# Web3 Sybil avoidance using network latency

Stokkink, Quinten ; Ileri, Can Umut; Epema, Dick; Pouwelse, Johan

**DOI**
10.1016/j.comnet.2023.109701

**Publication date**
2023

**Document Version**
Final published version

**Published in**
Computer Networks

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

# Web3 Sybil avoidance using network latency

Quinten Stokkink [*], Can Umut Ileri, Dick Epema, Johan Pouwelse

*Delft University of Technology (TU Delft), Mekelweg 5, 2628 CD Delft, The Netherlands*

A R T I C L E  I N F O

Dataset link: https://doi.org/10.4121/uuid:34850d65-1908-4249-b446-8e87c6d21ba0

*Keywords:*
Sybil attack
Latency
Network
Round-trip time
Identity

A B S T R A C T

Web3 is emerging as the new Internet-interaction model that facilitates direct collaboration between strangers without a need for prior trust between network participants and without central authorities. However, one of its shortcomings is the lack of a defense mechanism against the ability of a single user to generate a surplus of identities, known as the Sybil attack. Web3 has a Sybil attack problem because it uses peer sampling to establish connections between users. We evaluate the promising but under-explored direction of Sybil avoidance using network latency measurements, according to which two identities with equal latencies are suspected to be operated from the same node, and thus are likely Sybils. Network latency measurements have two desirable properties: they are only malleable by attackers by adding latency, and they do not require any trust between network participants. Our basic SybilSys mechanism avoids Sybil attackers using only network latency measurements if attackers do not actively exploit their malleability. We present an enhanced version of SybilSys that protects against targeted attacks using a variant of the flow correlation attack, which we name TrafficJamTrigger. We show how the message flows of Round-Trip Time measurements can be used to expose attack patterns and we propose and evaluate six classifiers to recognize these patterns. Our experiments show, through both emulation and real-world deployment, that enhanced SybilSys can serve a fundamental role for Web3, effectively establishing connections to real users even in the face of networks consisting of 99% Sybils.

## 1. Introduction

The Web3 ecosystem, the decentralized web, is the most recent step in 50 years of continuous evolution of the Internet. This evolution is tightly connected to the history of the birth, adoption, and governance of *distributed protocols* [1]. Web3 is the emerging decentralized alternative to the currently dominant Web2, which is centrally governed [2]. In contrast to the governance of Web2, the governance model of Web3 could be described as a “collaboration of strangers”, characterized by a lack of prior trust between users. Therefore, Web3 could be best defined as a *collectively maintained public infrastructure* [2]. To create this infrastructure, users interact with other users, which they find through peer sampling. However, attackers can interfere with this sampling by using Sybils, a surplus of fake identities [3]. We present a peer sampling mechanism that avoids connections to Sybils using only network latency.

A primary driver of Web3 is the use of transparency to become “trustless”, the absence of a need for trust in a system [4]. To this end, Web3 typically consists of peer-to-peer technology (e.g., BitTorrent) with a distributed ledger (e.g., Bitcoin) and the social practice of sharing (e.g., GitHub). Web3 is based on open protocols, open source, open collaboration, and freely re-usable components. Web3 can provide critical infrastructure for identity, money, markets, data, and AI. Even security improvements are handled openly: Web3 is aiming to alter the incentives for exploiting bugs, thereby improving security. Incentive alignment is achieved by offering payment (e.g., Bitcoins) through bug bounty programs [5]. For example, the community has tried to address frontrunning, a method to exploit pending trading actions [6], collaboratively [7–9].

Scalability to billions of users is a problematic requirement for Web3. Scaling issues arise as nodes within large-scale distributed systems are unable to keep track of all other participating nodes. The membership table of nodes may become unmanageable due to its size and due to the need for frequent updates because of node arrivals and departures. Early work used this approach, leading to considerable synchronization costs at scale [10]. Instead of depending on a single global (data) structure, full scalability is only achieved with distributed infrastructure. However, the typical Web2 approach of using predefined critical infrastructure, like the cloud or privately owned servers, violates the trustless nature of Web3. Therefore, Web3 applications employ different means to tackle their scalability issue.

A fundamental mechanism at the heart of Web3 applications, which addresses the scalability issue of node discovery, is the *peer-sampling*

[*] Corresponding author.
*E-mail addresses:* q.a.stokkink@tudelft.nl (Q. Stokkink), c.u.ileri@tudelft.nl (C.U. Ileri), d.h.j.epema@tudelft.nl (D. Epema), j.a.pouwelse@tudelft.nl (J. Pouwelse).

https://doi.org/10.1016/j.comnet.2023.109701
Received 4 May 2022; Received in revised form 9 March 2023; Accepted 11 March 2023
Available online 16 March 2023

*service* [11] as it appears for example in Bitcoin and BitTorrent. To this end, Web3 requires the creation of an overlay network from network participants' computers without relying on centrally governed servers ("zero-server"), and resilience against fraud in the form of fake identities. A peer sampling service is required by all three known styles of gossip protocols used by Web3 applications, i.e., rumor-mongering protocols, anti-entropy protocols, and aggregation protocols [12]. Rumor-mongering protocols use flooding to spread information at a fixed rate to all nodes. Anti-entropy protocols connect to nodes randomly, comparing information and reconciling differences. Aggregation protocols conduct pair-wise information exchanges and combine values to arrive at a system-wide value. Therefore, all gossip-based Web3 designs are susceptible to attacks on the peer sampling mechanism, e.g., a Sybil attack.

Any node can use a peer sampling service to discover other nodes that are randomly sampled from all nodes in the system, these nodes are the node's initial *neighbors*. Further connections can be made to even more peers through connections to the neighbors of a node's neighbors. Thereby, the concept of neighbors exploits locality as an alternative to a centrally maintained membership table. However, if a bad-faith actor creates many fake identities (Sybils) to serve as neighbors, real identities may perpetually be introduced to more Sybils. This systemic oppression of nodes hinders their ability to maintain public infrastructure and, thereby, allows for their abuse. For example, an attacker can deprive nodes of the latest information in a cryptocurrency, which leads them to think that they will receive currency though this will not happen (also known as "double-spending" [13]).

Using network latency to avoid Sybils is a unique opportunity for Web3. In Web2 systems, centralized platforms are used to bring users in contact with each other, and the indirection of traffic that goes with it makes it difficult, if not impossible, to measure network latency between users. In contrast, in the Web3 ecosystem, nodes send messages to each other over the Internet without intermediaries, enabling a new method of defense through latency. Furthermore, these latency measurements are a particularly attractive option as they do not require a gossip protocol, avoiding a chicken and egg situation.

This work proposes a *Sybil-avoiding peer sampling* service to enable the collective infrastructure of Web3 that provides nodes with non-Sybil nodes without a-priori trust. To achieve this, we counter the Sybil attack solely using measured network distances, which does not require a centrally governed entity. The core contribution of our work is **a mechanism for dependable, zero-server, trustless network topology creation in spite of Sybils** called *SybilSys*. On a more granular level our contributions are as follows:

1. We argue that Sybil avoidance must be addressed in the peer sampling mechanism (Section 2) and we define an adversary model (Section 3).
2. We rationalize, implement and verify light-weight Sybil-avoiding peer sampling using only network latency measurements (Section 4).
3. We show how message flows of latency measurements can be used to detect Sybils over the Internet (Section 5).
4. We create a version of our peer sampling mechanism that specifically counters attacks against our initial mechanism (Section 6), based on message flows, and we verify it with real users (Section 7). We derive that this mechanism is trustless, predominantly samples honest nodes and cannot be efficiently attacked.

## 2. Problem description

The core problem of our work is breaking the dependency between requiring Sybil-free gossip protocols and using gossiped information to eliminate Sybils. Gossip protocols do not guarantee delivery of information in a network filled with Sybils and Sybils cannot be identified without the information gossiped through these protocols. In avoiding gossip protocols, Sybil avoidance becomes a key problem for the peer sampling service. A straightforward gossip-free approach, like relying on unique IP-addresses or `traceroute` to filter Sybils is still open to attacks, as recently shown through the Erebus attack [14].

**Adversarial behavior is often considered to be out of scope** for peer sampling research (e.g., with assumptions that "each participant is limited to one identity" [15]). For decades, analytical studies proposed peer sampling services and tried to determine if they actually lead to uniform sampling or become unstable—while ignoring security [11,15–17].

Bitcoin is an example of a system that seemingly resists Sybils, but can still suffer attacks on the network layer [18]. Only on its application layer does Bitcoin provide a solution to Sybil attacks by crafting a high computational-cost barrier for tampering through a mathematical puzzle known as proof-of-work. Regrettably, alternatives to proof-of-work—or in general proof-of-something—systems with both lower computational costs and preservation of decentralization remain elusive [19]. Nevertheless, attacks on Bitcoin using Sybils are effective. Using Sybils to cause (even temporary) network partitioning is monetarily profitable for attackers [20] and leads to long-term advantages [21]. An interesting ongoing experiment is Blockstack, a chain which externalized protection against Sybils by relying on Bitcoin [22]. Blockstack rewards prior mining in Bitcoin and uses a verifiable random function for leader election, presumably leading to undesirable rich-get-richer dynamics.

**Peer sampling is inherently fragile and sensitive** to fraud and faults. Protocols with mass adoption typically encounter problems such as malicious behavior, accidental deviations from protocol specifications, malfunctioning nodes, correlated failures, network partitioning, data corruption, and dissemination of incorrect information. The state-of-the-art remains fragile. Especially Sybils that follow specialized attack strategies (beyond simply existing) go undetected in social networks [23–25] and are able to influence applications based on (federated) Artificial Intelligence [26–28].

An example of using a "score function" on social interactions, to protect against Sybils, is found in GossipSub [29]. With various parameters, each node attempts to keep track of the honesty of its neighbors. Their "application-specific score" imports information from the application level for Sybil avoidance and their "IP Address Collocation Factor" parameter is very effective against simple Sybil attacks. The global impact of this approach is still limited because it is not possible to trust your neighbors to accurately report their neighbors' Internet addresses (i.e., this violates our trustless requirement). Every address obtained by an attacker can be re-used to create unique Sybils in GossipSub. Services that offer unique IP addresses, *rotating proxy infrastructure* rental, can dramatically decrease the monetary cost of an attack on this system.

**Using network latency is a promising approach** to the Sybil problem, due to the grounding in the laws of physics. Network latency can only be tweaked by attackers by artificially increasing it, *never* by decreasing it [30]. No software-based attack can alter the lower bound on latency between two nodes, as the underlying shortest physical link between nodes remains the same.

Network latency for Sybil avoidance is used in related fields outside of epidemic protocols and peer sampling. There have been prior attempts to make use of network latency for the detection and avoidance of Sybils [30,31]. Early work uses triangulation to pinpoint physical coordinates within wireless sensor networks. Unfortunately, triangulation has proven to fail with realistic network conditions [32–34] due to lack of latency symmetry for bidirectional wireless links and invalidity of the *triangular inequality* from Euclidean geometry. Moreover, triangulation requires trust in the localization data provided by strangers, violating the trustless requirement.

## 3. Adversary model

Web3 nodes aim to cooperatively maintain public infrastructure in the presence of attackers that employ an overpowering number of identities on a limited set of physical resources. Violation of public infrastructure by attackers can be detected by sharing immutable Web3 data. For example, the immutable history of Bitcoin exposes any attempted violations by attackers through its chain of blocks (connected using cryptographic hashes). Therefore, a connection to only a single other honest node is required for any honest node to detect violations. Conversely, the goal of the attackers is to disallow this sharing of data, to not have their violations detected. To distinguish attackers, two types of entities are considered:

- *honest nodes* control only one logical node and one identity per physical machine in the network.
- *attackers* create and control multiple identities per physical machine in the network. Any identity controlled by an attacker entity is a *Sybil identity*.

The focus of this work is to counter the cheapest attack on peer sampling services: the generation of Sybil identities on a single node. Within this scope, the goal of attackers is centered around operating a surplus of Sybil identities on a single node in an attempt to isolate a cluster of honest nodes from the rest of the network (also known as an Eclipse attack [35]). Therefore, we consider a Sybil-avoiding peer sampling mechanism to be successful if it associates only a single identity with a single node. Our goal does not include the more complex case of countering Sybil identities operated by a single attacker from multiple nodes, e.g., by using a botnet, as existing work has already been shown to work when attackers are limited to one identity per IP address [15,36].

The peer sampling mechanism that we present in this work depends on latency measurements between nodes and we now discuss how it can be attacked by adversaries. In particular, in the remainder of this section we discuss how the possible attacks are addressed in our work. Our analysis considers attacks on the hardware, software, and sessions, required to perform a latency measurement between nodes over the Internet.

Attacks on the ability of distributed communications networks, like the Internet, to pass messages between nodes have been studied for over 60 years [37]. However, to this day, messages that are sent over the Internet are still not guaranteed to arrive at their destination. We mitigate all cases of latency measurement messages not arriving (e.g., due to attacks, offline nodes, and network disruption or outage) by periodically contacting neighbors and disconnecting from them if they do not respond. Of course, a network that lacks even the basic means of communication between nodes trivially forgoes the need of a Sybil defense, like the one we propose. Therefore, we make the following assumptions of basic meaningful connectivity:

- **Assumption 1:** Nodes are addressable over the Internet (e.g., through IP) and connectable based on their address.
- **Assumption 2:** Honest nodes are online long enough to engage in the peer sampling process.
- **Assumption 3:** Attackers do not control all hardware on the routing path over the Internet between them and an honest node. In Section 5.1 we discuss attackers that control part of the hardware along the routing path.

It may be possible for adversaries to replay messages that are sent between other nodes. Our approach to counter replay attacks is to make use of unique pairs of a request and response message (explained in Section 4). These pairs are made unique through the use of a random nonce that is added to the request and response message. However, if the random nonce can be predicted, an attacker can send a latency response before any request was made and potentially lower its measured latency. Therefore, we assume that the random nonce generation used provides the following guarantee:

- **Assumption 4:** The request and response messages that are sent between nodes contain a unique nonce that cannot be predicted by an attacker.

The manner in which our basic mechanism establishes node identities is through their latency. However, honest nodes that share the same latency are all seen as belonging to the same identity and, thereby, as attackers. In this case, a single honest node would still be sampled but all others in the same location would be ignored, which is inefficient. In Section 5, we further enhance our identity establishment through estimation of Internet routing paths between nodes and how they overlap. However, it is possible that the routing paths of a node with two others do not overlap at all, e.g., using wormhole routing. In this second case, our enhanced mechanism may erroneously sample an attacker. Both of these cases can be addressed by assuming that honest nodes have typical consumer connections:

- **Assumption 5:** Honest nodes are geographically dispersed.
- **Assumption 6:** Honest nodes use a single network adapter and a single last-mile connection to their physical node.

Adversaries may attempt to abuse the connections that are opened between nodes by sending a large amount of data. In this work, we do not provide or implement any protection against these Denial-of-Service attacks. However, we note that our peer sampling mechanism has highly regular communication patterns, which allows it to be coupled to most of the many available Denial-of-Service protection mechanisms [38].

## 4. Basic SybilSys: basal Sybil-avoiding peer sampling

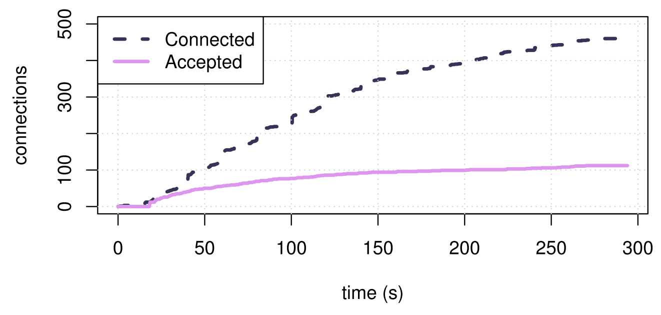
We first describe a basic mechanism called Basic SybilSys, which leverages round-trip time (RTT) measurements to avoid connecting to multiple identities from the same physical location, i.e., Sybils. This mechanism does not address the problem that comes with using network latency for Sybil avoidance: we assume attackers do not delay messages in order to try to defeat Basic SybilSys (altering latency measurements, see Section 2). Enhanced SybilSys drops this assumption (Section 5).

In every node, Basic SybilSys operates three lists of nodes: *discovered*, *connected*, and *accepted*. Node identities are moved between these lists in three steps. During each of these steps a node identity may be ignored or a connection to it may be dropped. When an identity ends up in the accepted list, it is made available by SybilSys to a given application.

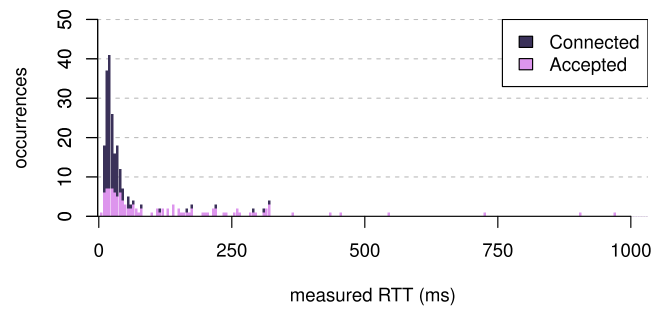
In the first step, Basic SybilSys is executed by a node to request new nodes from its initial accepted neighbors. These initial neighbors are typically either defined by a user or sampled from a specialized server, known as a rendezvous server [39]. Basic SybilSys can be implemented using a single request message and a single response message. The request message indicates a node wants to learn about new nodes and the response contains the Internet addresses of one or more nodes. New addresses in the responses are added to the *discovered node* list.

In the second step, the latency toward newly discovered nodes is measured. Repeatedly (using a fixed-interval schedule), a single identity is randomly sampled from the discovered node list and an outgoing connection attempt is made. During the connection establishment we measure the response time. We use the time between the transmission of the request and the response as the RTT, which includes the remote message processing time in a measurement. A node's identity moves into the *connected* node list once a connection is established (to the physical node over the Internet) and its RTT is known.

Finally, the essential step of Sybil filtering is carried out. The measured latencies are compared to those of all nodes on the accepted list. Non-Sybil identities are promoted from the connected node list to the *accepted* node list, while suspected Sybil connections are simply dropped. Basic SybilSys reasons that two identities with the same RTT

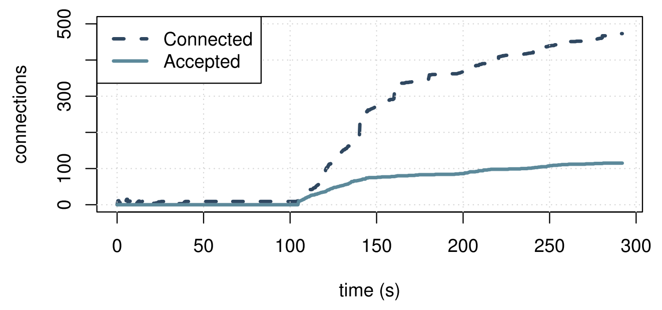


(a)

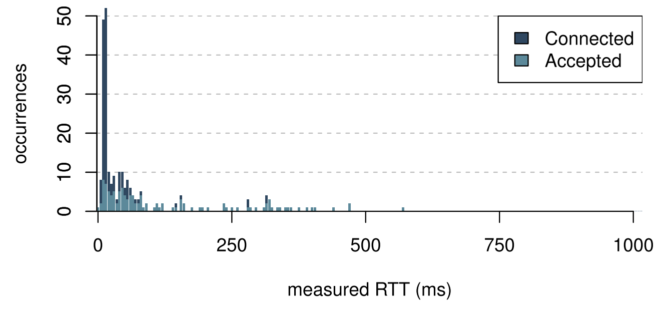


(b)

**Fig. 1.** The number of connected and accepted nodes without Sybils (a) and a histogram of their latencies at the time of 300 s (b).



(a)



(b)

**Fig. 2.** The number of connected and accepted nodes with 99% Sybils (a) and a histogram of their latencies at the time of 300 s (b).

reside on the same node and are therefore Sybils. We only accept a new node if we do not yet have a node on the accepted list with the same RTT. This filtering ensures the *latency diversity* property: all nodes using Basic SybilSys are surrounded by nodes with unique latencies.

The performance of Basic SybilSys relies on the filtering step, which ensures that no two (or more) nodes are accepted with a similar latency. This dramatically limits Sybil attacks to one identity per node (which is the same number as an honest node has). However, SybilSys may also filter identities that are not Sybils if they are operated from different nodes that happen to have the same latency.

## *4.1. Latency and diversity*

Our algorithm uses a simple, practical, and economical method to measure latency and (latency) diversity. We use a simple probe packet and response to measure Internet latency, an application-level ping approach. Repeated probing of a single target increases the measurement accuracy [30–34].

The latency measurements of two target nodes are considered (latency) *diverse* if their difference is larger than some threshold parameter $\Delta$. Our implementation uses the median of five latency probes as the latency measurement to avoid transient congestion and packet loss [40]. If more probes have been performed, the last five are used to form the latency measurement value.

Other, more advanced, implementations of latency diversity are also possible. For example, the Kolmogorov–Smirnov test can be used to test latency similarity [41], giving the distance between two series of latency measurements. It would also be possible to modulate $\Delta$ in our approach by deriving the variance of measurements from already-accepted nodes.

## *4.2. Real-world deployment*

We now deploy Basic SybilSys over the Internet for a Web3 application to demonstrate to what extent our peer sampling service avoids Sybils. The distributed system we utilize is called Tribler [42], which is our research vehicle for cooperative systems research, and which has been installed by 2 million users.[1] Tribler uses peer sampling and gossip protocols to offer a video streaming service.

Our setup consists of a single “measuring” node that uses Basic SybilSys to sample nodes from the Tribler network, which still use the default peer sampling service of Tribler. We evaluate one configuration with and one without Sybils, and we extend our setup with a second “attacker” node that operates Sybils in the former configuration. For the configuration without Sybils we do not include an “attacker” node in our setup. The experiment terminates after 300 s.

When used for the configuration with Sybils, the “attacker” node operates 100 Sybils in such a way that they do not hinder real users in the Tribler network. To obtain a presence of 99% Sybils, we modify the mechanism that provides an initial sample of nodes to be non-random. With a probability of 99% each identity in this sample is a Sybil, operated from our “attacker” node, instead of being randomly sampled from the Tribler network. Each Sybil only introduces one of the other Sybils from the “attacker” node for Basic SybilSys’ discovery step response messages (instead of a random sample of its accepted nodes). Lastly, we modify our response messages to not forward Sybils to the regular users from the Tribler network.

We present (a) the numbers of connected and accepted nodes over time, for which we (b) verify the diversity property (at a time of 300 s into the experiment). These results are presented for our configuration without Sybils in Fig. 1 and for a configuration with a Sybil presence of 99% in Fig. 2.

The horizontal axes of Figs. 1(a) and 2(a) represent the elapsed time since the start of the experiment, and their vertical axes give the number of identities that are connected or accepted (all accepted identities are also counted as connected). From a network crawl we estimate that the Tribler network contains roughly 50 000 nodes, meaning that

[1] https://www.tribler.org/.

approximately one hundredth of our network is reached in about five minutes.

The histograms of Figs. 1(b) and 2(b) show the number of occurrences of measured latencies, in intervals of 5 ms. The chosen time range of the histograms (0 through 1000 ms) omits one occurrence in Fig. 1(b) (at 1295 ms) and two occurrences in Fig. 2(b) (at 1655 ms and 1730 ms).

Our results show that Basic SybilSys correctly provides latency diversity and eventually finds honest nodes. The implementation repeatedly traverses its most-recently added neighbors starting from 10 initially accepted nodes, which is visible in Figs. 1(b) and 2(b): no single latency interval contains more than 10 accepted nodes. Furthermore, it can be observed that the accepted nodes are spread out over the RTT intervals, which is the latency diversity we seek. The second property of Basic SybilSys is that it eventually finds and accepts nodes in the network. However, the more Sybils are present in the network, the longer it will take to find latency-diverse nodes.

Our results highlight two implementation details of Basic SybilSys. First, in Figs. 1(a) and 2(a) sudden increases in the number of connections are visible (e.g., at 140 and 160 s in Fig. 2(a)), which occur due to the spam prevention for requesting random samples from the deployed rendezvous servers. The second detail is that the Basic SybilSys implementation uses latency-diverse random walks from latency-diverse starting points. As long as SybilSys cannot find 10 latency-diverse nodes to start from, it continues to repeatedly drop connections and request new random samples. This leads to a longer period of low numbers of connections as the number of Sybils grows.

## 5. Enhanced SybilSys: hardening to attacks

The Basic SybilSys algorithm presented in the previous sections is vulnerable to the *latency measurement interference attack*, in which an attacker may wait before answering a network latency measurement message. Furthermore, Basic SybilSys does not account for changes in latency (e.g., due to mobile nodes that physically travel) nor a node's initial accepted neighbors completely consisting of Sybils (causing a node to be perpetually introduced to only more Sybils). The issue of latency updates (and any other updates in routing path) is addressed in Section 5.1. Countering attackers that manage to occupy the initial set of accepted neighbors is discussed in Section 6.4.

### 5.1. Detecting measurement interference

When two message flows that are of "sufficient size" are routed over the Internet to a single address, they will join in a network interface queue at some point along their ways. The sizes of the message flows should be adjusted to account for the latency of the nodes that send them. For example, if two nodes both send only one message at the same time and their latencies to a shared queue are 50 ms and 100 ms, the queue has likely already forwarded the first message within the roughly 50 ms before the second message arrives. By sending more messages, over a longer time period, it becomes more likely that messages of the two message flows occupy the same queue at the same time. Thereby, additional queueing delay for messages occurs when they have to wait for messages of another flow.

Enhanced SybilSys leverages the fact that RTT measurements capture queuing delays and can therefore be used to detect joining message flows. By sending a message flow of latency probes (i.e., RTT measurement messages) to each of two identities, a joining of both flows is exposed through the RTT measurement responses. Our measurement strategy, called *TrafficJamTrigger*, creates these latency probe flows and adjusts their size such that the probes' response messages are likely to join.

The pattern that emerges from joining flows is a sudden (short) increase in measured RTT. In contrast, if messages follow the same physical path during their flow, i.e., they originate from the same physical machine and are Sybils, there is no deferred joining of these flows' physical paths and the phenomenon will not occur. Therefore, the lack of successful pattern detection can be used as an indicator for identities being Sybils. Secondarily, if messages do follow the same path (i.e., to Sybils on the same node), we expect higher RTTs to be measured over the entire series of measurement responses as all messages already occupy the same queues.

TrafficJamTrigger is applied to a message flow between a node sending latency probes and one of two identities that are simultaneously being measured. An identity is classified as an attacker if its flow does not show signs of the two flows joining. Enhanced SybilSys uses this classification to further filter nodes beyond using the latency diversity of Basic SybilSys. This intrinsically avoids identities whose message flows are routed through a single proxy (e.g., the Erebus attack that reroutes message flows through a single autonomous system [14]).

Our TrafficJamTrigger mechanism is grounded in existing work on the *flow correlation* attack on Tor [43]. The goal of the flow correlation attack is for an attacker to detect whether a single honest node receives messages from two message flows, for each of which the honest node uses a different identity. Thereby, the attacker wishes to defeat Tor's anonymization (that allows the honest node to communicate using the two identities without exposing the Internet address of its node). An attacker uses the correlation between a sudden increase in RTT and messages being sent over both message flows to infer that both flows lead to the same node. This attack has been shown to work for both TCP and UDP [44]. TrafficJamTrigger uses the concept of the flow correlation attack to determine whether identities are Sybils or not by creating RTT measurement message flows to those identities and looking for an RTT increase.

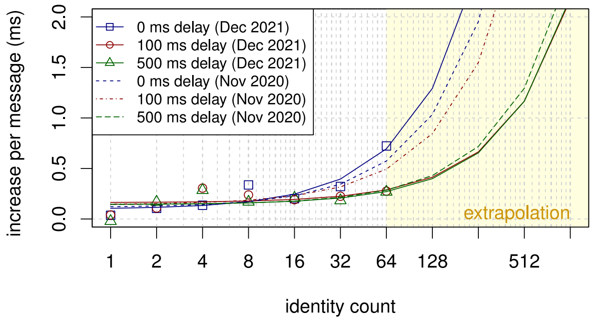
Routing over the Internet is not the same as routing over Tor. The routing of messages through Tor "circuits" follows fixed paths over the Internet. In contrast, any two messages sent to the same IP address may take radically different paths over the Internet. However, even though the paths are not explicitly defined by the application layer, paths over the Internet have enough temporal stability to apply the same technique: Internet paths can remain stable for days or months, and wireless paths are expected to last for multiple seconds, if not minutes [45].

### 5.2. Verifying the flow join pattern and its impact on RTT

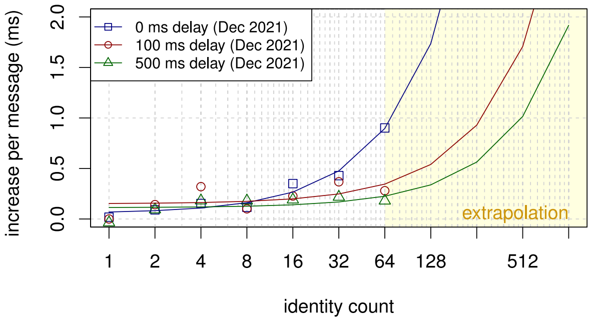
We now verify our claim that flow correlation can be used over the Internet through two small experiment setups. Our first setup shows the increase in the measured RTT as the number of identities on a single node increases when messages share a single path. Our second setup shows the patterns in RTT measurements of joining flows and of a single flow, focusing on the manner in which measurements increase over time instead of the increase's intensity. Both setups use TrafficJamTrigger.

For both setups, we implement TrafficJamTrigger by sending bursts of 20 messages to each identity that is being measured. The identities are sent such bursts in descending order of their "initial RTT", which is measured for each identity in isolation before using TrafficJamTrigger, in order to avoid message flow joins during these initial measurements.

Our first setup consists of two nodes connected over a LAN, with two intermediate routers. We use one node to measure the RTTs and the other to respond to RTT measurement requests. The measuring node uses TrafficJamTrigger in an attempt to detect an increase in the RTT of the identities operated from the responding node. To show to what extent the number of identities influences a latency measurement interference attack, we test three configurations with basic delays of 0 ms, 100 ms, and 500 ms that determine how long every Sybil waits before responding to a latency probe. Assuming the Sybils to be numbered starting from zero, Sybil $i$ uses a delay of $i \cdot d$, with $d$ the basic delay of the configuration. The delays correspond to an optimal attack, where an

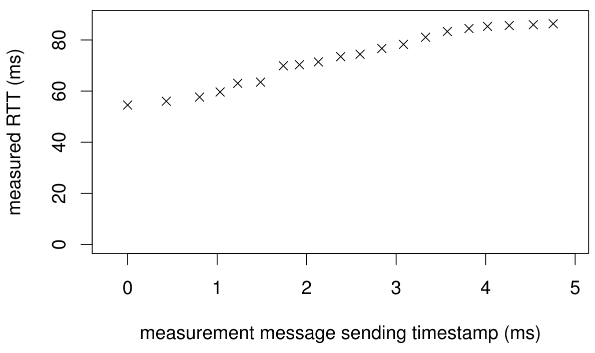


(a) Python implementation (measured in 2020 and 2021)

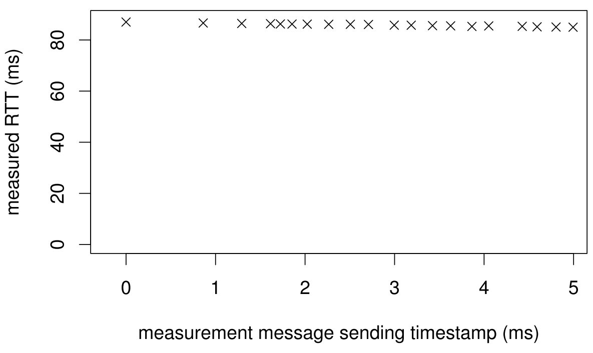


(b) Java implementation

**Fig. 3.** The measured latency increase versus the number of identities on a node with TrafficJamTrigger in comparison to the measured latency without using TrafficJamTrigger, with linear regression lines to provide extrapolation. The regression lines of 100 ms and 500 ms for December 2021 in Fig. 3(a) overlap.



(a)



(b)

**Fig. 4.** The measured RTT per message sent in a single 20-message burst versus the message sending timestamp (starting from 0) for an honest node (a) and a Sybil (b).

attacker only requires a single Sybil to take over every unique latency slot in the accepted node list.

Our results go up to the point where our measuring node starts filling its (network socket) receiving buffer, which we find at 64 identities. When the receiving buffer of the measuring node starts filling, the increase in RTT jumps into the order of dozens of milliseconds due to the limits of our hardware. In turn, if the jump is used to detect Sybils, the use of limited hardware would directly result in trivial Sybil detection. Therefore, to predict for more powerful hardware, we provide an extrapolation of our results for higher identity counts instead.

In Fig. 3 we present the measured RTT versus the number of identities on the responding node. We calculate linear regression models for each set of measurements and plot them as lines in our graphs (these would be straight lines if the horizontal axis would be linear instead of on a log scale). To show programming language (in)dependence, both a Python (Fig. 3(a)) and a Java (Fig. 3(b)) implementation are evaluated. To show the repeatability of this experiment we include the results from both 2020 and 2021 for the same Python implementation.

From the results of Fig. 3 we observe an increase in the measured RTT as the number of identities deployed on the responding node increases. The increase in RTT is even observed when Sybils wait longer before responding to probes. However, the RTT increase due to identities sharing a path should disappear when Sybils increase their waiting time for probe responses. Therefore, the shown increase using 500 ms, which is almost double the highest measured increase of 256.6 ms, must be due to other factors (e.g., identity management inefficiencies in our implementation). Whereas these other factors help in the detection of Sybils, we note that the 500 ms RTT increase could feasibly be eliminated with specialized attacker hardware. Nevertheless, when message flows do share a path (0 ms delay), the RTT clearly increases with the number of identities.

Our second setup, to show the pattern that emerges when flows join, consists of a measuring node that is connected to 50 honest nodes sampled from our Tribler network and two separate nodes that each run 25 Sybils. Of the two Sybil nodes, one is at a distance of 300 m on the same (TU Delft) campus as the measuring node, and the other is 48 km away, hosted in a datacenter. None of the 52 nodes that are connected to the measuring node are on the same LAN as the measuring node. We present the typical pattern of RTTs resulting from each of the bursts' response messages when two honest nodes are being measured (a) and when two Sybils are being measured (b). The latter situation, with two Sybils, corresponds to that of our first setup when two identities share the same node (see Fig. 3).

To highlight the difference between honest nodes and Sybils, Fig. 4 shows two typical bursts, which we found through qualitative analysis of several dozens of measurement pairs. The first obvious difference is the steady increase in measured RTT for the honest node versus its absence for the Sybil. However, this steady increase (roughly 1.6 ms between messages) simply shows the application's buffer being filled by other messages. This is not necessarily a result of applying TrafficJamTrigger. Instead, the pattern is characterized by the sudden jump in measured RTT for the honest nodes (at timestamp 1.74 ms, after the sixth measurement), which does not occur for the Sybils.

The effectiveness of classification using the pattern that we observed (i.e., whether it is statistically significant) depends on the manner in which the pattern is detected. Therefore, in the remainder of this work we define six binary classifiers to detect the RTT jump pattern and we evaluate them using real users.

## 5.3. Properties of Enhanced SybilSys

Enhanced SybilSys does not aim to prevent attackers from creating Sybils. An attacker can temporarily block communication between honest nodes using Sybils, known as an Eclipse attack [35]. Nevertheless, we summarily present the five properties of Enhanced SybilSys that greatly increase the effort needed for attackers to perform a successful Sybil attack, based on our findings of Section 4, and Section 5.

**Trustless self-reinforcing Sybil avoidance** is guaranteed through a bias toward honest nodes when introducing neighbors. Nodes keep connections open to those neighbors that they have accepted as non-Sybil through their latency-diversity property. Therefore, honest nodes are more likely to increasingly introduce and be introduced to other honest nodes without requiring a trust relationship with the introducing node.

**Attackers must create Sybils with unique RTTs** to be able to suppress honest nodes (governed by the threshold parameter $\Delta$). This suppression implies that Sybils must wholly occupy the accepted list of a node, which is filtered using latency diversity. By estimating what RTTs will be measured to honest nodes, an attacker can interfere with latency measurements to have its Sybils appear to match the estimations. This causes the honest nodes to be filtered out when they are eventually found. However, every diverse RTT of an honest node requires a new Sybil group. For example, to obtain a Sybil presence in a network that contains two honest nodes that differ in latency more than $\Delta$, an attacker must create two groups of Sybils that have RTTs within $\Delta$ of each honest node (doubling the number of required Sybils by using latency diversity).

**Attackers require low latency to targets** to compete with honest nodes. To perform an Eclipse attack, Sybils must have an overpowering presence within a latency of $\Delta$ of any honest node. However, the only attack on RTT measurements is to introduce additional delay, as they cannot be decreased by an attacker. Therefore, if an honest node exists that has a measured RTT that differs more than $\Delta$ from the node that operates Sybils, it is impossible to suppress this honest node using Sybils. As a consequence, honest nodes that are in close physical proximity cannot be attacked without an attacker buying hardware close to the targets. For a network-wide attack an attacker would have to buy hardware close to all honest nodes in the network, which makes a network-wide attack possible, but infeasible.

**Open networks (Web3) cannot be efficiently attacked** as the network participants are not known beforehand. In contrast, in networks with known participants, it is feasible for an attacker to measure and predict latencies between other network participants that are online for extended periods. However, Web3 technology is characterized by participants from all over the world that continuously join and leave the system. In order for an attacker to disrupt connectivity in such a network, all possible latencies must be preempted and have a majority of Sybils to suppress newly joining honest nodes with a unique latency. Therefore, practically, with a latency measurement timeout of 5 s and a diversity threshold of $\Delta = 5$ ms, allowing 1000 distinct latencies, and with a sufficient number of honest nodes, the required number of Sybils has to be increased thousand-fold in comparison to an equal Sybil attack without latency diversity.

**Low Sybil counts must be used per node** to decrease the chance of a connection being dropped. As the number of identities per node grows, it becomes increasingly likely that a connection to any of these identities is dropped. Therefore, a significant Sybil attack must be spread out over multiple nodes.

## 6. Enhanced SybilSys: implementation

In this section we discuss the implementation details of Enhanced SybilSys: the data structure that stores node identities, how flow joining is brought about, how it is detected when flows do join, maintaining and refreshing the node storage data structure, and the parameterization of Enhanced SybilSys. To implement flow recognition and data structure maintenance multiple options are presented, which are evaluated in Section 7.

### 6.1. The peer discovery tree

Every newly joining node in the system creates a peer discovery tree as its environment to operate in with itself as the root. Using the peer sampling process, it first selects an initial set of latency-diverse nodes, which we call the *bootstrap set*, as its children in this tree. Each of these children then helps the creator by building a (linear) branch of the tree consisting of latency-diverse nodes, starting with itself, of a pre-specified maximum length. In fact, they initiate a random walk, reporting back to the creator the identities of the nodes sampled along the branch and discarding those that are found by the creator to be not latency-diverse with any of the previous nodes in the branch. In the latter case, another node is sampled. As a consequence of this construction, the nodes in the bootstrap set and the nodes in each of the branches separately are latency diverse, but nodes in different branches may not be. An example of a possible peer discovery tree is given in Fig. 5.

TrafficJamTrigger must be applied to all pairs of nodes that require latency diversity. However, because neighborhoods are of limited size, this is not a problematic requirement. Furthermore, the pairs that require diversity are not all pairs of nodes in the peer discovery tree. For example, given the ten initially sampled latency-diverse nodes that grow branches of four diverse child nodes (as in Section 4.2), only 105 pairs need to be probed: 45 pairs for the ten initial nodes and 6 pairs for each of the 10 branches of four nodes. We continuously apply TrafficJamTrigger to random pairs in the peer discovery tree after the initial filtering using latency diversity. We discuss how a tree is updated when node pairs classify as Sybils in Section 6.4.

### 6.2. Bringing about message flow joins

TrafficJamTrigger sends bursts of RTT measurement messages to two identities to decide whether they share a node. However, it may not be sufficient to send two flows of measurement requests to two identities one after the other. For example, a node close to a measuring node may already have responded to all latency probes before a node that is further away has even started responding. This example would not lead to two message flows joining in a queue and therefore lead to the honest nodes being labeled as Sybils. To this end, our probe sending strategy consists of alternating short bursts of messages to the two identities instead.

When measuring two identities, TrafficJamTrigger sends a burst of messages to the identity with the higher RTT, which is immediately succeeded by messages to the other identity. The intent of this “slowest first” ordering is to account for the case when an honest node has a low latency and responds before the measuring node even finishes sending out its latency probes to the node with the higher RTT. In practice, we use the heuristic of alternatingly sending the same number of bursts (of 20 messages each) to both identities, one for every 200 ms of RTT of the slower identity (for example, two bursts to each identity if the highest RTT is 234 ms). Despite it being a heuristic, we show that this strategy works well enough to detect attackers in Section 7, and we leave further optimization to future work.

### 6.3. Recognizing message flow joins

Any binary classifier that is able to detect a sudden jump in a burst of measured RTTs, shown in Section 5.2, for the node with the highest initial RTT (Section 6.2) should suffice for TrafficJamTrigger. However, the steady increase or decrease of measured RTTs makes it more difficult to detect this jump. Queues may be in the process of being filled or emptied due to other messages (Section 5.2). For example, an RTT jump of 0.2 ms between two measurements over a steady increase of 1 ms leads to an RTT difference of 1.2 ms between them, while the jump is actually of equal magnitude compared to a difference of 0.2 ms when there is no steady increase.

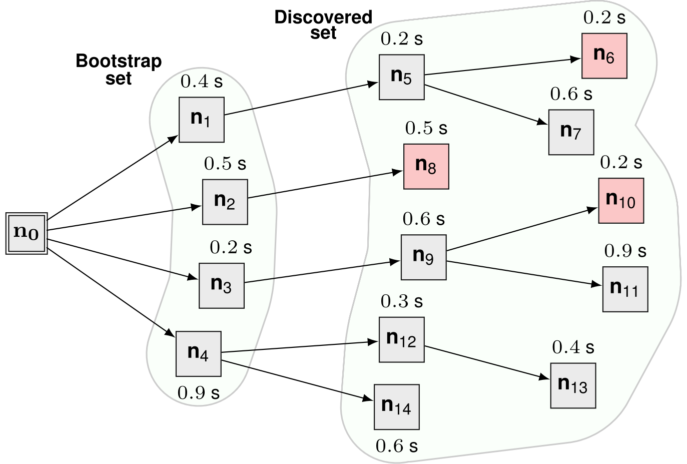


**Fig. 5.** Example of a peer discovery tree for a node $n_0$ with a maximum depth of 3 and $\Delta = 0.1$ s. Nodes with dark (red) shading violate the latency diversity. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

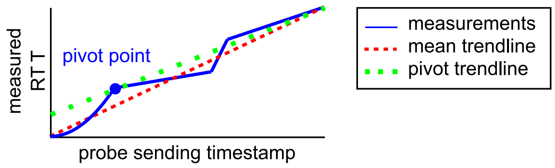


**Fig. 6.** Example construction of the pivot point, mean trendline, and pivot trendline, for an imaginary measurement curve.

A linear interpolation, of two specifically selected measurements of the same burst, is constructed to compensate for the increase (or decrease) during a burst. We call the linear interpolation a *trendline* and we explain the two ways in which we construct it shortly. A burst's measurements are recalculated as their difference to the value of their corresponding trendline at the time of sending a measurement probe.

We consider two types of trendlines: the *mean trendline* and the *pivot trendline*. The mean trendline starts with the first measurement of a TrafficJamTrigger burst and ends with its last measurement. The pivot trendline starts from the *pivot point* instead. The pivot point intuitively corresponds to the point where a queue has finished filling and streams have not joined yet (the fifth point in Fig. 4(a), just before the RTT jump between the sixth and the seventh point). We define this pivot point as the last measurement in a burst that increases more, in comparison to its predecessor, than the preceding measurement. As a visual reference, the two trendlines and the pivot point are shown for an imaginary measurement curve in Fig. 6.

We now present six classifiers for detecting the RTT-jump pattern, given in Table 1. The first three classifiers are based on the nearness of the RTT measurements to their trendline. We express this *nearness* as the mean squared error (MSE) of RTT measurements being smaller than a parameter $\epsilon$. Two identities are classified as Sybils if the TrafficJamTrigger measurements of the slower node have an MSE smaller than $\epsilon$. The difference between the three MSE-based classifiers lies in the RTT measurements that are included in the computation of the MSE: the *MSE* classifier uses all measurements, the *MSE pre-pivot* classifier uses all measurements up to the pivot point, and the *MSE post-pivot* uses only the measurements after the pivot point.

Two further classifiers are based on the shape of the measurement progression, taking no parameters. The *log-like* classifier is based on whether the majority of the measurements are higher than the trendline. The *wave-like* classifier decides if nodes are Sybils if the TrafficJamTrigger measurements do not cross the trendline exactly once, going from values smaller than the trendline to values larger than those of the trendline.

**Table 1**
Binary classifiers that detect artificial delay using the RTT burst measurements of the node with the highest initial RTT.

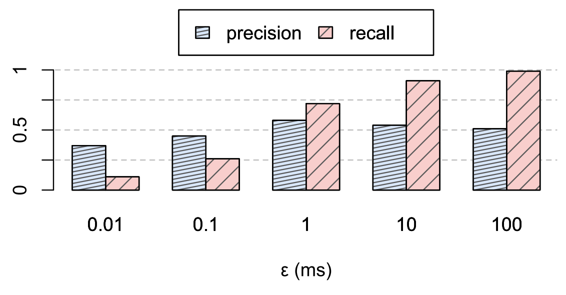
| Classifier | Attacker classification condition |
|---|---|
| MSE | MSE of all RTT measurements is smaller than $\epsilon$. |
| MSE pre-pivot | MSE of RTT measurements before the *pivot* is smaller than $\epsilon$. |
| MSE post-pivot | MSE of RTT measurements after the *pivot* is smaller than $\epsilon$. |
| Log-like | Most of the measurements are above the trendline. |
| Wave-like | RTT measurement progression does not go from below to above the trendline. |
| Baseline increase | The average of the measurements is a given percentage higher than the initial RTT measurement before the burst. |

Our final classifier is the *baseline increase* classifier, which exploits the latency increase phenomenon mentioned in Section 5.2. This classifier deems two identities to be Sybils if the identity with the highest RTT exceeds a given increase in average RTT during a burst compared to its initial RTT. The results of Fig. 3 show that this method increases in efficacy as the number of Sybils on a device grows.

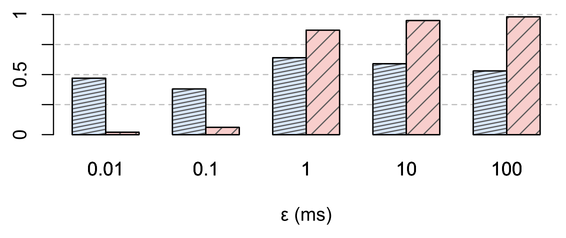
## 6.4. On accepted node removal

Enhanced SybilSys may require the removal of accepted nodes. Nodes may go offline, they may physically move (and no longer be latency-diverse), and TrafficJamTrigger may classify identity pairs as Sybils. Secondarily, nodes should be continuously removed (and resampled) to avoid the situation in which a successful attack could ever *permanently* hold all of the spots in a node's peer discovery tree if Sybils go undetected. In all of these cases a node's peer discovery tree has to be updated.

We consider three (binary) aspects that can be combined for a total of eight possible configurations for node removal from the peer discovery tree. First, we explore what node pairs to select for classification. We distinguish the selection of node pairs in the bootstrap set combined with those in each single branch (the *local* pairs) and selecting pairs from the entire peer discovery tree (*all* pairs). Secondly, we investigate what node pairs to remove for continuous resampling, the churn strategy. We consider periodically removing the node pair with the highest (*worst*) classifier score (though it may not be above a classifier's threshold), and removing a *random* node pair. Lastly, we inspect whether to either *remove* all following nodes in a branch when a node is removed or to only remove an intermediate node in a branch and *keep* its descendants by linking its child to its parent. In Section 6.5 we evaluate all eight configurations.



(a)



(b)

**Fig. 7.** The precision and recall for different values of $\varepsilon$ for the MSE classifier using the mean trendline (a) and the pivot trendline (b).

**Table 2**
The number of honest nodes found using the node removal strategies of Section 6.4 averaged over 30 experiment runs (higher is better).

| Node removal strategy | | | Number of honest nodes | | | |
|---|---|---|---|---|---|---|
| Pairs | Churn | Descendant | Mean | Median | Min | Max |
| *local* | *worst* | *remove* | 9.18 | 9 | 4 | 17 |
| *local* | *worst* | *keep* | 8.88 | 9 | 5 | 14 |
| *local* | *random* | *remove* | 9.39 | 9 | 5 | 15 |
| *local* | *random* | *keep* | 8.56 | 9 | 5 | 12 |
| *all* | *worst* | *remove* | 9.57 | 10 | 5 | 16 |
| *all* | *worst* | *keep* | 8.91 | 9 | 5 | 13 |
| *all* | *random* | *remove* | 9.95 | 10 | 6 | 17 |
| *all* | *random* | *keep* | **10.24** | 10 | 6 | 16 |

## 6.5. *Configuration for deployment*

We configure Enhanced SybilSys to seamlessly integrate with our existing Tribler network in order to deploy and evaluate it with real users of Tribler (Section 7). We adopt the parameterization of Tribler's peer sampling mechanism as much as possible, namely the total number of nodes and the periodicity of the peer discovery algorithm. First, we derive the peer discovery tree's bootstrap set size and its maximum branch length (see Section 6.1) from the total number of nodes. The remainder of this section presents small-scale experiments to configure parameters that cannot be derived from Tribler: the parameterization of the classifiers (given in Section 6.3) and the configuration of Enhanced SybilSys' node removal (Section 6.4).

We configure Enhanced SybilSys' peer discovery tree based on the peer sampling mechanism of Tribler. We adopt the total number of 20 as the target size of the peer discovery tree. Given the target number of 20 nodes, we parameterize SybilSys with 10 branches that grow to a branch length of 4 (if not stopped at 20 nodes, the tree would grow to a size of 40 nodes).

The experiments to parameterize the classifiers emphasize classifier quality over the speed of node discovery. The setup of our configuration experiments uses two nodes that each operate 25 Sybil identities and we sample 50 honest nodes from the Tribler network for each run (equal to the setup of Section 5.2). We run 60-second experiments using $\Delta = 0.05$ s, a bootstrap set size of 6, a maximum peer discovery tree branch length of 4 and a total tree size of 20 for each node. With less branches than in actual deployment, the quality of classifiers is stressed more.

We parameterize our classifiers (also known as "training" in Machine Learning) to provide a balance between recall and precision. On the one hand, the recall of any binary classifier corresponds to the chance that whatever entity is being classified, Sybils in our case, is classified as that entity. As Enhanced SybilSys uses classifiers to mark nodes for removal, the recall corresponds to the chance that a Sybil is (eventually) removed. The recall should therefore be as high as possible. On the other hand, the precision of a binary classifier corresponds to the fraction of entities that are correctly classified over all the entities. A precision lower than 0.5 causes Enhanced SybilSys to mostly mark honest nodes for removal. Inversely, because nodes share their neighbors with each other, a precision of over 0.5 means that nodes predominantly resample nodes from a set of nodes that is more likely to contain honest nodes. Therefore, if the precision is at least over 0.5, this resampling ensures that honest nodes are predominantly connected to other honest nodes.

We use our setup to configure the MSE-based classifiers and the baseline increase classifier. We show the impact of different values of $\varepsilon$ on the precision and recall for both the mean trendline in Fig. 7(a) and the pivot trendline in Fig. 7(b) for the MSE classifier. The results of Fig. 7 show a trade-off between the classifier's recall and its precision. We pick a value that lies between the collapse of the precision and the collapse of the recall, with a precision of at least 0.5. Therefore, for both trendlines, we configure the MSE classifier to use a value of $\varepsilon = 10$ ms. The pre-pivot and post-pivot parameterization have a similar collapse and we again choose values between their collapses, which we find at $\varepsilon = 10$ ms and $\varepsilon = 0.01$ ms respectively. In the same fashion, for the baseline increase classifier we pick a 20% increase in latency to classify Sybils.

In order to configure the node removal of Enhanced SybilSys, we evaluate the strategies of Section 6.4 using our experiment setup with the MSE classifier. We present the number of honest nodes each node-removal strategy yields in Table 2. Firstly, the result of applying classification over all pairs in the peer discovery tree is similar to that of classification within each branch and the bootstrap set, though classifying all pairs performs slightly better. Secondly, enforcing churn through continuously removing the "most Sybil" 2-tuple also does not lead to any big differences in honest node counts compared to random node removal, performing worse in all cases but one. Lastly, we find that—in all but one case—removing all descendants in a branch leads to more honest node retention than only removing the particular Sybil node and linking its child and parent node. Our results show that the configuration that retains the most honest nodes consists of applying classification over all node pairs, periodically removing random nodes and linking descendants to the parents of removed nodes in a branch.

## 6.6. *Overview of Enhanced Sybilsys*

Omitting the passive latency measurements (Section 4), bootstrap set construction (Section 6.1), and peer maintenance (Section 6.4), the activity diagram of Fig. 8 serves as a summary of our proposed algorithm. The periodicity of this algorithm consists of each node in the network running it once every 0.5 s (the default periodicity of Tribler's peer sampling).

## 7. Enhanced SybilSys: evaluation

We now evaluate both our flow joining detection classifiers and our complete Enhanced SybilSys algorithm (configured for deployment as described in Section 6.5) through two experiments. First, we construct several data sets of TrafficJamTrigger measurements to which we apply the six classifiers defined in Section 6.3. Second, we evaluate Enhanced SybilSys using the best-performing classifier to mimic a best-effort

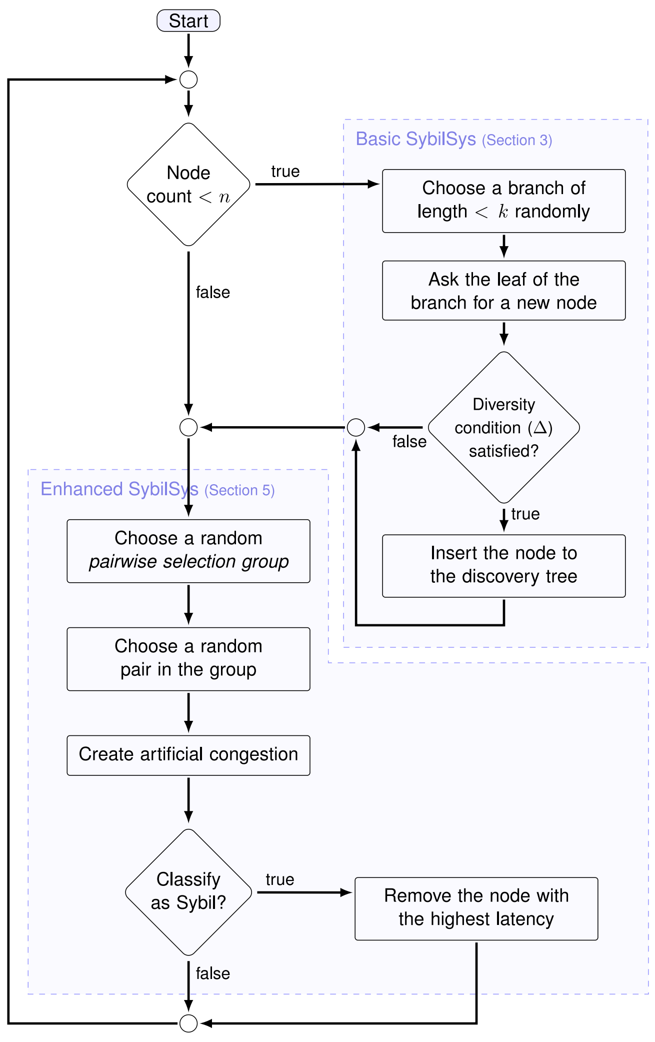

Start
Node count $< n$
true
false
Basic SybilSys (Section 3)
Choose a branch of length $< k$ randomly
Ask the leaf of the branch for a new node
Diversity condition ($\Delta$) satisfied?
false
true
Insert the node to the discovery tree
Enhanced SybilSys (Section 5)
Choose a random *pairwise selection group*
Choose a random pair in the group
Create artificial congestion
Classify as Sybil?
true
false
Remove the node with the highest latency

**Fig. 8.** Activity diagram of our Enhanced SybilSys for the requested total node count $n$, and the maximum branch length $k$.

defense by honest nodes. Finally, based on the experimental results, we estimate the cost of attacking Enhanced SybilSys.

Both experiments of this section use real users of our Tribler network and we introduce Sybils into the network that actively attack Enhanced SybilSys using the latency measurement interference attack. We denote the node that uses Enhanced SybilSys in order to filter out Sybils as the "measuring node" and we set its latency diversity threshold $\Delta = 5$ ms, based on the real-world deployment results of Section 4.2.

## 7.1. RTT-based classifiers evaluation

To evaluate our binary classifiers (Section 6.3), we create four data sets consisting of TrafficJamTrigger measurements, one for pairs of honest nodes and three for pairs of Sybils. We create more sets for Sybils to evaluate the impact of running Sybils on three different numbers of nodes, which should make them harder to detect (Section 5.2). The setup for all four of these sets consists of a measuring node that connects to all identities in a given set and performs TrafficJamTrigger measurements for all pairs of identities it is connected to. The data sets only contain the measurement bursts which are used for classification, those of the node with the highest RTT for each pair (Section 6.3). We do not perform TrafficJamTrigger measurements, and we do not create datasets, for pairs consisting of an honest node and a Sybil. The

**Table 3**
Overview of the data sets used for the evaluation of the classifiers.

| Data set | Nodes | Identities per node | Measured pairs |
|---|---|---|---|
| Honest | 500 | 1 | 124 750 |
| Sybil (1) | 1 | 100 | 4950 |
| Sybil (2) | 2 | 50 | 4950 |
| Sybil (3) | 4 | 25 | 4950 |

**Table 4**
Sybil classifier performance for both trendline types.

| Classifier | Mean | | Pivot | |
|---|---|---|---|---|
| | Precision | Recall | Precision | Recall |
| Random | 0.5021 | 0.4982 | 0.4984 | 0.4974 |
| MSE | 0.5372 | 0.9119 | 0.5895 | 0.9479 |
| MSE pre-pivot | 0.5319 | **0.9139** | 0.5860 | **0.9544** |
| MSE post-pivot | 0.4428 | 0.5877 | 0.1661 | 0.1037 |
| Log-like | 0.5457 | 0.4683 | **0.6577** | 0.9069 |
| Wave-like | **0.6283** | 0.4761 | 0.6544 | 0.9071 |
| Baseline increase | 0.5951 | 0.7350 | 0.5965 | 0.7392 |

reason is that we deem it unethical to Sybil attack unsuspecting users of Tribler. This limitation may skew our results. For reproducibility and further research, we have made our anonymized data sets publicly available [46].

Our honest set was created using the nodes of real Tribler users found with a network crawler that assumed the role of the measuring node. These users were running real workloads and had real network congestion during their measurement. All users had unique IP addresses and remained online for the duration of our experiment. Over the course of three days, our network crawler had eventually connected to 500 nodes and the experiment was started. TrafficJamTrigger measurements were performed by the crawler for all possible pairs of these 500 nodes, leading to 124 750 unique sequences of measurement bursts. The mean size of a sequence of bursts was 99 messages, and the median was 39 messages (to reiterate, we send 20 messages per 200 ms of RTT). Due to network effects, there were messages that did not receive a response. Out of 124 750 sequences of bursts, only 7414 had *all* of their individual messages responded to. Despite these network effects, Section 7.2 shows that a classifier can be used for effective Sybil filtering.

The Sybil sets capture artificially delayed Sybil attackers. We establish three data sets for different attacker setups, where the attacker nodes are in different physical locations: (1) a single node operating 100 Sybil identities, (2) two nodes operating 50 Sybil identities each, and (3) four nodes operating 25 Sybil identities each. Each setup operates in its own overlay network partition in order to keep Sybil attackers from communicating with Tribler users. TrafficJamTrigger measurements were performed for all possible Sybil pairs, including those running on different nodes. All three setups have 100 identities, each resulting data set includes 4950 TrafficJamTrigger sequences of measurement bursts per setup. For reference, the summary of our data sets is given in Table 3.

We evaluate the binary classifiers (parameterized using the training set of Section 6.5) using a test set comprised of our honest set and our Sybils sets. However, our honest set and Sybils sets are not of equal sizes (124 750 bursts sequences for honest nodes and three sets of 4950 bursts sequences for Sybils) and this *imbalance* would skew the precision and recall metrics [47]. To make up for the size differences, we use statistical bootstrapping, which consists of random resampling with replacement from the (smaller) Sybil sets to match the size of the (largest) honest set. Other methods can be used to equalize the set sizes (e.g., taking a subset of the honest set), though their estimation errors should all converge for a sufficiently-large smaller set [48]. Opinions on what constitutes a sufficiently-large set differ, but typically this is in the order of hundreds of data points (our smallest data set has 4950).

The precision and recall, calculated using our test set, for all of the RTT classifiers are presented in Table 4 for both trendline methods (see Section 6.3). For reference, we include the *random* classifier, which simply classifies an identity as Sybil with a 50% chance. We only include this reference to show that our statistical bootstrapping method is correctly implemented and leads to 50% precision and 50% recall.

Our results indicate that a jump in latency measurements is best detected by the MSE pre-pivot classifier, which classifies measurements as intermediate queues are filling before the pivot point. In contrast, the MSE post-pivot classifier shows both very poor precision and very poor recall, showing it is very difficult to detect Sybils after the pivot point using the Mean Squared Error, i.e., after queues are filled. This lack of detection aligns with the intuition that there can be no jump (up) in RTT measurements if all RTT probes already suffer the maximum queuing time. The MSE classifier, which considers both the pre-pivot and post-pivot measurements, achieves similar results to the pre-pivot classifier in spite of the inclusion of the post-pivot measurements. The log-like and wave-like classifiers have a relatively high precision of over 0.5 and are also usable with Enhanced SybilSys. These two classifiers benefit greatly from the pivot trendline, nearly doubling in recall value. The baseline increase classifier does not use a trendline and we have simply evaluated it twice, with similar results. All in all, all classifiers except the MSE post-pivot classifier are able to effectively filter Sybils with Enhanced SybilSys. Nevertheless, as recall is the decisive metric for convergence of the set of accepted nodes to a set of honest nodes, we use the MSE pre-pivot classifier for our final algorithm in the following evaluation.

## 7.2. Real-world evaluation of Enhanced SybilSys

Our second evaluation consists of combining the best performing Sybil classifier, MSE pre-pivot, with our Enhanced SybilSys peer discovery mechanism and evaluating it in a network with Sybil nodes. Again, every honest node is running a peer-to-peer file sharing client, leading to real-world congestion effects. The aim of our evaluation is to show that honest nodes are still able to find each other in these extremely challenging conditions. We explore two aspects of how an honest node finds other honest nodes: (1) the time it takes until it finds the first other honest node, and (2) how the number of honest nodes it has found progresses over time.

For both aspects we explore, we create a setup with in total 100 identities (both honest and Sybils). Our setup consists of a measuring node that is connected to a number of honest nodes from the Tribler network and up to four attacker nodes that run 25 Sybil identities each. To create networks with a given Sybil fraction, we randomly sample identities from these honest nodes and from the attackers' Sybil identities. For example, a network with a Sybil fraction of 0.75 contains 25 random honest nodes and 75 random Sybils, each of which may run on any of the four attacker nodes.

We determine the number of Sybils per node based on the results shown in previous sections. In Section 5.2 (the second setup) shows the relationship between the number of identities per node and how easily Sybils are detected using an initial RTT measurement, corresponding to the baseline increase classifier. As the latency of all identities on a single node grows with the number of identities on the node, so does the efficacy of our baseline increase classifier (see Section 6.3). Using the setup of Section 6.5, we derive that the largest number of Sybils per node that does not increase the recall of our baseline increase classifier is 25 identities (reaching a recall of 0.92 at 50 Sybil identities, equaling the MSE pre-pivot classifier).

We present the time until half of the honest (measuring) nodes has found another honest node. The reason for using this metric is that when there is at least one connection to another honest node, peer sampling is successful for Web3 applications (see Section 3). For brevity, we call the neighborhood of an honest node (excluding the measuring node itself) that includes at least one honest node *robust*. The results we present are averaged over 20 runs of at least 10 min.

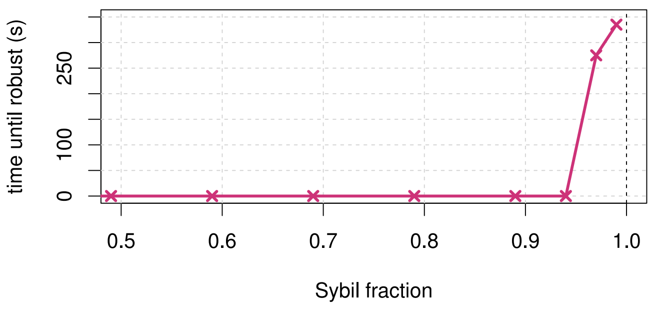


**Fig. 9.** Time until half of the honest nodes has found another honest node, i.e., until their neighborhoods are robust, as the Sybil fraction increases.

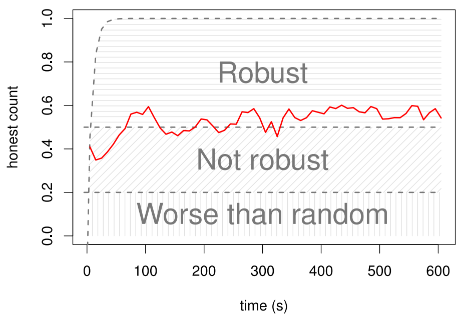


**Fig. 10.** The number of honest nodes found by the measuring node over time (the dark, red, line) in a 99% Sybil network. A random sample of 20 nodes leads to 0.2 found honest nodes, a robust neighborhood contains more than 0.5 honest nodes, and a perfect classifier upper bounds the number of found honest nodes with a geometric distribution.

The results of our first exploration given in Fig. 9 show that up to a Sybil fraction of 0.95 (with only 5 honest nodes for 100 identities), the neighborhoods of honest nodes are robust nearly instantly. This robustness is grounded in the chance of sampling at least one honest node with 20 samples, the target number of nodes (see Section 6.5), from a set of 100 identities that include 5 honest nodes (essentially binomial sampling). Our results underline that, up to a Sybils fraction of 0.95, Sybils can be trivially avoided by opening a sufficiently large number of connections. Nevertheless, Enhanced SybilSys is useful when the initial random sample of nodes does not contain an honest node and when the number of connections cannot be scaled up to match the Sybil presence in a network (e.g., due to security or hardware constraints). For the Sybil fractions of 0.97 and 0.99, an honest node using Enhanced SybilSys is expected to find another honest node after 275 and 335 s, respectively, with a granularity of five seconds.

We now zoom in on how the number of honest nodes found progresses over time for a setup with a fraction of Sybils of 0.99, which is the most challenging case for 100 identities. It forces Enhanced SybilSys to filter out latency-diverse Sybils using TrafficJamTrigger in order to find the single honest node. For comparison, we use two benchmarks, for both an upper bound on the number of honest nodes found (which could be reached using a perfect classifier with a precision and recall of 1.0) and the number expected to be found in a random sample (e.g., from a rendezvous server). Any Sybil-avoiding peer sampling mechanism is upper bounded by the chance to discover an honest node through iterative random sampling, which follows a geometric distribution. In contrast, a random sample of 20 uniformly randomly sampled nodes (the configured number of target nodes) leads to 0.2 honest nodes on average for a network that contains one honest node out of 100 nodes.

**Table 5**
The estimated cost of performing a Sybil attack with $20 hardware to delay honest node connections by 6 min, depending on the network size.

| Network | Year | Online nodes | Attack cost |
|---|---|---|---|
| Bitcoin | 2017 | 10k [49] | $0.79M |
| Ethereum | 2018 | 15k [50] | $1.19M |
| Kazaa | 2002 | 30k [51] | $2.38M |
| *Tribler* | 2020 | 50k [this work] | $3.96M |
| Napster | 2000 | 500k [52] | $39.6M |
| Napster | 2001 | 1.57M [53] | $124.3M |

The results depicted in Fig. 10 show that Enhanced SybilSys does not reach the theoretical maximum honest node count. However, Enhanced SybilSys does outperform the benchmark of purely random sampling. Our results show that the average number of connected honest nodes reaches over 0.5 after 6 min, at which point the majority of experiment runs have found and retained the singular honest node.

### 7.3. On the cost of attacking

We evaluate the monetary cost of delaying the discovery of honest nodes by six minutes (the time we found in Section 7.2 for a network to become robust). To put this into perspective, we use the statistics of Tribler, which is estimated to have 50 000 concurrent users. In Section 7.2, it is shown that the ratio of the number of Sybil identities to the number of honest nodes should be at least 99:1 to delay honest node discovery by six minutes. Therefore, using Section 7.2's setup of one node per 25 identities, a successful attack is required to use more than 198 000 nodes in different physical locations. Even if attackers use free Wi-Fi and old $20 Raspberry Pi's, a Sybil attack on this network would cost just under 4 million dollars. At this cost, attacks other than the Sybil attack are economically more viable to block communication. For example, cheaper attacks are a man-in-the-middle attack on all users (about $50\,000 \times 20\$$, excluding labor costs) and rented virtual private servers (given a $10 price tag per server, leading to $198\,000 \times 10\$$).

Using the same assumptions as for the estimate using our own network, we estimate the attack cost for several other popular peer-to-peer technologies. The costs are calculated using the estimated number of online nodes for each technology. The resulting cost to delay connections between honest nodes (for six minutes) is given in Table 5. For instance, we calculate a cost of $124.3M for the 1.57 million online nodes that Napster had in 2001. However, our estimate is based on measurements that were done for a file-sharing application and its heavy bandwidth load decreases classifier effectiveness, as observed in Section 7.1. We predict that applications that have lighter bandwidth loads will have more success in detecting Sybils. Therefore, using Sybils to attack networks where users have lighter bandwidth loads is expected to have a higher cost.

## 8. Related work

Decentralized techniques to avoid Sybil identities have been studied in many contexts, including computational puzzles [54–57], graph-based approaches [58–61] and reputation mechanisms [62–65].

In the case of computational puzzles, computational resources of nodes are challenged to limit the influence of attackers. SybilControl [54] proposes admission control that requires each identity to periodically solve computational puzzles, suppressing the influence of attackers who failed to solve a puzzle. To protect honest identities from devoting excessive computational effort, da Costa Cordeiro et al. use computational puzzles that have adaptive difficulty [57] and that are energy-efficient [66]. An issue with the computational puzzle approach is that attackers practically use the computational disparity between consumer devices and their hardware to take control over peer-to-peer networks, in particular the networks of cryptocurrencies. Besides, in the field of cryptocurrencies, these attacks are not just for the sake of vandalism: these attacks are profitable [20]. In contrast, our approach does not depend on the computational resources of attackers, but the routing infrastructure in between an attacker and a target, which is unlikely to be fully in the hands of an attacker given the global structure of the Internet.

Graph-based approaches leverage the sparse connections between Sybils and honest identities. SybilGuard [61] and SybilLimit [60] rely on random walks for ranking nodes to filter out attackers. However, both approaches suffer from a large number of false negatives as well as computational complexity due to the requirement of testing all suspect nodes. SybilRank [58] is less prone to false negatives and reduces the computational complexity of SybilGuard and SybilLimit. However, these graph-based approaches rely on a set of globally known honest nodes called *trust seeds*. In our approach, none of the components assume a-priori trusted nodes.

Reputation mechanisms rely on assigning numerical scores to each identity based on reported interaction histories [62–65]. However, addressing the problem of establishing trust relationships, through some form of decentralized reputation system, is also beholden to a plethora of attacks [67]. Creating trust requires a circular dependency between the creation of trust and the mitigation of attacks, which easily leads to reputation building with fake histories [68]. We have sought to avoid this problem in our own work by only mitigating attacks based on first-hand knowledge, completely avoiding both requiring centralized infrastructure and the need to establish trust.

**Latency as a Sybil classifier.** Many studies have attempted to use triangulation for detecting and avoiding Sybils, using latency to pinpoint physical coordinates [30,31] or (locally) by using signal strength [69–74]. However, the approach of signal strength, while highly similar, is not applicable for interactions over the Internet. Bazzi and Konjevod [30] exploit the geometric properties of network latency to test the distinctness of identities. However, this work builds on the assumption that the latency of nodes has strong geometric properties and that these latency measurements are symmetric. Empirical studies provide sufficient evidence to reject the validity of both assumptions [32–34]. In our work, we do not rely on those assumptions and consider artificially delayed latency measurements. Sherr et al. propose Veracity [31], a decentralized service for securing network coordinate systems, which also considers the case where an attacker artificially delays the latencies. However, it mainly addresses the coordinate system protection problem and has limited guarantees against malicious identities, providing results for a structured network (DHT) consisting of up to 40% active attackers, using 500 simulated nodes. We show in our experiments that the combination of Basic SybilSys and TrafficJamTrigger can handle up to 99% active attackers, using 500 real users, running real workloads.

**Digital Identity.** Detection and prevention of fake identities plays an essential role in identity management systems [66,75,76]. In a recent study, Stokkink et al. [75] have created a decentralized digital identity solution that requires no intermediation by third parties, suitable to even replace physical passports. This may seem to suggest a lack of communication to third parties and therefore to forego the problem of the Sybil attack. Indeed, as pointed out by Maram et al. [76], the building of identity information is Sybil-resistant. However, for the overall solution, this claim of Sybil-resistance would be false since these systems still require communication with others [75]. These solutions require decentrally established communication channels, which can be Sybil attacked. However, once communication to honest nodes is established, these solutions may be a great complement to our technology.

## 9. Conclusion

To aid in avoiding Sybil attacks on Web3 applications, this paper presented Basic SybilSys, a Sybil-avoiding peer discovery mechanism based solely on network latency. Secondarily, this paper introduced

TrafficJamTrigger, a network latency measurement strategy, to allow for detection of attacks that counter our Basic SybilSys mechanism. The combination of Basic SybilSys and TrafficJamTrigger forms Enhanced SybilSys.

Enhanced SybilSys promises to establish connections between users while avoiding Sybils without the need for trust or a centrally governed infrastructure. As Enhanced SybilSys only serves to establish connections, any of the numerous existing Sybil avoidance algorithms and mechanisms (e.g., computational puzzles, social network inference, reputation mechanisms and digital identities) can be used to complement Enhanced SybilSys. Whereas Enhanced SybilSys already avoids Sybils, future work may explore the complementary technology it enables to further filter Sybils and to achieve even less Sybils in a node's neighborhood. Furthermore, to improve Enhanced SybilSys, future work may also investigate other means of detecting Sybils using TrafficJamTrigger measurements.

Through tests with Enhanced SybilSys, it was shown to what extent network latency is able to ensure connections to non-Sybils over the Internet and what the expected monetary cost of undermining our Sybil avoidance is. It has been shown how non-Sybil nodes are still found even when a network consists of 99% Sybils. Using the results for the Tribler network, it is estimated that an attacker would have to invest millions of dollars to obtain a significant presence in a real overlay network to perform a successful (albeit temporary) Eclipse attack. Therefore, SybilSys succeeds in driving up the cost of the previously cheapest and easiest attack on the communication layer of Web3 solutions.

## CRediT authorship contribution statement

**Quinten Stokkink:** Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Software, Visualization, Writing – original draft. **Can Umut Ileri:** Conceptualization, Formal analysis, Methodology, Writing – original draft. **Dick Epema:** Project administration, Supervision, Writing – original draft. **Johan Pouwelse:** Formal analysis, Funding acquisition, Supervision, Writing – original draft.

## Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

## Data availability

The data sets used in this manuscript are publicly available at https://doi.org/10.4121/uuid:34850d65-1908-4249-b446-8e87c6d21ba0

## Acknowledgments

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors. The authors would like to thank Jan S. Rellermeyer for his contribution to an earlier version of this work. We thank Martijn de Vos whose comments helped improve and clarify this work.

## References

[1] Dave Peck, the PSL Team, An engineer's hype-free observations on web3 (and its possibilities), 2021, https://www.psl.com/feed-posts/web3-engineer-take.

[2] Shermin Voshmgir, Token Economy: How the Web3 Reinvents the Internet, Vol. 2, Token Kitchen, 2020.

[3] John R. Douceur, The sybil attack, in: International Workshop on Peer-To-Peer Systems, Springer, 2002, pp. 251–260.

[4] Usman W. Chohan, Are cryptocurrencies truly trustless? in: Cryptofinance and Mechanisms of Exchange, Springer, 2019, pp. 77–89.

[5] Hideaki Hata, Mingyu Guo, M. Ali Babar, Understanding the heterogeneity of contributors in bug bounty programs, in: 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM, IEEE, 2017, pp. 223–228.

[6] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, Ari Juels, Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability, in: 2020 IEEE Symposium on Security and Privacy, SP, IEEE, 2020, pp. 910–927.

[7] Dan Robinson, Georgios Konstantopoulos, Ethereum is a dark forest, 2020, https://www.paradigm.xyz/2020/08/ethereum-is-a-dark-forest/.

[8] Tal Be'ery, Ethology: A safari tour in ethereum's dark forest, 2020, https://zengo.com/ethology-a-safari-tour-in-ethereums-dark-forest/.

[9] DeGate Team, An analysis of ethereum front-running and its defense solutions, 2021, https://medium.com/degate/an-analysis-of-ethereum-front-running-and-its-defense-solutions-34ef81ba8456.

[10] Kenneth P. Birman, Mark Hayden, Oznur Ozkasap, Zhen Xiao, Mihai Budiu, Yaron Minsky, Bimodal multicast, ACM Trans. Comput. Syst. (TOCS) 17 (2) (1999) 41–88.

[11] Márk Jelasity, Spyros Voulgaris, Rachid Guerraoui, Anne-Marie Kermarrec, Maarten Van Steen, Gossip-based peer sampling, ACM Trans. Comput. Syst. (TOCS) 25 (3) (2007) 8–es.

[12] Ken Birman, The promise, and limitations, of gossip protocols, Oper. Syst. Rev. 41 (5) (2007) 8–13.

[13] Ghassan O. Karame, Elli Androulaki, Srdjan Capkun, Double-spending fast payments in bitcoin, in: Proceedings of the 2012 ACM Conference on Computer and Communications Security, 2012, pp. 906–917.

[14] Muoi Tran, Akshaye Shenoi, Min Suk Kang, On the routing-aware peering against network-eclipse attacks in bitcoin, in: 30th USENIX Security Symposium (USENIX Security 21), 2021.

[15] Harry C. Li, Allen Clement, Edmund L. Wong, Jeff Napper, Indrajit Roy, Lorenzo Alvisi, Michael Dahlin, Bar gossip, in: Proceedings of the 7th Symposium on Operating Systems Design and Implementation, 2006, pp. 191–204.

[16] Márk Jelasity, Rachid Guerraoui, Anne-Marie Kermarrec, Maarten Van Steen, The peer sampling service: Experimental evaluation of unstructured gossip-based implementations, in: ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing, Springer, 2004, pp. 79–98.

[17] Anne-Marie Kermarrec, Alessio Pace, Vivien Quema, Valerio Schiavoni, Nat-resilient gossip peer sampling, in: 2009 29th IEEE International Conference on Distributed Computing Systems, IEEE, 2009, pp. 360–367.

[18] Adja Elloh Yves-Christian, Badis Hammi, Ahmed Serhrouchni, Houda Labiod, Total eclipse: How to completely isolate a bitcoin peer, in: 2018 Third International Conference on Security of Smart Cities, Industrial Control System and Communications, SSIC, IEEE, 2018, pp. 1–7.

[19] Abdelatif Hafid, Abdelhakim Senhaji Hafid, Mustapha Samih, Scaling blockchains: A comprehensive survey, IEEE Access 8 (2020) 125244–125262.

[20] Savva Shanaev, Arina Shuraeva, Mikhail Vasenin, Maksim Kuznetsov, Cryptocurrency value and 51% attacks: evidence from event studies, J. Altern. Invest. 22 (3) (2019) 65–77.

[21] Muhammad Saad, Victor Cook, Lan Nguyen, My T. Thai, Aziz Mohaisen, Partitioning attacks on bitcoin: Colliding space, time, and logic, in: 2019 IEEE 39th International Conference on Distributed Computing Systems, ICDCS, IEEE, 2019, pp. 1175–1187.

[22] Muneeb Ali, Jude Nelson, Ryan Shea, Michael J. Freedman, Bootstrapping trust in distributed systems with blockchains, USENIX; Login 41 (3) (2016) 52–58.

[23] Mansour Alsaleh, Abdulrahman Alarifi, Abdul Malik Al-Salman, Mohammed Alfayez, Abdulmajeed Almuhaysin, Tsd: Detecting sybil accounts in twitter, in: 2014 13th International Conference on Machine Learning and Applications, IEEE, 2014, pp. 463–469.

[24] Jing Jiang, Zifei Shan, Wenpeng Sha, Xiao Wang, Yafei Dai, Detecting and validating sybil groups in the wild, in: 2012 32nd International Conference on Distributed Computing Systems Workshops, IEEE, 2012, pp. 127–132.

[25] P. Gao, B. Wang, N.Z. Gong, S.R. Kulkarni, K. Thomas, P. Mittal, Sybilfuse: Combining local attributes with global structure to perform robust sybil detection, in: 2018 IEEE Conference on Communications and Network Security, CNS, 2018, pp. 1–9.

[26] Anastasia Koloskova, Sebastian Stich, Martin Jaggi, Decentralized stochastic optimization and gossip algorithms with compressed communication, in: International Conference on Machine Learning, PMLR, 2019, pp. 3478–3487.

[27] Clement Fung, Chris J.M. Yoon, Ivan Beschastnikh, Mitigating sybils in federated learning poisoning, 2018, arXiv preprint arXiv:1808.04866.

[28] Clement Fung, Chris J.M. Yoon, Ivan Beschastnikh, The limitations of federated learning in sybil settings, in: 23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020), 2020, pp. 301–316.

[29] Dimitris Vyzovitis, Yusef Napora, Dirk McCormick, David Dias, Yiannis Psaras, Gossipsub: Attack-resilient message propagation in the filecoin and eth2. 0 networks, 2020, arXiv preprint arXiv:2007.02754.

[30] Rida A. Bazzi, Goran Konjevod, On the establishment of distinct identities in overlay networks, Distrib. Comput. 19 (4) (2007) 267–287.

[31] Micah Sherr, Matt Blaze, Boon Thau Loo, Veracity: Practical secure network coordinates via vote-based agreements, in: USENIX Annual Technical Conference, 2009.

[32] Jonathan Ledlie, Paul Gardner, Margo I. Seltzer, Network Coordinates in the Wild, Vol. 7, NSDI, 2007, pp. 299–311.

[33] Cristian Lumezanu, Randy Baden, Neil Spring, Bobby Bhattacharjee, Triangle inequality variations in the internet, in: ACM SIGCOMM Conference on Internet Measurement, ACM, 2009.
[34] Raúl Landa, Joao Taveira Araújo, Richard G. Clegg, Eleni Mykoniati, David Griffin, Miguel Rio, The large-scale geography of internet round trip times, in: 2013 IFIP Networking Conference, IEEE, 2013, pp. 1–9.
[35] Atul Singh, et al., Eclipse attacks on overlay networks: Threats and defenses, in: IEEE INFOCOM, Citeseer, 2006.
[36] Brian Neil Levine, Clay Shields, N. Boris Margolin, A Survey of Solutions To the Sybil Attack, Vol. 7, University of Massachusetts Amherst, Amherst, MA, 2006, p. 224.
[37] Paul Baran, On distributed communications networks. rand corporation, in: P-2626), Santa Monica, 1962, 40 Pp, 32, 1962, pp. 168–267.
[38] Georgios Loukas, Gülay Öke, Protection against denial of service attacks: A survey, Comput. J. 53 (7) (2010) 1020–1037.
[39] Bryan Ford, Pyda Srisuresh, Dan Kegel, Peer-to-peer communication across network address translators, in: USENIX Annual Technical Conference, General Track, 2005, pp. 179–192.
[40] Frank Dabek, Russ Cox, Frans Kaashoek, Robert Morris, Vivaldi: A decentralized network coordinate system, in: ACM SIGCOMM Computer Communication Review, Vol. 34, ACM, 2004, pp. 15–26, (4).
[41] Ashwin Lall, Data streaming algorithms for the kolmogorov-smirnov test, in: 2015 IEEE International Conference on Big Data (Big Data), IEEE, 2015, pp. 95–104.
[42] Johan Pouwelse, Pawel Garbacki, Jun Wang, Arno Bakker, Jie Yang, Alexandru Iosup, Dick Epema, Marcel Reinders, Maarten Van Steen, Henk J. Sips, Tribler: a social-based peer-to-peer system, Concurr. Comput.: Pract. Exper. 20 (2) (2008) 127–138.
[43] Tadayoshi Kohno, Andre Broido, Kimberly C. Claffy, Remote physical device fingerprinting, IEEE Trans. Dependable Secure Comput. 2 (2) (2005) 93–108.
[44] Sebastian Zander, Steven J. Murdoch, An improved clock-skew measurement technique for revealing hidden services, in: USENIX Security Symposium, 2008, pp. 211–226.
[45] Krishna Ramachandran, Irfan Sheriff, Elizabeth Belding, Kevin Almeroth, Routing stability in static wireless mesh networks, in: International Conference on Passive and Active Network Measurement, Springer, 2007, pp. 73–82.
[46] Quinten Stokkink, Can Umut Ileri, Johan Pouwelse, Jan S. Rellermeyer, Latency Collision Measurements. 4TU.Centre for Research Data, Dataset, 2020, http://dx.doi.org/10.4121/uuid:34850d65-1908-4249-b446-8e87c6d21ba0.
[47] Sotiris Kotsiantis, Dimitris Kanellopoulos, Panayiotis Pintelas, et al., Handling imbalanced datasets: A review, GESTS Int. Trans. Comput. Sci. Eng. 30 (1) (2006) 25–36.
[48] Cathy O'Neil, Rachel Schutt, Doing Data Science: Straight Talk from the Frontline, O'Reilly Media, Inc., 2013.
[49] Varun Deshpande, Hakim Badis, Laurent George, Btcmap: mapping bitcoin peer-to-peer network topology, in: 2018 IFIP/IEEE International Conference on Performance Evaluation and Modeling in Wired and Wireless Networks, PEMWN, IEEE, 2018, pp. 1–6.
[50] Seoung Kyun Kim, Zane Ma, Siddharth Murali, Joshua Mason, Andrew Miller, Michael Bailey, Measuring ethereum network peers, in: Proceedings of the Internet Measurement Conference 2018, 2018, pp. 91–104.
[51] Jian Liang, Rakesh Kumar, Keith W. Ross, The kazaa overlay: A measurement study, Comput. Netw. J. (Elsevier) 49 (6) 2005.
[52] CNN Money, Napster: 20 million users. https://money.cnn.com/2000/07/19/technology/napster/index.htm.
[53] Jintae Lee, An end-user perspective on file-sharing systems, Commun. ACM 46 (2) (2003) 49–53.
[54] Frank Li, Prateek Mittal, Matthew Caesar, Nikita Borisov, Sybilcontrol: Practical sybil defense with computational puzzles, in: Proceedings of the Seventh ACM Workshop on Scalable Trusted Computing, 2012, pp. 67–78.
[55] Miguel Castro, Peter Druschel, Ayalvadi Ganesh, Antony Rowstron, Dan S. Wallach, Secure routing for structured peer-to-peer overlay networks, Oper. Syst. Rev. 36 (SI) (2002) 299–314.
[56] Hosam Rowaihy, William Enck, Patrick McDaniel, Thomas La Porta, Limiting sybil attacks in structured p2p networks, in: IEEE INFOCOM 2007-26th IEEE International Conference on Computer Communications, IEEE, 2007, pp. 2596–2600.
[57] Weverton Luis da Costa Cordeiro, Flávio Roberto Santos, Gustavo Huff Mauch, Marinho Pilla Barcelos, Luciano Paschoal Gaspary, Identity management based on adaptive puzzles to protect p2p systems from sybil attacks, Comput. Netw. 56 (11) (2012) 2569–2589.
[58] Qiang Cao, Michael Sirivianos, Xiaowei Yang, Tiago Pregueiro, Aiding the detection of fake accounts in large scale social online services, in: 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12), 2012, pp. 197–210.
[59] Nguyen Tran, Jinyang Li, Lakshminarayanan Subramanian, Sherman S.M. Chow, Optimal sybil-resilient node admission control, in: 2011 Proceedings IEEE INFOCOM, IEEE, 2011, pp. 3218–3226.
[60] Haifeng Yu, Phillip Gibbons, Michael Kaminsky, Feng Xiao, Sybillimit: A near-optimal social network defense against sybil attacks, in: 2008 IEEE Symposium on Security and Privacy (Sp 2008), IEEE, 2008, pp. 3–17.
[61] Haifeng Yu, Michael Kaminsky, Phillip Gibbons, Abraham Flaxman, Sybilguard: defending against sybil attacks via social networks, ACM SIGCOMM Comput. Commun. Rev. 36 (4) (2006) 267–278.
[62] Sepandar D. Kamvar, Mario T. Schlosser, Hector Garcia-Molina, The eigentrust algorithm for reputation management in p2p networks, in: Proceedings of the 12th International Conference on World Wide Web, 2003, pp. 640–651.
[63] Atsushi Yamamoto, Daisuke Asahara, Tomoko Itao, Satoshi Tanaka, Tatsuya Suda, Distributed pagerank: a distributed reputation model for open peer-to-peer network, in: 2004 International Symposium on Applications and the Internet Workshops. 2004 Workshops, IEEE, 2004, pp. 389–394.
[64] Michel Meulpolder, Johan A. Pouwelse, Dick H.J. Epema, Henk J. Sips, Bartercast: A practical approach to prevent lazy freeriding in p2p networks, in: 2009 IEEE International Symposium on Parallel & Distributed Processing, IEEE, 2009, pp. 1–8.
[65] Alex Biryukov, Daniel Feher, Recon: Sybil-resistant consensus from reputation, Pervasive Mob. Comput. 61 (2020) 101109.
[66] Weverton Luis da Costa Cordeiro, Flávio Roberto Santos, Marinho Pilla Barcellos, Luciano Paschoal Gaspary, Hanna Kavalionak, Alessio Guerri-eri, Alberto Montresor, Making puzzles green and useful for adaptive identity management in large-scale distributed systems, Comput. Netw. 95 (2016) 97–114.
[67] Eleni Koutrouli, Aphrodite Tsalgatidou, Taxonomy of attacks and defense mechanisms in p2p reputation systems - lessons for reputation system designers, Comp. Sci. Rev. 6 (2–3) (2012) 47–70.
[68] Alexander Stannat, Can Umut Ileri, Dion Gijswijt, Johan Pouwelse, Achieving sybil-proofness in distributed work systems, in: International Conference on Autonomous Agents and Multiagent Systems 2021.
[69] Arthanareeswaran Angappan, T.P. Saravanabava, P. Sakthivel, K.S. Vishvaksenan, Novel sybil attack detection using rssi and neighbour information to ensure secure communication in wsn, J. Ambient Intell. Humaniz. Comput. 12 (6) (2021) 6567–6578.
[70] Yong Huang, Wei Wang, Yiyuan Wang, Tao Jiang, Qian Zhang, Lightweight sybil-resilient multi-robot networks by multipath manipulation, in: IEEE INFOCOM 2020-IEEE Conference on Computer Communications, IEEE, 2020, pp. 2185–2193.
[71] Chundong Wang, Likun Zhu, Liangyi Gong, Zhentang Zhao, Lei Yang, Zheli Liu, Xiaochun Cheng, Accurate sybil attack detection based on fine-grained physical channel information, Sensors 18 (3) (2018) 878.
[72] Yue Liu, David R. Bild, Robert P. Dick, Z. Morley Mao, Dan S. Wallach, The mason test: A defense against sybil attacks in wireless networks without trusted authorities, IEEE Trans. Mob. Comput. 14 (11) (2015) 2376–2391.
[73] Gilles Guette, Bertrand Ducourthial, On the sybil attack detection in vanet, in: 2007 IEEE International Conference on Mobile Adhoc and Sensor Systems, IEEE, 2007, pp. 1–6.
[74] Murat Demirbas, Youngwhan Song, An rssi-based scheme for sybil attack detection in wireless sensor networks, in: 2006 International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM'06), ieee, 2006, pp. 5–pp.
[75] Quinten Stokkink, Georgy Ishmaev, Dick Epema, Johan Pouwelse, A truly self-sovereign identity system, in: 2021 IEEE 46th Conference on Local Computer Networks, LCN, IEEE, 2021, pp. 1–8.
[76] Deepak Maram, Harjasleen Malvai, Fan Zhang, Nerla Jean-Louis, Alexander Frolov, Tyler Kell, Tyrone Lobban, Christine Moy, Ari Juels, Andrew Miller, Candid: Can-do decentralized identity with legacy compatibility, sybil-resistance, and accountability, in: 2021 IEEE Symposium on Security and Privacy, SP, IEEE, 2021, pp. 1348–1366.

**Quinten Stokkink** is a Ph.D. student at the Delft University of Technology where he performs research on cooperative systems, specializing in Self-Sovereign Identity. After receiving his M.Sc. in 2017, he created the IPv8 networking library, serving both to power the first Dutch Government backed Self-Sovereign Identity prototype and to replace the Dispersy elastic database library within the Tribler video-on-demand client. His current research interests lie in defining the required networking technology for digital representation of natural persons and its security considerations. He has published papers on both details of Self-Sovereign Identity technology and its ethical considerations.

**Can Umut Ileri** is a PostDoctoral researcher for Distributed Systems at TU Delft and the Delft Blockchain Lab. His research interests lie in distributed algorithms for cooperative systems. His current research focuses on building a universal computational model for trust in decentralized systems. Before joining TU Delft, he worked as a research assistant at Ege University where he did his PhD. During his Ph.D. studies, he worked on distributed algorithms, with a special focus on graph theory and self-stabilization. He designed self-stabilizing algorithms for graph-theoretical problems such as capacitated matching, capacitated minimum spanning tree and k-way graph partitioning.

**Dick H.J. Epema** is a full professor of Distributed Systems at Delft University of Technology. His research interests are in the areas of resource management and scheduling in distributed systems, including datacenters and clouds, and of cooperative systems, including blockchains. He is the Director of the Delft Blockchain Lab, which brings together all blockchain-related research of Delft University of Technology. He has published over 150 papers, and he has organized such conferences as CCGrid, HPDC, and the IEEE P2P Conference.

**Johan Pouwelse** is an associate professor at Delft University of Technology, specialized in large-scale cooperative systems. During his Ph.D. he created the first system for cooperative resource management. The resulting driver got accepted into the Linux kernel and this architecture is still used by every Android and iOS device. Also, he conducted the first resource usage measurements for IEEE 802.11b, known now as wifi. After receiving his Ph.D., he conducted one of the largest measurements of the Bittorrent P2P network. He founded the Tribler video-on-demand client in 2005, which has been installed by 1.8 million people over the past decade.