Therefore, the probability that $|\mathsf{N}_i^d \cap \mathsf{N}_j^d| = y$ is

$$\Pr(Y = y) = \frac{\binom{|V|-1}{\lambda_j}\binom{\lambda_j}{y}\binom{|V|-1-\lambda_j}{\lambda_i-y}}{\binom{|V|-1}{\lambda_i}\binom{|V|-1}{\lambda_j}} = \frac{\binom{\lambda_j}{y}\binom{|V|-1-\lambda_j}{\lambda_i-y}}{\binom{|V|-1}{\lambda_i}},$$

which follows the hypergeometric distribution. Hence,

$$\overline{Y} = \mathbb{E}[Y] = \sum_{y=0}^{\lambda_i} y \cdot \Pr(Y = y) = \frac{\lambda_i \lambda_j}{|V| - 1}.$$

□

Fig. 10 shows the expected number of common nodes between two neighbor sets of the same size $\lambda = |\mathsf{N}_i^d| = |\mathsf{N}_j^d|$. Larger neighbor sets are likely to share more nodes. But it diminishes as the network size increases. When the network is sufficiently large, neighbor sets are likely to be disjoint.

Using the numbers in Figs. 9 and 10, one can estimate the number of common nodes between two neighborhoods given the network parameters. For example, for $(|V|, f, d) = (1000, 5, 2)$, the expected neighborhood size $\overline{|\mathsf{N}^d|}$ is about 30, in which case any two neighbor sets are expected to share about 0.9 nodes.

Lemma 2. *If the neighbor sets of v_i and v_j satisfy the following condition, there will be more benign nodes than malicious ones in a witness group selected uniform-randomly between the two:*

$$p_m < \frac{|\mathsf{N}_i^d| + |\mathsf{N}_j^d|}{2\left(\frac{|\mathsf{N}_i^d|^2}{|\mathsf{N}_i^d|-y} + \frac{|\mathsf{N}_j^d|^2}{|\mathsf{N}_j^d|-y}\right)}, \tag{4}$$

where y is the number of common nodes between them.

Proof. Let $\lambda_x = |\mathsf{N}_x^d|$, i.e., neighborhood size, for simplicity of notation. From the benign side's point of view, it is the worst-case when all the common nodes are benign. In this case, the

probability that a node in $\mathsf{N}_i^d - \mathsf{N}_j^d$ (resp. $\mathsf{N}_j^d - \mathsf{N}_i^d$) being malicious is increased to $\frac{\lambda_i}{\lambda_i - y} p_m$ (resp. $\frac{\lambda_j}{\lambda_j - y} p_m$). Hence, the number of malicious nodes (resp. benign nodes) selected from v_i's neighborhood will be $\alpha_i |W| \frac{\lambda_i p_m}{\lambda_i - y}$ (resp. $\alpha_i |W| \frac{\lambda_i (1-p_m) - y}{\lambda_i - y}$). Now, to have more benign nodes than malicious ones in a witness group between the two sides,

$$\underbrace{\sum_{x \in \{i,j\}} \alpha_x \frac{\lambda_x (1 - p_m) - y}{\lambda_x - y}}_{\text{benign}} > \underbrace{\sum_{x \in \{i,j\}} \alpha_x \frac{\lambda_x p_m}{\lambda_x - y}}_{\text{malicious}}$$

should hold. Because $\alpha_i = \frac{\lambda_i}{\lambda_i + \lambda_j}$ and $\alpha_j = \frac{\lambda_j}{\lambda_i + \lambda_j}$,

$$p_m < \frac{\lambda_i + \lambda_j}{2\left(\frac{\lambda_i^2}{\lambda_i - y} + \frac{\lambda_j^2}{\lambda_j - y}\right)}.$$

□

Using the results above, we can find proper values for the network parameters, i.e., peerset size f and depth limit d.

Theorem 1. *For an average network in which nodes' neighborhoods are of similar size* $\overline{|\mathsf{N}^d|}$,

$$p_m < \left(|V| - 1 - \overline{|\mathsf{N}^d|}\right) / \left(2(|V| - 1)\right) \tag{5}$$

should hold to have more benign nodes than malicious nodes in a witness group.

Proof. Due to Lemma 2, $p_m < \frac{\overline{|\mathsf{N}^d|} - y}{2\overline{|\mathsf{N}^d|}}$ holds. For an average network, $y = \overline{|\mathsf{N}_i^d \cap \mathsf{N}_j^d|} = \frac{\overline{|\mathsf{N}^d|}^2}{|V| - 1}$, due to Lemma 1. Thus

$$p_m < \frac{\overline{|\mathsf{N}^d|} - \overline{|\mathsf{N}_i^d \cap \mathsf{N}_j^d|}}{2\overline{|\mathsf{N}^d|}} = \frac{|V| - 1 - \overline{|\mathsf{N}^d|}}{2(|V| - 1)}.$$

□

Given an upper-bound on p_m, we can find the depth limit d and the maximum peerset size f that satisfy Eq. (5). From Theorem 1, we can see that Eq. (5) may not satisfy if neighborhood is too large compared to the network size:

Example 3. *Suppose* $|V| = 100$ *and* $p_m = 25\%$. *By Eq.* (5),

$$\overline{|\mathsf{N}^d|} < (|V| - 1)(1 - 2p_m) = 49.5$$

should hold. We can find (f, d) *that satisfy the above condition, from Fig. 9. For instance,* $\overline{|\mathsf{N}^d|} = 26.46$ *for* $(f, d) = (5, 2)$ *when* $|V| = 100$. *However,* $(f, d) = (5, 3)$ *cannot be used because* $\overline{|\mathsf{N}^d|} = 79.13$. *In this case, most of the nodes in both neighborhoods are common (thus are excluded from the witness selection), which leaves very few witness candidates (that are in* $\mathsf{N}_i^d \cup \mathsf{N}_j^d - \mathsf{N}_i^d \cap \mathsf{N}_j^d$*). Because we assume* $\mathsf{N}_i^d \cap \mathsf{N}_j^d$ *consists of as many benign neighbors as possible, the probability that the remaining candidates are malicious increases.*

For a very large network, $\lim_{|V| \to \infty} \overline{|\mathsf{N}_i^d \cap \mathsf{N}_j^d|} = 0$ due to Lemma 1, thus the condition in Eq. (5) becomes $p_m < 1/2$.

The second case is when v_j and its colluding nodes form a network of their own by *not* following the protocol because, as analyzed in Sec. IV-B, they gain no benefit from having peer relationship with benign nodes. This network would be formed only by v_j and its colluders, disconnected from the network of benign nodes (thus $\mathsf{N}_i^d \cap \mathsf{N}_j^d = \emptyset$). Because all the witness candidates from v_j's side will be malicious, $|\mathsf{N}_i^d| > |\mathsf{N}_j^d|$ should hold for the benign witnesses to be the majority of a witness group. Given an upper-bound on $|V_m| = |V| p_m$, i.e., the size of the collusion group, we can find the values for the network parameters, i.e., d and f, such that the expected neighborhood size $\overline{|\mathsf{N}^d|}$ is greater than $|V_m|$. For instance, for $|V| = 1000$ and $p_m = 10\%$, $(f, d) = (5, 2)$ will *not* satisfy this condition as neighborhoods would be too small: $\overline{|\mathsf{N}^d|} = 29.63$ while $|V_m| = 100$. The benign side's neighborhood would be large enough if $(f, d) = (10, 3)$ or $(5, 3)$.

VI. Evaluation

We implemented AccountNet as a Python library using `libsodium` [23] for digital signature and a fork [24] of it for an elliptic curve-based verifiable random function. Nodes use `XML-RPC` for the AccountNet-specific communication such as the peer shuffling and the witness group formation. We deployed the nodes to (up to) 80 virtual machines hosted on Amazon EC2 `t3a.2xlarge` instances (8 vcpus, 32 GiB memory). To impose realistic network latency among the nodes, we used `NetEM` tool to add network delay of 20 ms for both egress and loopback packets. Hence, even the nodes running on the same VM experience at least about 40 ms of round trip delay. Table I lists the network parameters. We analyzed the network every 10 seconds. When we use 'round' in what follows, it refers to this analysis round, not the shuffling round.

Nodes are distributed evenly among the VM instances – 125 nodes run on a single VM. In each VM, nodes are launched with a random interval (uniform from $[0, 10]$ seconds). The launch rate is the same for all VMs (but the network growth rates differ). Hence, as can be seen from the left plot in Figure 11, the network reaches the full size (i.e., specified $|V|$) at around similar analysis rounds (about the 70^{th}–75^{th} round).

When launching a node, it is randomly flagged as malicious, for analysis purposes, with a probability of p_m. If $|V| = 1000$

TABLE I
NETWORK CONFIGURATION PARAMETERS.

Parameter	Values
Network size, $\|V\|$	$500, 1000, 5000, 10000$
Maximum peerset size, f	$5, 7, 10$
Shuffling length, L	$\lceil f/2 \rceil$
Neighborhood depth limit, d	$2, 3$
Prob. of malicious node, p_m	$1\%, 10\%$
Shuffling period	10 seconds (on average)

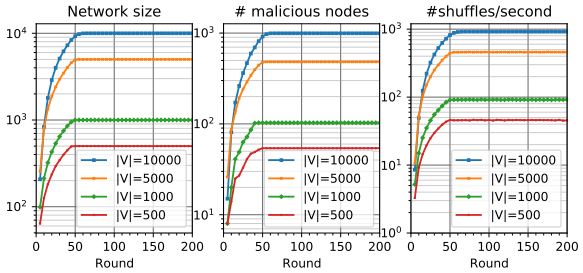


Fig. 11. Network size, number of malicious nodes (when $p_m = 0.1$), and shuffling rate. The y-axes are in log scale.

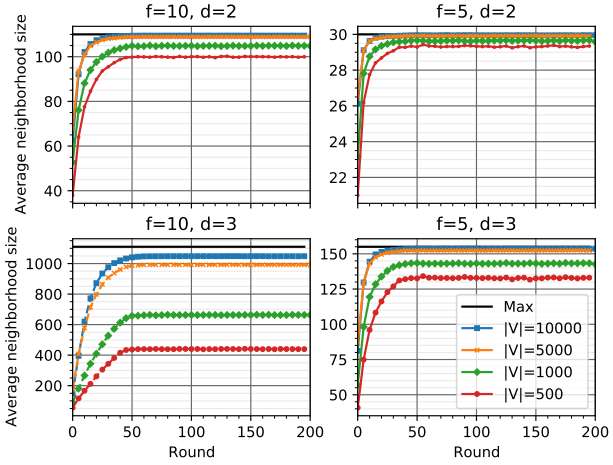


Fig. 12. Average neighborhood sizes for different network configurations.

and $p_m = 10\%$, the network ends up having about 100 malicious nodes as shown in the middle plot of Fig. 11. Note again that the malicious nodes could consist of multiple collusion groups. We view them as a single group for the worst-case security analysis.

Each node initiates a shuffling every about 10 seconds with a random variance. When $|V| = 10000$, approximately 1000 shuffles on average (i.e., shuffling rate = $0.1|V|$ shuffles/sec) are expected to occur every second. The right plot in Fig. 11 shows the shuffling rates for different network sizes. Note that the shuffling period was chosen rather arbitrarily. A shorter (resp. longer) period can be used, in which case the network is more (resp. less) reactive to changes in the network (e.g., new/leaving nodes).

A. Results

In what follows, we evaluate the impact of different network configurations (such as the neighborhood depth limit d and the peerset size f) on the behavior of AccountNet. Appendix A presents supplementary evaluation results that show the effectiveness of AccountNet's peer shuffling on peer discovery and network connectivity.

Average size of neighborhood and common nodes: Fig. 12 shows the average neighborhood sizes for different network configurations. As discussed in Sec. V, the average neighborhood size highly depends on the peerset size f and the depth limit d. We can also see that the experimental results match the analytically computed values, $\overline{|\mathrm{N}^d|}$, shown in Figure 9.

TABLE II
AVERAGE NEIGHBORHOOD SIZE.

	$f=10, d=3$		$f=5, d=2$	
$\|V\|$	Analysis	Measurement	Analysis	Measurement
500	446.25	439.19	29.26	29.35
1000	671.97	663.42	29.63	29.67
5000	996.29	991.79	29.93	29.91
10000	1051.10	1048.37	29.96	29.95

TABLE III
AVG. NUMBER OF COMMON NODES BETWEEN NEIGHBORHOODS.

	$f=10, d=3$		$f=5, d=2$	
$\|V\|$	Analysis	Measurement	Analysis	Measurement
500	387.98	388.27	1.80	1.85
1000	440.01	449.19	0.90	0.96
5000	196.85	206.00	0.18	0.19
10000	109.84	115.54	0.09	0.10

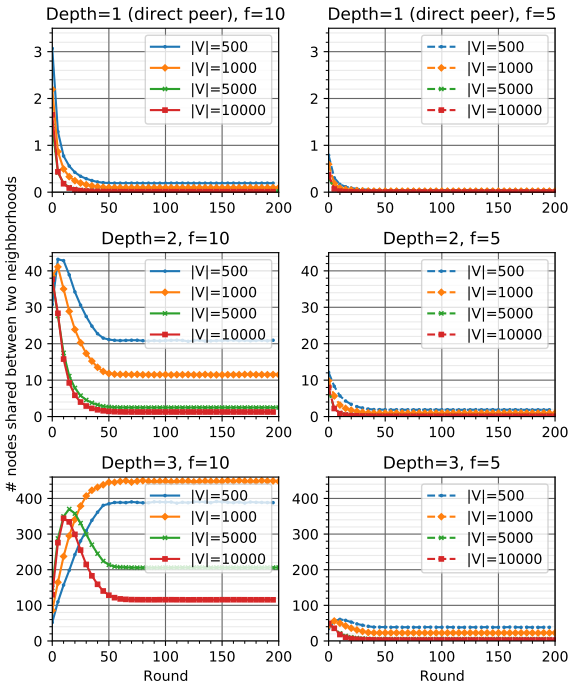


Fig. 13. Average number of common nodes in pairs of neighborhoods.

As explained in Sec. V, nodes shared between two neighbor sets are excluded when forming a witness group. Fig. 13 shows the number of nodes shared between the neighborhoods of every pair. We can see that two neighbor sets are likely to have more common nodes when the network is small. In addition, the number of common nodes increases with the neighborhood depth limit d simply because one's neighborhood size increases exponentially with d. For the same depth, nodes are likely to share more neighbors when the peerset size f is larger. We can also observe that during the initial rounds, nodes are likely to share more nodes in their neighborhoods because the peersets are not sufficiently shuffled. As the plots show, the number of common nodes drops quickly as the shuffling rounds proceed. However, the result for $d = 3$ and $f = 10$ (i.e., the bottom left plot), especially of $|V| = 500, 1000$, shows a different trend. These are when the neighborhoods are too large relative to the network size. Hence, the nodes in these cases are likely to have similar neighbor sets. For instance, for $|V| = 500$, the average neighborhood size for $d = 3, f = 10$ is about 440 as shown in Table II while the average number of common nodes in pairs of neighborhoods is close to 390 (shown in Table III). Because most of the nodes in the neighbor set are shared with other nodes, such a configuration leaves very few candidates for a witness, which may affect the witness selection process as will be shown shortly. Meanwhile, Table III shows that the analysis in Lemma 1 can accurately estimate the average number of common nodes between neighborhoods. This result, in conjunction with Tables II, is one of the indications that the network behaves uniform-randomly.

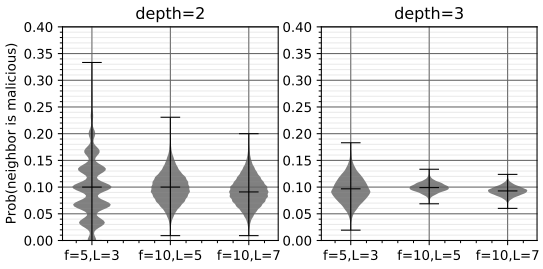


Fig. 14. Prob. of neighbor node being malicious.

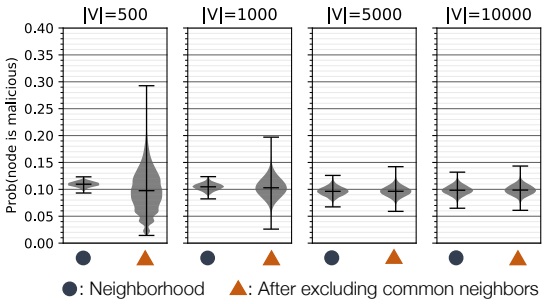


Fig. 15. Prob. of witness candidate being malicious ($f = 10$, $d = 3$).

Probability of neighbor being malicious: Fig. 14 compares the probability of a neighbor node being malicious. If the network behaves uniform-randomly, the average probability should be close to the specified p_m. The left plot compares the probability distributions for different (f, L) combinations for $|V| = 10000$, $p_m = 10\%$, and $d = 2$. We can see that as the peerset size f and/or the shuffle length L are smaller, the variance of the distribution is higher (i.e., some nodes have more/less malicious neighbor nodes than others) because peers are less-aggressively shuffled when f and/or L are smaller. When the neighborhood is expanded to $d = 3$ (the right plot), the distributions become narrower. These results suggest that large neighborhoods with aggressive shuffling can reduce the risk of having an above-average number of malicious neighbors.

Probability of witness candidate being malicious: Recall that common neighbor nodes are excluded when drawing witnesses. To see how this affects the distribution of malicious candidates, we analyzed the snapshots at the 200^{th} round by applying the witness group formation to every pair of nodes in the network. Fig. 15 shows that although the average probability remains the same, the variances become larger after excluding common neighbors. The case of $|V| = 500$ shows a potentially problematic situation: a large variance in the population of malicious candidates. This happens because nodes share too many neighbors – according to Tables II and III, the average neighborhood size is 440, but about 390 of them are shared with another neighborhood. This result suggests that neighborhoods should not be too large relative to the network size.

Effective shuffle history length: As explained in Sec. IV, nodes exchange their peerset update histories to prove the correctness of their current peersets. Because of shuffling, nodes need to exchange only a part of the history that is long enough for the proof. Hence, we measured the length of peerset update history that nodes exchange when shuffling peers. Fig. 16

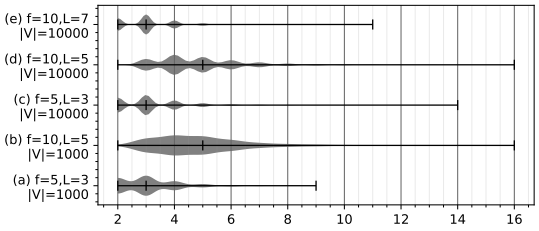


Fig. 16. Lengths of effective peerset update history.

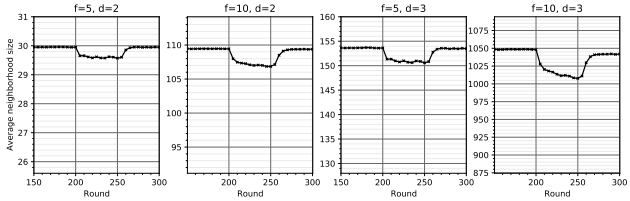


Fig. 17. Average neighborhood sizes when 1000 out of 10000 nodes leave.

compares the distribution obtained from snapshots when the networks are in the steady state. Comparing (a) and (b) (or (c) and (d)), we can first see that nodes need to send longer lists of peerset update records when the peerset size (f) is large. Now, comparing (d) and (e), a higher L (i.e., the shuffle length) leads to shorter history length because peersets change aggressively, and thus peers are unlikely to stay for long periods – e.g., when $f = 10$ and $L = 7$, the probability that a peer stays for 4 consecutive shuffling rounds is less than 1%. Overall, the effective history is kept short, hence nodes do not need to send a long list of entries.

Network churn: To see how AccountNet behaves when nodes leave the network, we performed a network churn. Specifically, from about the 200^{th} round, 10% of the network nodes (which are randomly selected) start leaving the network *ungracefully*. Fig. 17 shows that when nodes are actively leaving, the average neighborhood sizes dip below what can be analytically computed. For instance, for $f = 10$ and $d = 3$, it drops to as low as 1007.67 at the 250^{th} round. It is about 3.28% less than what it should be for the network size ($|V| = 9000$). Hence, when determining the network parameters, the average neighborhood size should be estimated with a margin to cope with network churn. Nevertheless, we can see that the network quickly self-organizes after the churn. Note that because the nodes left

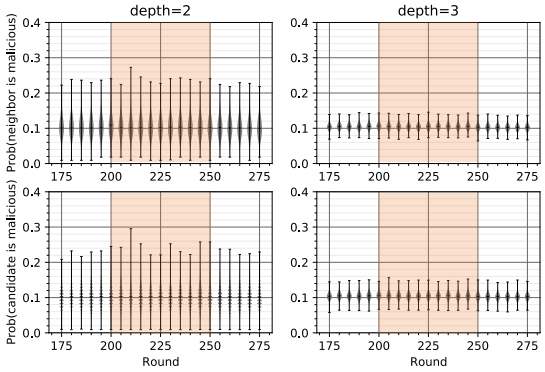


Fig. 18. Probability that a neighbor (top plots) or a remaining candidate (bottom plots) is malicious when 1000 out of 10000 nodes leave. $f = 10$.

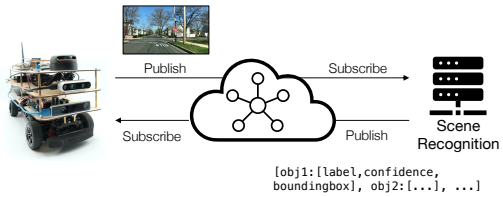


Fig. 19. Cloud-based ML service over AccountNet.

without notifying any others, one's leaving is discovered only when it is selected by another node as a shuffling counterpart. Discovery of inactive nodes can be made faster by an active ping [11] or using a shorter shuffling period. Meanwhile, Fig. 18 shows the probability distribution of (top) a neighbor node or (bottom) a witness candidate being malicious. The results show no statistically significant impact of network churn on them, and the trends are consistent with Figs. 14 and 15.

B. Use Case: Cloud-based ML Service

We implemented a broker-less publish-subscribe communication layer on top of AccountNet and applied it to a cloud-based ML inference service – a robotic ground vehicle outsources an object detection task to the cloud service as shown in Fig. 19. In this architecture, nodes publish data messages to *topics* (e.g., `scene_image`) that identify the content of data and subscribe to topics relevant to their tasks. The vehicle-side node publishes scene images, captured by a front-facing camera, to topic `scene_image`. The cloud-side node, which subscribes to the topic, utilizes an ML service [1] to detect objects in a given scene image. Then, it publishes the detection results (e.g., object labels, confidence levels, locations, etc.) to `detected_objects` topic. Note again that in AccountNet, nodes act as publisher, subscriber, witness, or all of them.

How should the network be configured to prevent a malicious node in this application from manipulating evidence about scene images or the results of object detection? Suppose the network consists of $|V| = 1000$ nodes and assume the online ML service is ill-intentioned and thus is in collusion with 99 other nodes (i.e., $p_m = 10\%$). As analyzed in Sec. IV-B, the collusion group has two choices: (i) following the AccountNet protocol correctly (but act maliciously when relaying data) or (ii) forming a separate network of their own. In the former case, the network should not have too big neighborhoods because otherwise, most of the benign witness candidates could be excluded. For the configuration given above, we can find, using Eq. (5) in Sec. V-B, that any (f, d) pairs that make the average neighborhood size not larger than 799.2 can be used. According to Fig. 12, $(f, d) = (5, 2), (5, 3), (10, 2), (10, 3)$, etc. satisfy the condition. Now, suppose instead that the malicious group did not want to follow the AccountNet protocol and thus has formed a separate overlay. In this case, all the witness candidates from the ML service-side will be malicious. Hence, the benign network should be able to make average-case neighborhoods large enough that they outnumber the malicious group. According to Fig. 12, the average neighborhood size would be slightly above 100 with $(f, d) = (10, 2)$. But as seen above it can shrink when there is network churn. Hence, considering a safe margin, $(f, d) = (10, 2)$ should not be used. However, with $(f, d) = (10, 3)$ or $(5, 3)$, benign nodes would likely have much larger neighborhoods than the collusion group would. Hence, more benign witnesses will be selected than malicious ones when the network is configured with $(f, d) = (10, 3)$ or $(5, 3)$.

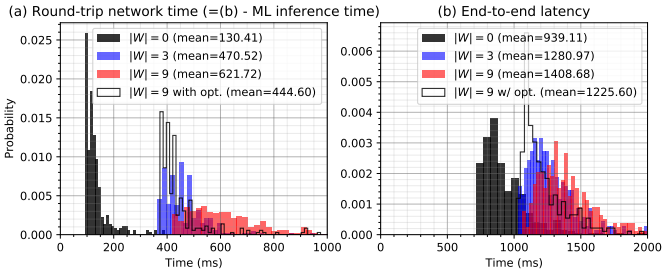


Fig. 20. Latency of the cloud-based object detection application using AccountNet. Notice the difference in the x-axis scales.

For an overhead evaluation purpose, we ran 1000 nodes on 8 × EC2 instances that form AccountNet. Then, we measured the end-to-end latency of the object detection application that runs on top of it. Fig. 20(b) compares the end-to-end latency for different sizes of witness group. It is measured from when the vehicle node publishes a scene image to when it receives a detection result. Hence it includes the ML service's inference time, which is about 809 ms on average but highly varying: $\sigma = 191$ ms, although we fixed the scene image to a single 2010 x 1125 resolution file to reduce the variance. Fig. 20(a) shows the round-trip time that does *not* include the ML inference time. The latency increases with AccountNet simply because of data relay through the witness nodes. However such an overhead can be masked to some extent, as can be seen from Fig. 20(b), by the ML inference service of which execution time is relatively long and highly varying. Meanwhile, in the original implementation, data is delivered up to the subscriber's application layer once the relayed data are received from *all* witnesses. However, one can improve the latency by moving up the data as soon as identical data have been received from more than $|W|/2$ witnesses. We implemented this optimization, which reduced the relay overhead considerably as shown in Fig. 20 ('with opt.').

VII. Related Work

Peer-to-peer (P2P) overlay allows systems across wide area networks to distribute data in a fault-tolerant, scalable, and self-organizing way [25]. P2P overlay can be categorized into two classes: structured and unstructured. In structured overlays [15]–[17], peer relationship is deterministic as it is derived from node identifiers such as IP address. On the other hand in unstructured overlays [19], [20], peer relationship is probabilistic. In gossip-based peer sampling [13], [14], [21], each node maintains a set of peers and keeps it updated by periodically exchanging (i.e., shuffling) the information with others in a random fashion.

Although structured overlays enable efficient data search and retrieval, they are vulnerable to *Eclipse* attacks; a group of malicious nodes tries to pollute a victim's routing table, which could lead to network partitioning and denial of service attacks. Castro *et al.* [3] defend against Eclipse attacks by entrusting

a set of trusted certification authorities to assign IDs to nodes. In [18], Singh *et al.* have nodes audit each other's peer set and refuse to have a peer relationship with a node whose degree is significantly above the average. Hildrum *et al.* [26] propose a proximity neighbor selection mechanism that considers network latency to prevent a small set of malicious nodes from becoming neighbors of a large number of benign nodes. Fireflies [11] provides each node with a *full* view of live nodes in the network using an accusation-rebuttal process.

In unstructured overlays, Eclipse attacks target to fool benign nodes into having malicious nodes as their peers, to isolate the victim from the rest of the network. PuppetCast [10] employs trusted, central bootstrap servers that hand out random samples of peers to new nodes by maintaining a complete membership of the network. Jesi *et al.* [27], sharing a similar idea as [18], consider 'hub' nodes, who have large in-degree values and thus are over-represented in the views of other nodes, thus potentially malicious. In BAR gossip [8], a node uses a pseudo-random number generation to select a partner from its peers for exchanging data (e.g., video stream packets). The partner can verify if the random partner selection was unbiased. But this requires a strong assumption that the full membership view is known to every node and that no node can join or leave.

The P2P nature of blockchains makes them vulnerable to Eclipse attacks. Heilman *et al.* [4] were able to pollute a Bitcoin node's address tables by repeatedly sending unsolicited incoming connections. They proposed countermeasures that include random eviction and selection of peer connections, increasing the peerset size, detecting peers with abnormal connection size, etc. Marcus *et al.* [28] showed Eclipse attacks on Ethereum nodes running on a structured overlay, Kademlia [29], and proposed similar countermeasures. Nevertheless, even with the countermeasures, Eclipse attacks are still possible when a massive number of Sybil identities can be created [30]. As can be seen, most of these works either (a) entrust trusted authorities to manage full node membership or (b) use heuristics (e.g., node degree) to detect potentially malicious peers. AccountNet does require neither any central entity nor full node membership.

Accountability has been studied mostly in the context of detecting *faulty* nodes in distributed applications. In PeerReview [31], each node creates logs of messages that it sent/received, which are used by 'witnesses' to determine whether the node deviated from the correct behavior. They are selected from other nodes in a structured manner [32]. However, they assume that at least one witness in a group is correct. FullReview [33] considers selfish witnesses who are tempted not to act as monitors. They use a game theoretic approach that incentivizes selfish witnesses to follow the monitoring protocol. However, both PeerReview and FullReview require an assumption that a reference implementation for the node being monitored exists so that monitor nodes can replay and check the logs.

VIII. Conclusion

AccountNet is a peer-to-peer overlay in which the participants act as witnesses to data propagation between others. With AccountNet's verifiable peer shuffling, a malicious group of nodes has to either follow the protocol faithfully or form a separate network of their own. In either case, the uniform-randomness of the benign nodes' peer shuffling guarantees, probabilistically, that witness groups contain more honest witnesses than malicious ones, which we showed both analytically and experimentally. Systems that want to create provable evidence of their correct behavior can benefit from AccountNet by following its protocol faithfully.

Acknowledgment

This work is supported in part by NSF grant 2302610. Any opinions, findings, and conclusions or recommendations expressed here are those of the authors and do not necessarily reflect the views of sponsors.

References

[1] "Amazon rekognition," https://aws.amazon.com/rekognition/.
[2] "Azure Cognitive Services," https://azure.microsoft.com/en-us/services/cognitive-services/.
[3] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. S. Wallach, "Secure routing for structured peer-to-peer overlay networks," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 299–314, 2002.
[4] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network," in *Proc. of the 24th USENIX Conference on Security Symposium*, 2015, p. 129–144.
[5] J. Kuffner, "Cloud-enabled humanoid robots," in *Proc. of the 10th IEEE-RAS International Conference on Humanoid Robots*, 2010.
[6] B. Kehoe, S. Patil, P. Abbeel, and K. Goldberg, "A survey of research on cloud robotics and automation," *IEEE Transactions on automation science and engineering*, vol. 12, no. 2, pp. 398–409, 2015.
[7] J. R. Douceur, "The sybil attack," in *Proc. of the 1st International Workshop on Peer-to-Peer Systems*, 2002, p. 251–260.
[8] H. C. Li, A. Clement, E. L. Wong, J. Napper, I. Roy, L. Alvisi, and M. Dahlin, "Bar gossip," in *Proc. of the 7th symposium on Operating systems design and implementation*, 2006, pp. 191–204.
[9] E. Bortnikov, M. Gurevich, I. Keidar, G. Kliot, and A. Shraer, "Brahms: Byzantine resilient random membership sampling," in *Proc. of the 27th ACM Symposium on Principles of Distributed Computing*, 2008.
[10] A. Bakker and M. van Steen, "Puppetcast: A secure peer sampling protocol," in *European Conference on Computer Network Defense*, 2008.
[11] H. Johansen, A. Allavena, and R. van Renesse, "Fireflies: Scalable support for intrusion-tolerant network overlays," *SIGOPS Oper. Syst. Rev.*, vol. 40, no. 4, p. 3–13, Apr. 2006.
[12] N. Borisov, "Computational puzzles as sybil defenses," in *Proc. of the 6th IEEE International Conference on Peer-to-Peer Computing*, 2006.
[13] A. Allavena, A. Demers, and J. E. Hopcroft, "Correctness of a gossip based membership protocol," in *Proc. of the 24th ACM Symposium on Principles of Distributed Computing*, 2005, p. 292–301.
[14] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen, "Gossip-based peer sampling," *ACM Trans. Comput. Syst.*, vol. 25, no. 3, p. 8–es, Aug. 2007.
[15] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems," in *Proc. of the IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*, 2001, pp. 329–350.
[16] I. Stoica, R. Morris, D. Liben-Nowell, D. R. Karger, M. F. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for internet applications," vol. 11, no. 1, 2003, p. 17–32.
[17] B. Zhao, L. Huang, J. Stribling, S. Rhea, A. Joseph, and J. Kubiatowicz, "Tapestry: a resilient global-scale overlay for service deployment," *IEEE Journal on Selected Areas in Communications*, no. 1, pp. 41–53, 2004.
[18] A. Singh, T. wan "Johnny" Ngan, P. Druschel, and D. S. Wallach, "Eclipse attacks on overlay networks: Threats and defenses," in *Proc. of the 25th IEEE International Conference on Computer Communications*, 2006.
[19] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, "Freenet: A distributed anonymous information storage and retrieval system," in *Proc. of the International Workshop on Designing Privacy Enhancing Technologies: Design Issues in Anonymity and Unobservability*, 2001.
[20] M. Ripeanu, "Peer-to-peer architecture case study: Gnutella network," in *Proc. of the 1st International Conference on Peer-to-Peer Computing*, 2001.
[21] S. Voulgaris, D. Gavidia, and M. V. Steen, "Cyclon: Inexpensive membership management for unstructured p2p overlays," *Journal of Network and Systems Management*, vol. 13, pp. 197–217, 2005.

[22] S. Micali, M. Rabin, and S. Vadhan, "Verifiable random functions," in *Proc. of the 40th Annual Symposium on Foundations of Computer Science*, 1999.
[23] https://doc.libsodium.org/.
[24] https://github.com/algorand/libsodium.
[25] E. K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim, "A survey and comparison of peer-to-peer overlay network schemes," *IEEE Communications Surveys Tutorials*, vol. 7, no. 2, pp. 72–93, 2005.
[26] K. Hildrum and J. Kubiatowicz, "Asymptotically efficient approaches to fault-tolerance in peer-to-peer networks," in *Distributed Computing*, F. E. Fich, Ed. Springer Berlin Heidelberg, 2003, pp. 321–336.
[27] G. P. Jesi, A. Montresor, and M. van Steen, "Secure peer sampling," *Computer Networks*, vol. 54, no. 12, pp. 2086–2098, 2010.
[28] Y. Marcus, E. Heilman, and S. Goldberg, "Low-resource eclipse attacks on ethereum's peer-to-peer network," *IACR Cryptol. ePrint Arch.*, 2018.
[29] P. Maymounkov and D. Mazières, "Kademlia: A peer-to-peer information system based on the xor metric," in *Peer-to-Peer Systems*. Springer Berlin Heidelberg, 2002, pp. 53–65.
[30] M. Tran, A. Shenoi, and M. S. Kang, "On the routing-aware peering against network-eclipse attacks in bitcoin," in *Proc. of the 30th USENIX Security Symposium*, 2021, pp. 1253–1270.
[31] A. Haeberlen, P. Kouznetsov, and P. Druschel, "Peerreview: Practical accountability for distributed systems," in *Proc. of the 21st ACM SIGOPS Symposium on Operating Systems Principles*, 2007.
[32] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, and D. Lewin, "Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the world wide web," in *Proc. of the 29th Annual ACM Symposium on Theory of Computing*, 1997.
[33] A. Diarra, S. B. Mokhtar, P.-L. Aublin, and V. Quéma, "Fullreview: Practical accountability in presence of selfish nodes," in *Proc. of the 33rd IEEE International Symposium on Reliable Distributed Systems*, 2014.
[34] D. J. Watts and S. H. Strogatz, "Collective dynamics of 'small-world'networks," *nature*, vol. 393, no. 6684, pp. 440–442, 1998.

Appendix A
Supplementary Evaluation Results

Peer coverage: Nodes discover new peers as they shuffle their peersets with others. We analyzed how many distinct nodes that each node in the network ends up seeing as peers (at least once). Let us call this metric 'peer coverage.' Fig. 21 shows the distribution of the coverages over time for different network configurations. As shown in the figure, nodes discover new peers faster when having a larger peerset (i.e., higher f). For a small network ($|V| = 500$), most of the nodes in the network quickly end up seeing most of the other nodes. For a larger network ($|V| = 10000$), which is shown in the middle plot, it takes longer for the network to reach the full coverage. However, this does not indicate that nodes discover fewer unseen peers. The bottom plot in Fig. 21 compares the average coverage for different network sizes. The coverage-growth rate is in fact higher for a larger-size network because a shuffling is likely to discover more unseen peers when compared to a smaller network. Meanwhile, the shuffle length (L) also affects the coverage. As shown in the middle plot, the coverage grows faster if nodes shuffle peers aggressively (i.e., higher L) when the peerset size is same.

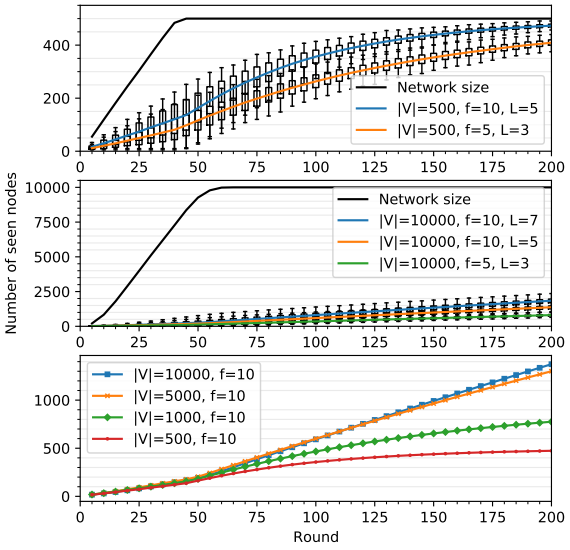


Fig. 21. Number of nodes seen as peers.

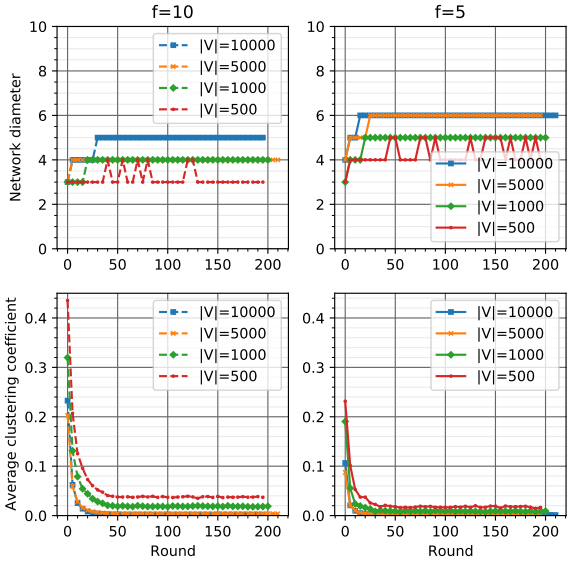


Fig. 22. Network diameter and clustering coefficient.

Network structure: One set of metrics that can represent the effectiveness of peer shuffling is the network *diameter* and the *clustering coefficient*. The diameter of a network is the maximum distance from a node to all others. The clustering coefficient [34] represents the tendency of a node's peers being a clique – the value of 1 indicates that every peer of a node is also connected to each other and that of 0 means that the peers are not connected to each other. As the network size increases, the diameter is likely to increase while the average clustering coefficient is likely to decrease. The latter could be high if peer selection is biased. Hence, a well-shuffled network is desired to keep these two values small. Fig. 22 shows how the network configurations affect these metrics. As can be seen from the top plots, the diameter is kept small even for larger networks. When the network size is same, a smaller f (i.e., peerset size) leads to larger diameter. For a small network (e.g., $|V| = 500$), the neighborhood of any node would be as large as the entire network even with a relatively small depth limit (e.g., $d = 3$). The bottom plots compare the average clustering coefficients for different configurations. As the peerset size f increases and/or the network size $|V|$ decreases, the coefficient naturally increases because the chance that two peers of a node could be in a peer relation also increases. Nevertheless, the average clustering coefficients converge quickly as the nodes join and perform peerset shufflings and are kept small.