

Formal Model-Driven Analysis of Resilience of GossipSub to Attacks from Misbehaving Peers

Ankit Kumar[◇]

Northeastern University

kumar.anki@northeastern.edu

Max von Hippel[◇]

Northeastern University

vonhippel.m@northeastern.edu

Pete Manolios[†]

Northeastern University

pete@ccs.neu.edu

Cristina Nita-Rotaru[†]

Northeastern University

c.nitarotaru@northeastern.edu

Abstract—GossipSub is a new peer-to-peer communication protocol designed to counter attacks from misbehaving peers by carefully controlling what information is disseminated and to whom, via a *score function* computed by each peer that captures positive and negative behaviors of its neighbors. The score function depends on several parameters (weights, caps, thresholds, etc.) that can be configured by applications using GossipSub. The specification for GossipSub is written in English and its resilience to attacks from misbehaving peers is supported empirically by emulation testing using an implementation in Golang.

In this work we take a foundational approach to understanding the resilience of GossipSub to attacks from misbehaving peers. We build the first formal model of GossipSub, using the ACL2s theorem prover. Our model is officially endorsed by GossipSub developers. It can simulate GossipSub networks of arbitrary size and topology, with arbitrarily configured peers, and can be used to prove and disprove theorems about the protocol. We formalize fundamental security properties stating that the score function is fair, penalizes bad behavior and rewards good behavior. We prove that the score function is always fair, but can be configured in ways that either penalize good behavior or ignore bad behavior. Using our model, we run GossipSub with the specific configurations for two popular real-world applications: the FileCoin and Eth2.0 blockchains. We show that all properties hold for FileCoin. However, given any Eth2.0 network (of any topology and size) with any number of potentially misbehaving peers, we can synthesize attacks where these peers are able to continuously misbehave by never forwarding topic messages, while maintaining positive scores so that they are never pruned from the network by GossipSub.

1. Introduction

Gossip protocols are a fundamental building block for communication in peer-to-peer (P2P) systems. One such protocol is GossipSub, and is used by popular applications such as FileCoin [1] and Eth2.0 [2]. As of November 2022, the market cap of FileCoin is \$1.4B USD [3]. Eth2.0 is the

second most valuable cryptocurrency, after Bitcoin, with a November 2022 market cap of \$143B USD [4]. A major concern in P2P systems including GossipSub is the risk of attacks launched by misbehaving peers. As such attacks can have significant financial implications, it is critical that GossipSub is resilient to attacks from misbehaving peers.

GossipSub addresses attacks by misbehaving peers via fine-granularity dissemination techniques and heuristic defense mechanisms that are carefully controlled by a score function capturing positive and negative peer behaviors, globally and within topics. (GossipSub partitions communication by *topics* to which peers can subscribe or unsubscribe – as in pub/sub systems.) Scores calculated using the score function are local to a peer and are not shared with other peers. There are many aspects of the score function that are crucial to its correctness such as: defining good and bad behavior, specifying weights that are applied to good and bad behavior in the score function, and choosing thresholds for making determinations about peer behaviors, to name just a few. Applications using GossipSub have the ability to configure such parameters.

Where does the assurance that GossipSub is indeed resilient to attacks from misbehaving peers come from? GossipSub is defined by a prose specification [5], [6] and an implementation in Golang [7]. The Golang implementation was subjected to unit tests and manual code review by expert programmers, and tested under various threat models at scale using the TESTGROUND network emulator [8], [9].

While the manual reviews and emulations empirically suggest that the protocol is resilient to attacks, there is no rigorous specification of what this resilience means or of what assumptions are necessary for the correctness of the system. Formal methods can help disambiguate system specifications and formulate implicit assumptions made by the system designers. They also can expose flaws in system requirements, often not captured through testing. In contrast to testing, formal methods provide mathematical proofs that show that a system does or does not behave correctly.

Formal methods have been previously applied to gossip and pub/sub protocols [10], [11], [12], [13], [14], however with important limitations. First, they studied much simpler protocols than GossipSub, based on *flooding*, where data is disseminated naively across the network without regard for the bandwidth. Second, they used model checking and made

◇. Contributed equally.

†. Listed alphabetically.

simplifying assumptions to avoid state-space explosion.

1.1. Our contributions

We focus on rigorously studying GossipSub and its resilience to attacks from misbehaving peers using ACL2s, a theorem prover based on a purely functional LISP-based language [15], [16]. ACL2s is highly expressive, allowing us to express arbitrary computation and properties over infinite-state systems in a single framework. Our contributions are:

- **A formal model of GossipSub:** In contrast to prior works that studied gossip protocols using heavily simplified or restricted models, we study GossipSub by modeling every aspect of its prose specification. Our model is not just an abstraction, it is an *actual executable program* that can be formally reasoned about. In cases where we find the specification ambiguous, we compare it to the implementation and also consult the specification authors. Our model can simulate GossipSub networks of arbitrary size and topology, with arbitrarily configured peers, and can be used to prove and disprove protocol properties. Our model code will be made publicly available after publication, allowing developers of applications using or planning to use GossipSub to verify security properties for the configuration corresponding to their application. Our model has been officially endorsed by GossipSub developers as a formal specification for the protocol in their documentation [17].

- **Security properties and analyses:** Since the prose specification [5], [6] and emulation analysis [8] do not list explicit properties, we formalize four security properties about the score function that can be inferred based on a close reading of these documents. These properties are necessary for the score-based defense mechanisms to defend against attacks from misbehaving peers.

- (1) If a peer’s performance for some topic is continuously non-positive, then, eventually, the peer’s score will be non-positive.
- (2) When a peer misbehaves, its score decreases.
- (3) When a peer behaves, its score does not decrease.
- (4) Peers are scored fairly: if they appear to behave identically, they are given identical scores.

We prove that (4) holds for all GossipSub configurations. In contrast, using ACL2s, we automatically find configurations for which (1) and (2) fail. We prove that the configuration used by Eth2.0 is one such configuration where both these properties fail, and we prove that the FileCoin configuration satisfies all four although it achieves this satisfaction by compromising important protocol functionality. We discuss the implications of property failures for Eth2.0 and FileCoin.

- **Attack generation:** We show how violations of these properties for Eth2.0 can be used to create attacks against the entire network. Our attacks exploit the fact that the score function can be configured in ways where peers can misbehave without penalty. Eth2.0 uses one such configuration. To find these attacks we formalize what it means for the protocol to behave correctly, and then ask the ACL2s

theorem prover if it was possible for the protocol to behave incorrectly. In contrast, prior emulation and expert code review of GossipSub only looked at specific pre-programmed attack scenarios, *e.g.*, where an honest peer is surrounded by malicious peers who delay or drop messages forwarded from the honest peer, or where the network is saturated with malicious peers who instantaneously stop forwarding data. We take a more general approach, formalizing what it means for GossipSub to behave correctly, and then asking whether any attack scenarios exist – including unknown ones – in which the protocol might behave incorrectly. We synthesize and verify attacks violating the first property for Eth2.0. These attacks can be carried out on any Eth2.0 network, regardless of the topology or size. The attacks allow peers to continuously misbehave, by never forwarding topic messages, while maintaining positive scores so that they are never pruned from the network by the GossipSub layer of the Eth2.0 network. Finally, we also show that FileCoin uses GossipSub configurations that violate the GossipSub specification.

- **Ethics.** We submitted responsible disclosures to the GossipSub developers at Protocol Labs, as well as the Ethereum Foundation. Both groups provided feedback, agreeing with our results. The Ethereum Foundation is working on a patch, and notified maintainers of popular Ethereum implementations about the issue. An alternative to waiting for a patch is to use flooding at the cost of increased network consumption, which GossipSub was designed to avoid.

- **Outline.** We present background on gossip protocols and formal methods in Section 2. We describe GossipSub in Section 3. We show our ACL2s formalization of GossipSub, the security properties and attack generation in Section 4, and experimental results in Section 5. We overview related work in Section 6 and conclude in Section 7.

2. Background

We provide background on gossip protocols and attacks against them. We then overview previous work applying formal methods to gossip protocols and describe our approach.

2.1. Gossip Protocols and Misbehaving Nodes

P2P systems construct logical networks without requiring peers to maintain information about the global topology of the P2P network. Peers maintain information about their neighbors, peers they can communicate with directly. Communication between peers that are not neighbors is achieved through gossip protocols that propagate information throughout the network by having each peer disseminate information using its local information about other peers. The most basic gossip protocol is known as *flooding* where each peer naively propagates information to all of its neighbors. Of all approaches to disseminating data in a system with malicious nodes that do not correctly forward data, flooding is the most resilient because it sends data on all possible paths. However, flooding is not scalable due to high bandwidth cost, so other approaches with a better balance

between information propagation and bandwidth cost were proposed. One such example is GossipSub.

P2P systems are engineered to deal with not only system dynamics, such as *churn* where peers join and leave the system as desired, peer failures, and network partitions, but also with attacks from misbehaving peers. Such misbehaving peers can be Sybils, or peers that have been compromised by an attacker. In Sybil attacks, a single attacker orchestrates a multitude of identities (called Sybils) to gain unfair influence over the network [18]. In the absence of a central entity for authentication, defenses against Sybils have focused on examining the network topology and looking for anomalies in this graph. In many real systems such solutions requiring global information are impossible, so more local approaches were proposed, *e.g.*, examining the geo-location of IP addresses (this is not a robust defense, considering how easy is to fake IP addresses), or imposing a network topology that constrains an attacker in what identity they can assume in the system. More recently, some systems focused on the functionality of the application itself and made acting as part of the system incur a computation cost (proof-of-work) – as in Bitcoin, where the constraint is computational, and the correctness of the system relies on assumptions about the computational power available to the attacker and the theoretical complexity of the proof-of-work problem. Recent solutions against Sybil attacks were also proposed for social [19], [20], [21], [22] and vehicular [23], [24] networks, where such attacks are also prevalent.

In gossip protocols, the main impact that misbehaving nodes can have is to disrupt communication by dropping or delaying application data or P2P control messages. Gossip protocols that do not use flooding are more vulnerable to attacks from misbehaving nodes as a small number of nodes can disrupt communication, potentially across the entire system. Since peers are expected to deliver application and control messages, maintain the logical network, and signal operational status, a potential defense against misbehaving nodes is to observe peer behavior and use this information to decide to whom new messages should be forwarded.

2.2. Formal Methods for Gossip Protocols

Formal methods (FM) refer to tools and techniques used to specify and reason about systems with mathematical rigor, using logic. Mathematical specifications of systems can be used to formalize all possible system behaviors as well as properties that systems are expected to satisfy. There are many formal techniques for either proving or disproving that systems, or abstractions thereof, satisfy properties. One class of tools, which includes interactive theorem proving, has high expressiveness, allowing one to specify arbitrary, Turing-complete computational systems and properties. Such tools require well-trained human proof engineers with the ability to interact with the tools in order to obtain formal, mechanically-checked proofs. Another class of tools includes decision procedures for restricted fragments of logic, such as temporal logic. Such tools can be used in a more automated way, although they impose

severe limitations on what can be expressed, *e.g.*, properties over integers with only addition, multiplication and equality, due to their undecidability, are already too expressive to be handled by such tools. Examples of decision procedures include automated theorem provers (such as SMT solvers), model checkers, type checkers and static analyses based on abstract interpretation. Using decision procedures effectively often requires reasoning about expressible abstractions of systems, *e.g.*, abstractions based on types or abstract domains or finite-state abstractions [25], [26].

Prior works applying FM to gossip protocols. Multiple prior works proposed general frameworks for verifying pub/sub protocols, but did not consider security or attacks [10], [11], [27]. In a similar vein, Díaz et. al. model-checked a pub/sub architecture for discoverable web services [13]. Dagand et. al. created OPIS, an OCAML framework for building and reasoning about distributed systems, which included a formal framework for defining gossip protocols. Systems built in their tool can be evaluated using the ISABELLE and COQ theorem provers, or using a model checker and simulator of their own design [28]. Bakhshi et. al. surveyed formal methods techniques that could be applied to gossip protocols [29]. A subsequent work built a pen and paper framework for modeling dissemination in gossip protocols by abstracting their behaviors to just pair-wise interactions [30]. Van Ditmarsch et. al. built an epistemic model checker as part of their framework to improve dissemination in gossip protocols [31]. Two prior works studied gossip protocols using probabilistic model checking [12], [14]. Because of state-space explosion, all the model checking papers had to abstract the protocol logic and/or restrict the properties they studied.

Gossip protocols that were previously studied with formal methods used (partial or total) flooding, so that even if misbehaving nodes decide not to forward data, in a sufficiently well connected network, every message will eventually reach every node. Thus, prior works that applied formal methods to gossip protocols focused on proving that all messages were eventually fully disseminated. This approach does not apply for protocols that balance bandwidth overhead with data delivery (such as GossipSub) as they have different specifications and safety properties.

2.3. Our Approach

We study GossipSub, a gossip service that addresses attacks from misbehaving nodes by using a score function to capture peer behavior combined with defense mechanisms that adaptively modify the local network topology.

We use interactive theorem proving because methods based on decision procedures, such as model checking, cannot be used to study the actual infinite-state protocol. While theorem proving requires more human effort, it allows us to provide a formal, executable model of the protocol, to formalize properties that the protocol should satisfy and to prove or disprove such properties for various configurations.

Note that in an interactive theorem-prover, we can articulate any Turing machine (including infinite-state sys-

tems) and any predicate logic property about it, but, the prover might not be able to prove or disprove the property without expert human guidance. In contrast, symbolic model-checkers like SPIN [32], TAMARIN [26], and PROVERIF [25], only support restricted models and logics that lend themselves to automated analysis. For example, SPIN supports finite Kripke Structures and Linear Temporal Logic properties, while PROVERIF supports an applied π -calculus with cryptographic primitives, and properties relating to secrecy, authentication, and process-equivalence.

We use the ACL2 Sedan (ACL2s) [33], [34] theorem prover, which extends ACL2 [35], [36] with an advanced data definition framework (*Defdata*) [37], the *cgen* framework for automatic counterexample generation [38], [39], [40], a powerful termination analysis based on calling-context graphs [41] and ordinals [42], [43], [44], a property-based modeling/analysis framework, and IDE support.

In contrast to other theorem-provers, ACL2s allows us to build an *executable* model using the Defdata framework, and then generate attack specifications against that model using the *cgen* framework – which rivals or out-performs other state of the art tools such as ALLOY or LEAN’s *hammer* tactic [39]. And since ACL2s is LISP-based, the model is more expressive and readable to the average software engineer than, *e.g.*, COQ or LEAN code. Reasoning in ACL2s is facilitated by a collection of proof methods including rewriting, numerous decision procedures, and a large collection of libraries. Thus, we model GossipSub by implementing it as a fully functional computer program in ACL2s, and then we reason about it, all in the same system.

Our model can be used for large-scale simulations, as a formal specification for GossipSub, and also as a reference with which to prove or disprove properties of GossipSub.

To the best of our knowledge, we are the first to fully formalize an executable model of a non-trivial gossip protocol not entirely based on flooding, and then automatically prove and disprove properties about that protocol.

3. GossipSub

In this section, we overview the design of GossipSub, provide more details about the score function, and describe how GossipSub was validated by its designers. Below we use the words *node* and *peer* interchangeably and we refer to members of a GossipSub peer’s local peer-list as *neighbors*.

3.1. Overview

The basic approach to quickly disseminate information in a P2P system is where every peer forwards every new message to all of its neighbors, *flooding* the network. Because data travels on all possible paths in the P2P network, this approach is the most resilient to attacks from misbehaving nodes that do not correctly forward messages. However, all this dissemination incurs a significant bandwidth cost.

GossipSub was proposed to decrease this bandwidth cost by using a mechanism called *lazy pull* to balance

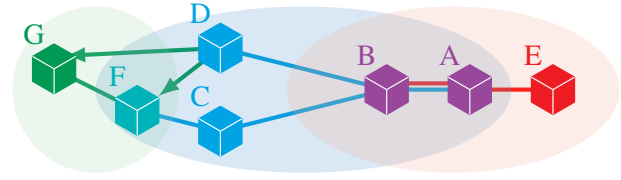


Figure 1. Example GossipSub network where cubes denote peers, each ellipse contains all the peers that subscribe to a particular (colored) topic, peers are colored according to their subscriptions, an undirected edge between two peers denotes a mesh connection over the corresponding colored topic, and a directed edge from a peer p to a peer q over a topic c denotes that q is in the fanout $p.F(c)$. For simplicity, not all realistic connections are shown. Note, mesh connections are reciprocal and occur in a subscribed topic, while fanout connections are directed and begin outside a subscribed topic. Also, two peers may subscribe to the same topic without being mesh-neighbors on that topic, for example peers D and C .

speed of message dissemination with bandwidth consumption. Specifically, the metadata of messages are periodically disseminated in a controlled manner, whereas full messages are sent upon request. GossipSub partitions data in *topics* to which peers can subscribe or unsubscribe as in pub/sub systems. For each topic, nodes create and maintain a dissemination topology. If the node subscribes to the topic, the topology is called a *peer mesh*, otherwise it is a *peer fanout*. A peer’s meshes and fanouts are subsets of its peer-list, and the mesh and fanout for a given topic are disjoint.

Unfortunately, by avoiding flooding, GossipSub becomes less resilient to attacks against communication from malicious nodes. In such attacks, malicious nodes either do not forward data, or do so on a delayed schedule. To address this, GossipSub uses a set of defense mechanisms based on a *score* that is locally maintained by each peer for each of its neighbors, capturing their observable positive and negative behaviors. A positive/negative score is intended to indicate *good/bad* behavior, respectively. Peers re-calculate scores periodically and use them to adjust their meshes and fanouts, determining to whom they will send data.

Message dissemination. Each peer p is initialized with a mutable list of other peers and their subscriptions – and these listed peers are the neighbors of p . Over time new neighbors can join and existing neighbors can leave the network. Peers and their neighbors communicate over topics. We denote by $p.T$ the set of topics p is aware of and by $p.S$ the set of topics that peer p subscribes to. Both sets are mutable and we define $p.U = p.T \setminus p.S$ to be the topics to which p does not subscribe. Each peer p communicates full-messages on a subscribed topic s only to a subset of its s -subscribing neighbors, denoted $p.M(s)$ and called a *mesh*. Likewise, a peer p communicates full-messages on a topic u to which it does not subscribe only to a subset of its u -subscribing peers $p.F(u)$, called a *fanout*. Meshes and fanouts are mutable, as are subscriptions, meaning a peer might unsubscribe from a topic, delete its corresponding mesh, and build a corresponding fanout; or vice versa.

Metadata dissemination. Metadata about recently received mesh and fanout messages are periodically broadcast

to a newly randomly selected subset of peers, allowing meta-data to disseminate quickly with low overhead, so peers can request specific messages from whoever has that content.

GossipSub Threat Model. The GossipSub developers assume the following (implicitly or explicitly): applications frequently inject new messages for dissemination; every network peer runs the same application with the same configuration; and the goal of honest peers is the rapid, on-demand, total dissemination of uncorrupted data, with low overhead. Honest peers follow the GossipSub state machine, responding to requests and forwarding data as quickly as possible, whereas malicious ones can perform any of the following network actions: sending valid or invalid messages, forwarding data with any amount of delay, dropping data to be forwarded, or sending any GossipSub control message at any time. The goal of the malicious peers is to misbehave by dropping or delaying data forwarding or by sending invalid messages, without their malicious actions being detected.

Defense mechanisms. GossipSub restricts the mesh and fanout peers to only those who appear less likely to be malicious nodes. This determination is made based on a score function that each peer computes about each of its neighbors. The score function is used to remove (prune) and add (graft) peers, *e.g.*, in Fig.1, if peer A penalizes B for sending invalid messages, causing B 's score to become negative, then B will be pruned from A 's meshes.

Several aspects of the score function are crucial to the correctness of the defense mechanisms, such as: defining good/bad behavior, specifying parameters that weigh good/bad behavior in the score function, choosing thresholds for determinations about behaviors, and selecting appropriate weights, decays, and caps used by the score function.

3.2. The Score Function

Peer behavior. The goal of the score function is to measure good and bad behaviors of peers. At a high level, the score function takes as input a list of counters, that count specific good and bad behaviors of the peer being scored. Some of these counters are indexed by topic (and are called topic-specific) while others are not (and are called global). For example, the invalid message deliveries counter for a neighbor q on a topic t counts the number of invalid message deliveries the scoring peer p has received from q on the topic t . Some of the counters decay periodically, so that recent events influence the score more than historical ones, and the degree to which each counter decays is specific to the counter (and topic, if the counter is topic-specific) and configured by the application. These counters are used to compute indicators that measure good and bad behaviors of the peer being scored, such as, the indicator $P_6(t)$ which equals the invalid message deliveries, squared. Such indicators might be non-continuous, and might take as input more than one counter. Like the counters, some of the indicators are topic-specific while others are global.

The score function multiplies the indicators that measure good behaviors with positive weights, and the indicators that measure bad behaviors with negative weights, and then

combines the resulting values using weighted and capped summations. A positive score is considered good, otherwise it is considered bad.

There are only two good behaviors in the GossipSub score function: staying in the mesh for a long time, and delivering messages on a subscribed topic within some threshold time window. The score function defines five possible bad behaviors: delivering messages on a subscribed topic at an insufficiently high rate, failing to quickly deliver a requested message on a subscribed topic, sending an invalid message (*e.g.*, one that does not type-check), being co-located on the same IP address as another peer, or trying to re-graft during the backoff period after being pruned. There is also an application-specific indicator that can be positive or negative, and which is totally controlled by the application. This allows the application to reward or penalize behaviors that it considers to be good or bad, respectively.

We denote the per-topic indicators with the shorthand $P_j(t)$, where j identifies the indicator and t is the topic. This is the notation used by the GossipSub developers and should not be confused with our lower-case notation for peers. To be clear, when we discuss how the peer p uses the per-topic indicator $P_j(t)$ while scoring the peer q , we are really referring to $p.P_{j,q}(t)$, an indicator that is local to the scoring peer p and indexed by the indicator name j , peer being scored q , and topic t . Likewise, when we discuss how the peer p uses the global indicator P_k while scoring the peer q , we are really referring to $p.P_{k,q}$, an indicator that is local to p and indexed by the indicator name k and peer being scored q . For each topic t , there are five topic-specific indicators: $P_1(t)$, $P_2(t)$, $P_3(t)$, $P_{3b}(t)$, and $P_4(t)$. The global indicators are P_5 , P_6 , and P_7 . All indicators are weighted in the score function with corresponding weights.

Per-Topic Indicators. We describe them below.

$P_1(t)$: *Time in Mesh.* It is the amount of time quanta a peer has continuously been a member of $p.M(t)$ capped by a small positive constant time in mesh cap that is configured topic-by-topic by the application. $P_1(t)$ is multiplied with a small positive constant configurable topical weight $w_1(t)$.

$P_2(t)$: *First Message Deliveries.* The number of messages on the topic t for which the peer was one of our first deliverers (as measured by a constant, topic-specific, application-configured temporal threshold), multiplied by the topic-specific first message deliveries decay, rounded down to zero if it falls below `DecayToZero`, and capped above by the positive corresponding first message deliveries cap. $P_2(t)$ is multiplied with a positive weight $w_2(t)$.

$P_3(t)$: *Mesh Message Delivery Rate.* Let the mesh message deliveries on a topic t be the number of messages delivered to p by q on t , multiplied at each time-step by a topic-specific mesh message deliveries decay, and rounded to 0 if it falls below `DecayToZero`. If the deliveries exceed a topical mesh message deliveries threshold, or if $P_1(t)$ does not exceed the topical mesh message deliveries activation, then $P_3(t)$ is set to 0. Else, $P_3(t)$ is the difference squared. $P_3(t)$ is multiplied by a weight $w_3(t) < 0$.

$P_{3b}(t)$: *Mesh Message Delivery Failures.* Counts a random subset of the message delivery failures the scoring

peer p observed from the peer q on the topic t , multiplied at each time-step by the topical mesh failure penalty decay, and rounded down to zero if it falls below `DecayToZero`. A failure occurs when q declares that it has a message x , the scoring peer p responds by requesting the message x , and then q fails to respond with x in a timely fashion. Whenever p prunes the q , p increments $P_{3b}(t)$ by $P_3(t)$. Ideally, this punishes pruned peers so that they cannot quickly re-graft. $P_{3b}(t)$ is multiplied with a negative weight $w_{3b}(t)$.

$P_4(t)$: *Invalid Messages*. The number of invalid messages delivered to the scoring peer p by the peer q on the topic t , multiplied at each time-step by a topical invalid message deliveries decay and rounded to 0 if it falls below `DecayToZero`. Messages that does not type-check or that are marked invalid by the application are considered invalid. $P_4(t)$ is multiplied with a negative weight $w_4(t)$.

Global Indicators. We describe them below.

P_5 : *Application-Specific Score*. This is the score component assigned to the peer by the application itself. It is a real value that is multiplied in the score function by a positive weight w_5 , so that the application can, for example, signal misbehavior with a negative score, or gate peers before an application-specific handshake is completed.

P_6 : *IP Colocation Factor*. Let IP colocation factor refer to the number of neighbors of p using the same IP address as the peer q . If the IP colocation factor is not more than the IP colocation factor threshold, then P_6 is set to 0. Else, P_6 is set to the square of the difference. In the score function, P_6 is multiplied with a negative weight w_6 . This indicator can be used to detect Sybils *iff* the Sybils are IP co-located.

P_7 : *Behavioral Penalty*. Let the behavioral penalty be initialized at 0, incremented by the scoring peer p whenever q tries to re-graft less than `PruneBackoff` time after being pruned or has a mesh message delivery failure, rounded down to 0 if it falls below `DecayToZero`, and multiplied by the behavior penalty decay at each heartbeat maintenance event. Let `excess` equal the behavior penalty minus the behavior penalty threshold. Then $P_7 = \text{excess}^2$ if the penalty exceeds the threshold, else 0. In the score function, P_7 is multiplied with a negative weight w_7 .

Configuring the Score Function. The GossipSub prose specification states that $w_1(t)$ should be a “small positive”; $w_2(t)$ and w_5 should be “positive”; $w_3(t)$, $w_{3b}(t)$, $w_4(t)$, w_6 , and w_7 should be “negative”; `DecayToZero` should be “close to 0.0”, the decay parameters should all be “in (0.0, 1.0)”, time in mesh caps should be a “small positive value”, first message deliveries caps should be greater than or equal to the corresponding mesh message deliveries thresholds; IP colocation factor should be “at least 1”, and mesh message deliveries thresholds should be “positive” and depend on the “expected message rate for topic.” This final dependency is unexplained, nor is guidance is provided for the topic weight $\text{tw}(t)$ or topical mesh message deliveries activations [6].

The score function requires additional peer-specific constants which can be configured by the application. The first is a non-negative constant `TopicCap`, which is used to define the function $\text{TC}(x) = \min(x, \text{TopicCap})$ if `TopicCap` $\neq 0$ else x . This function limits the contribution

of topic-specific behaviors to the final score. Second, for each topic $t \in p.T$, the score function requires a positive constant $\text{tw}(t)$ called the *topic weight* of t controlling the relative influence of topic-specific behaviors to the overall score. The specification does not give guidance on how to configure the `TopicCap` or topic weights.

Note, the GossipSub specification does not require peers to configure their score functions the same way. In the case studies we considered (FileCoin and Eth2.0), nodes use identical configurations. We enumerate the score function configuration variables in Table 8 in the Appendix.

The Score Function. Recall that $p.T$ denotes the set of all topics the peer p knows about, including those it does not subscribe to. The GossipSub specification [6] defines the score computed by a peer p for a peer q as follows.

$$\begin{aligned} \text{score}(q) = & \text{TC}\left(\sum_{t \in p.T} \text{tw}(t) \left(\sum_{i \in \{1,2,3,3b,4\}} w_i(t) P_i(t)\right)\right) \\ & + w_5 P_5 + w_6 P_6 + w_7 P_7 \end{aligned}$$

3.3. Attack Mitigation Using Score Function

GossipSub leverages heuristic defense mechanisms based on two caches, *mcache* and *seen*. The *mcache* stores full messages and their identifiers, enabling lazy pull. To avoid memory overflow, the *mcache* is partitioned into lists called *history windows*. Periodically, a new history window containing the most recently sent or received new messages is pushed to the cache, and if the cache size exceeds a parameter `McacheLen`, then the oldest history window is deleted. The *seen* cache is a timed cache, but only tracks message identifiers and is used to avoid infinite forwarding loops. The defense mechanisms and their caches are tuned by a set of parameters, detailed in the Appendix in Table 8. Note that the GossipSub specification imposes numerous hard constraints on these parameters, *e.g.*, w_{3b} must be negative, and yet in the Golang implementation these constraints are made soft, *e.g.*, the implementation allows $w_{3b} = 0$. In many cases the soft constraints allow the application to disable important functionality in the protocol. For example, we discuss later how FileCoin sets $w_3 = w_{3b} = 0$, and this actually disables the application from detecting nodes who slow the network by delaying or blocking forwarding. This configuration violates the written specification.

Pruning. This mechanism is *controlled mesh (and fanout) maintenance*. Peers whose scores fall below zero are pruned from the mesh and fanout at every heartbeat maintenance event (by default, every second). This mechanism aims to remove peers who delay or block forwarded data, send invalid messages, or otherwise misbehave (as signaled by a negative score). Afterword, the peer checks if it is oversubscribed, *i.e.*, if its mesh degree exceeds the threshold `Dhi`. If so, it removes neighbors from the mesh starting from those with the lowest score and moving up until the mesh size equals the threshold `Dout`.

Opportunistic grafting. The goal of this mechanism is to add to the mesh- and fanout-peer sets, peers who behave properly (and thus accumulate positive score). If the

median score of fellow mesh peers is below the threshold `OpportunisticGraftThreshold`, then above-median scoring neighbors are opportunistically grafted.

Backoff on prune. This mechanism consists of adding a backoff period `PruneBackoff` after pruning during which the pruned peer is forbidden from re-grafting, ensuring that once malicious nodes are detected and pruned they cannot quickly rejoin. There is a similar period `UnsubscribeBackoff` for unsubscribing from a topic, before which a peer should not re-subscribe, but the specification does not penalize peers for resubscribing too quickly.

Flood publishing. To limit the impact of attacks when a message is first sent, GossipSub includes an optional *flood publishing* feature, where each peer sends every newly published message to all topic-subscribed neighbors whose scores exceed the positive `PublishThreshold`. This ensures that a new message is disseminated to properly behaving peers (who presumably have high scores) even when the network is saturated with malicious nodes.

Adaptive gossip dissemination. In GossipSub’s lazy pull mechanism, peers adaptively update the number of neighbors to whom they emit topical gossip. First, the gossiping peer selects all the mesh-neighbors whose scores exceed `GossipThreshold`. While this set is larger than `Dlazy`, the gossiping peer randomly removes $(1 - \text{GossipFactor})\%$ of neighbors from the set. Then it emits gossip to whoever remains in the set. The feature is designed to achieve some benefits of flood publishing without all the bandwidth cost, to combat malicious nodes.

3.4. Previous Attack Analysis of GossipSub

The Protocol Labs ResNetLab and software audit firm Least Authority tested GossipSub against a list of specific pre-programmed attack scenarios (*e.g.*, malicious peers saturate a network and simultaneously stop transmitting data) using a network emulator called `TESTGROUND` [45].

In each simulation, the attacker goal was to degrade overall network performance, *i.e.*, to increase average dissemination time and loss. They used simplified configurations with only one topic, and their configurations did not exactly match those currently used by FileCoin and Eth2.0, as these values have since been updated (partially as a consequence of their findings). They simulated 1,000 honest peers and 4,000 malicious nodes, allowing each malicious node to establish 100 connections, and each honest peer to establish 20 connections. They also tested the BitCoin and Eth1.0 gossip protocols, as well as GossipSub without the defense mechanisms. Their success criteria for the defense mechanisms were that (a) messages were fully disseminated in $< 6s$ for FileCoin or $< 12s$ for Eth2.0, and (b) the loss rate was low. (They did not specify what they considered to be low.) They found experimentally that all the attacks failed against GossipSub, and the defense mechanisms made GossipSub more resilient to malicious nodes attacks than the other tested protocols [8]. Separate from simulation testing, Least Authority also audited the Golang implementation and provided recommendations for improvement [9].

4. ACL2s GossipSub Model

In this section, we describe our computational model of GossipSub in ACL2s. We give a high-level overview, discuss how we validated it against the Golang implementation, then explain our data structures to represent network states, and how we implement transition functions for states. We then explain how we weave together states and events produced by these transition functions into entire network traces.

Recall that a core feature of GossipSub is how it uses heuristic defense mechanisms to promote well-behaved peers and demote poorly-behaved ones, *e.g.*, by grafting or flood-publishing to high-scoring peers while pruning peers with negative scores. We formalize this feature as the *fundamental property* of the defense mechanisms. Then, we formalize four novel correctness properties for the score function that are necessary for this fundamental property to hold. Note, these properties do not comprehensively cover all gossip protocols. The properties for a gossip protocol depend on the application, *e.g.*, one might prioritize dissemination speed, another reliability. We focused on the most general properties of the score function, which must hold in order for the fundamental property of the defense mechanisms to hold. We solicited security properties from the GossipSub developers and the Ethereum Foundation; both endorsed our properties but did not provide more.

We test our four properties using the counterexample generation facility provided by ACL2s, and find counterexamples to three of them. We use ACL2s to prove the fourth. Finally, we synthesize (generate) new traces that lead to such counterexamples, and show that these traces also violate the fundamental property of the defense mechanisms. The sequences of actions taken by adversary peers in these traces constitute attacks against GossipSub. In these attacks, adversaries misbehave by not forwarding data, thereby slowing down the entire network. They do so in a stealthy manner so that they are never demoted by the honest peers.

4.1. Overview

We used ACL2s to model and reason about GossipSub. We give a brief overview of how such code is written in the Appendix. Our model of GossipSub captures every aspect of GossipSub given in the prose specification [5], [6]: control messages, lazy pull, the internal peer state, meshes and fanouts, the score function, the defense mechanisms, etc., including every detail in Section 3, and other details from the prose specification that we omitted for readability and space. We made only the following abstractions: (1) the message payload is a list consisting of a message-id and a message, both represented by naturals; (2) the lower level transport protocols are represented using a partial ordering on message send and receive events; (3) establishing a network connection is represented by explicit `CONNECT` events; and (4) every sent message is received *i.e.*, the network is assumed to be free from any transient conditions, which also implies that (5) in-flight messages cannot be corrupted.

Despite the fact that our model is fully faithful to the written specification of GossipSub, and is itself an executable program, it is not a network library; it cannot be used in place of the Golang implementation; and it is not intended to be used as such. It is simply an in-memory model of GossipSub, which also happens to be a mathematical object that we can reason about using ACL2s. We aim to derive a GossipSub network-library from our model in the future.

4.2. Validating Our Model

Our model allows us to execute and reason about any component of the peer logic in isolation, the entire program for a peer, or even an entire network of peers. We developed our model in consultation with the GossipSub authors, who asked us to study the score function in their protocol. We validated our model in multiple ways. We implemented all the tests from their Golang code as tests or theorems in our model; instrumented the Golang code to print traces, which we type-checked with our model; and generated counterexamples in our model, which we translated into (passing) Golang unit-tests. Through these exercises, we found ambiguities in the prose and places where the code and prose disagreed. We reported errors to the developers and followed their advice to resolve ambiguities.

Discrepancies. In the specification, but not the implementation, the activation window is used when calculating P_3 and P_{3b} . In the implementation, P_3 is updated periodically, but in the specification it is updated only when the peer is pruned. Components of the score function can be disabled in the implementation but not the specification. We allowed disabling in our model because FileCoin uses this feature, but otherwise we followed the English specification.

Our ACL2s model is fully verified and contains 5877 lines of code, 189 definitions and 155 explicit theorems and properties. Every function definition involves proofs that are not included in the explicit theorem total, *e.g.*, of termination, that the input contracts imply the output contracts, etc. Most model development effort was devoted to translating the prose specification into a mathematical form, comparing it to the Golang implementation, and translating tests from Golang into ACL2s. Once we had the model, it was fairly straightforward to write properties to test for counterexamples. Note, developing and admitting all our functions, with full proofs of termination and contract guarantees, required effort comparable to developing and unit-testing the corresponding functions in a traditional implementation.

4.3. Model State and Transitions

We define the state of a peer using the `peer-state` type and of multiple peers using the `Group` type.

```
1 (defdata peer-state
2   (record (nts . nbr-topic-state)
3           (mst . msgs-state)
4           (nbr-tctrs . pt-tctrs-map)
5           (nbr-gctrs . p-gctrs-map)
6           (nbr-scores . peer-rational-map)))
```

```
7
8 (defdata group (alistof peer-id peer-state))
```

Each `peer-state` for a peer p is a record consisting of five components: (1) `nts` of type `nbr-topic-state`: a state containing p 's neighbors' subscriptions, $p.M$, $p.F$, and a map storing p 's last publication time in each topic (used for fanout maintenance); (2) `mst` of type `msgs-state`: a state containing a cache of full messages received, a map from recently seen message ids to their age (updated at every heart beat maintenance event), history windows and the completion status and count of both sent and received requests for messages; (3) `nbr-tctrs` of type `pt-tctrs-map`: a total map from each pair of neighbor q and $t \in p.T$ to `topic-counters`; (4) `nbr-gctrs` of type `p-gctrs-map`: a total map from each neighbor q to a list of global counters $\langle P_i \mid i \in \{5, 6, 7\} \rangle$; and (5) `nbr-scores` of type `peer-rational-map`: a total map from neighbors of p to their cached scores, which gets updated at every local heart beat maintenance. All scores and counters default to 0. Peers are identified using unique `peer-ids`, and a `Group` is simply a finite map (an association list) from `peer-id` to `peer-state`.

Notice that all `peer-states` in a `Group` are simultaneous: a `Group` captures an instantaneous snapshot of the GossipSub network. However, the peers themselves do not have access to a global clock. They are only aware of the partial ordering of events they can locally infer. A peer does not have access to any other `peer-state` besides its own, nor to any data that is not locally observable.

We define GossipSub network events using a type called `event`. An event occurs when a peer sends or receives a control message, joins or leaves a topic, goes through local heart beat maintenance event, forwards a message from the application layer or establishes a connection with another peer. Events where messages are sent or received carry the identity of two peers: sender and receiver. Every event carries the identity of the peer who triggered it, as its first element. The `event` type is described using BNF below, where `pid` is a peer identifier, `vr` is `SND` or `RCV`, `top` is any of the topics in the application running on GossipSub, and `msg` is any payload including control- or full-messages.

```
event ::= pid vr pid msg | pid JOIN top
        | pid CONNECT top | pid HBM top
        | pid APP top msg | pid LEAVE pid
```

Every application that runs on top of GossipSub tunes weights and parameters in order to define the score function such as $tw(s)$, $w_3(s)$, the mesh message deliveries decay on s , etc., as detailed in Section 3.2. We store topic-specific weights and parameters in a map from topics to corresponding weights and parameters, which we call `twp`. Note that `twp` is a constant specific to the application instance we are simulating. The `peer-state` transition function is called `ps-trx` (illustrated in Figure 4). It takes as input a `peer-state` called `ps`, a `twp`, and an event called `evnt`. We assume an oracle for making non-deterministic decisions in the model. In simulation runs the oracle can be replaced by a pseudo-random number generator, for convenience.

ps-trx outputs the peer's new peer-state, as well as the events it emits during the transition.

The Group transition function is called gs-trx and takes as input a Group called gp, a twp, and a work-list of events initialized with init-evnts. gs-trx assumes the same oracle as ps-trx. The peer who triggered the first event in a work list of events, transitions on that event using the peer-state transition function. A new Group is then generated where the peer's old peer-state is replaced with its new one. Along the way, it also computes any emitted events which are spliced with the events work list. The function is illustrated in Figure 3.

4.4. GossipSub Score Function Properties

The most important features of GossipSub are lower bandwidth consumption via lazy pull, and security against malicious peers via heuristic defense mechanisms. The fundamental idea of the heuristic defense mechanisms is that honest peers can be distinguished from malicious ones based on their observable behaviors, and thus, the overall network can be made more secure and performant if every honest peer promotes their well-behaving neighbors and demotes poorly-behaving ones. We formalize this requirement as follows, where poor and good behavior are defined by the good and bad behavior counters given in Section 3.

Fundamental Property of the Defense Mechanisms.

Peers who behave poorly will be demoted by their neighbors. Peers who behave better-than-average will be promoted by their neighbors. Promotion/demotion is entirely based on peer behavior.

Studying this fundamental property directly is difficult due to the massive search-space of possible attack vectors. Hence, we decompose the problem by proposing four novel security properties for the score function without which the *fundamental property* cannot possibly hold. We choose these properties such that a reasonable software developer might infer that they are true about GossipSub, based on textual descriptions of the protocol by the GossipSub developers. We encode these properties in ACL2s as predicates over data types defined above. The properties are defined in an app-specific manner, i.e., each property is parameterized by a fixed app-specific twp. Importantly, all four properties are independent of the number of peers, percentage of malicious peers, or network topology. They depend only on the topic, app-specific parameters, and performance counters of the peer being scored. Note, there is no preexisting formal or informal specification of what it means for GossipSub to behave correctly, which is why we created a fundamental property for the defense mechanism and four implied formal properties for the score function based on our interpretation of the textual specification.

The GossipSub developers write [8]: “The score function is used as a performance monitoring mechanism to identify and remove poorly performing or misbehaving nodes from the mesh.” Since meshes are topic specific, a natural question to ask is, does the score function identify poorly performing nodes in each topic? GossipSub peers can subscribe

to, and forward messages over several topics, and hence a single peer can be a member of several topic meshes. Since peers that accumulate a non-positive score are the ones that ultimately get pruned, we claim that continuously achieving a non-positive score in any topic should eventually result in a non-positive overall score, leading the peer to be demoted via pruning. If this is not true, then neither is the fundamental property, since one of the defense mechanisms is that poorly-behaving peers get opportunistically pruned. For example, in Fig. 1, if peer B throttles deliveries in topic blue to A, then we want to know if A will assign a negative score to B and thus prune it during mesh maintenance. We formalize this liveness property below.

Property 1. *If a peer's score relating to its performance in any topic is continuously non-positive, then the peer's overall score should eventually be non-positive:*

$$\forall q, t :: \langle \mathbf{G}(\text{score}(q) \text{ for topic } t \leq 0) \Rightarrow \mathbf{F}(\text{score}(q) \leq 0) \rangle$$

where $\text{score}(q)$ for topic t is defined below.

$$(\text{tw}(t)(\sum_{i \in \{1,2,3,3b,4\}} w_i(t)P_i(t)))$$

The GossipSub developers write that peers “that misbehave are penalized with negative score.” [8] This feature is important, because the opportunistic grafting and mesh and fanout maintenance defense mechanisms of GossipSub assure that over time a peer disconnects from neighbors who have negative or below-average scores and connects to those who have positive scores. So, if a peer could misbehave in a specific topic, without getting penalized with a negative score, then these defense mechanisms would be ineffective and the fundamental property would be violated.

The next three properties safety properties. We identify the following as bad-performance metrics indicating misbehavior: deficit in mesh message deliveries (DMMD), invalid message deliveries, and bad behaviors; where: DMMD is the maximum of 0 or the mesh message deliveries threshold minus the mesh message deliveries. Note, these metrics are used in the score function. When discussing peers q, q' in the properties below, we use $P_j, P_i(t)$ to denote indicators of q and $P'_j, P'_i(t)$ to denote indicators of q' .

Property 2. *Increasing bad-performance counters should decrease overall score. Formally, if $P'_i(t)$ differs from $P_i(t)$ only due to an increase in DMMD, invalid message deliveries, or bad behaviors for peer q in topic t , then:*

$$\forall q, t :: \langle (\text{score}(q) \text{ for topic } t) > (\text{score}(q') \text{ for topic } t) \rangle$$

A simplified ACL2s definition of the contraposition to this property in context of Eth2.0 is shown in Figure 5, where *eth-twp* is a twp specific to Eth2.0.

The GossipSub developers write that the role of $P_1(t)$ in the score function is to “boost peers already in the mesh”, and the role of $P_2(t)$ is to “reward peers who act fast on relaying messages.” The app-specific component P_5 “has an arbitrary real value, so that the application can signal misbehavior with a negative score” or good behavior with

a positive score [8]. We define good-performance counters (that measure good behavior) as mesh time, first message deliveries, and mesh message deliveries, and claim that increasing one of these counters should boost the overall score, implying the following analogue to Property 2:

Property 3. *Increasing good-performance counters will not decrease score for a mesh peer that has been in the mesh for a sufficiently long time. Formally, if $P'_i(t)$ differs from $P_i(t)$ only due to increase in mesh time, first message deliveries, or mesh message deliveries for peer q in topic t , and the mesh time is more than the `activationWindow` parameter, then:*

$$\forall q, t :: \langle (\text{score}(q) \text{ for topic } t) \leq (\text{score}(q') \text{ for topic } t) \rangle$$

In GossipSub, “all nodes start equal and build their profile based on their behavior” [8]. Concretely, the score function is referentially transparent: a peer’s score is a function of its behavior alone. Hence the score function is intrinsically unbiased, *i.e.*, if two peers behave identically, then they will achieve identical scores. If this property were not true, then the controlled mesh (and fanout) maintenance and opportunistic grafting defense mechanisms would behave unfairly with respect to peer behavior/misbehavior, potentially violating the fundamental property by making promotion and demotion decisions not based on the good and bad behavior counters. We formalize this in Property 4.

Property 4. *If two peers subscribe to the same topics $\in S$, and achieve identical per-topic params $P_1(t)$, $P_2(t)$, $P_3(t)$, $P_{3b}(t)$, $P_4(t)$, $\forall t \in T$ and identical global params P_5 , P_6 , P_7 ; then they achieve identical scores.*

We present an example of a *good* configuration which satisfies all of our properties in Table 7 in the Appendix.

4.5. Finding Counterexamples

Our model can be used not only to reason about and simulate a GossipSub network, but also, to automatically disprove invalid properties by computing concrete counterexamples. We generate counterexamples with our model using the `cgen` library built into ACL2s, which uses type enumerators, synergistically combined with theorem proving techniques, to generate values for variables within a property such that the hypotheses of the property hold but the implication does not. However, the sample space for counterexamples is huge. Hence, we need to define our own *custom* enumerators for the types of each of these variables, using our intuition about these variables and their types such that hypotheses in our properties are almost always satisfied thereby increasing the chances of discovering values that actually violate our property. A custom enumerator for a type τ is a function from naturals to a value of type τ . When necessary, we craft custom enumerators to quickly find interesting counterexamples. These counterexamples are snapshots of a GossipSub network where the property being tested is violated. Importantly, the snapshot might not be a reachable state of the network. We interpret these

counterexamples as attacker specifications: an attacker can attempt to violate the tested property by guiding the network into satisfying one of these counterexample specifications. Put differently, a counterexample is just a bad network state than attacker would like to achieve, whereas an attack is a sequence of actions performed by one or more attackers that guides an initial network state to a counterexample one. In such attacks, each attacker violates the fundamental property of the defense mechanisms, misbehaving (*e.g.*, by not forwarding data, or by sending invalid messages) while being promoted or without being demoted by its neighbors.

4.6. Generating Attacks

Given a specific `Group` representing the initial global state of the network, and a counterexample `Group` violating a `twp`-parameterized property, we next ask if an attacker could guide the initial `Group` to the counterexample `Group`. The state-space of `traces` is too large to be explored by the counterexample generation facility alone. Instead, we construct an `event` emitter which given the initial `Group` will produce a finite sequence of `events` that eventually leads the initial `Group` to a similar counterexample `Group`. The `event` emitter is sound, but it is not necessarily complete insofar as not every counterexample-class can be reached from every initial `Group`.

An `event` emitter takes in a group consisting of an observer peer and a target peer, list of all topics in which peers in the group communicate and the application specific `twp` to produce a sequence of events. This sequence of events will take our initial group to a state in which the `peer-state` of the observer closely resembles one of the states in our counterexamples. The `event` emitter assumes that all peers in the group behave honestly, except the target peer. Hence, the generated attack does not require any coordination from other misbehaving nodes in order to succeed, and so will succeed in other contexts beyond the one used at synthesis-time. Our `event` emitters can be used to generate attacks by multiple misbehaving nodes easily, by splicing events generated from multiple emitters, each corresponding to a misbehaving node.

Note that our four properties are defined entirely with respect to the score function. They do not take into account the network topology, the number of malicious versus honest peers in the network or their placement, the network dynamics (throughput, churn, etc.), or other variables whatsoever, except for those used by the score function. Consequently, when we prove that a property holds for a given `twp`, our proof shows that the property holds for *all possible GossipSub networks* configured by that `twp`, regardless of their topology, placement and number of malicious peers, etc. Conversely, if we prove that the property does not hold for a given `twp`, then we know we can attack *every single GossipSub network* configured with that `twp`, although the attacks themselves need to be generated using the network state (including topology, number and placement of malicious peers, etc.). In such cases, the attack generated for a large, complicated network will generally just be a scaled

version of the attack generated for an identically configured small one, so attacks on small networks can be seen as *explanations* for the attacks against more realistic ones.

4.7. Evaluating New Application Configurations

One advantage of our model is that it allows developers of new applications using GossipSub to check if their configuration satisfies our security properties. The developer needs to formalize the configuration as a `twp`, instantiate the properties using that `twp`, and then pass the resulting model file into ACL2s. One of three things may happen: ACL2s may prove the properties automatically, outputting `qed`; ACL2s may output counterexamples, as it does for Eth2.0 (in which case the configuration should be debugged using the counterexamples); or ACL2s may fail to do either. In the last scenario, the developer may either tweak the enumerators with which ACL2s generates its counterexamples, or guide the prover using supplemental lemmas, until the properties are disproven or proven. We exemplify how to tweak the enumerators in *scoring-eth2.lisp*, and how to guide the prover using lemmas in *scoring.lisp*. Both files are part of our publicly and anonymously available materials.

5. Experiments

In their emulation testing, the GossipSub developers checked if the defense mechanisms improved the resiliency of GossipSub to specific attacks from misbehaving peers against network performance. We ask a more fundamental question: does the score function upon which the defense mechanisms rely actually measure what it is intended to measure? If the answer is *no*, then there might exist covert attack strategies that are undetectable using the score function with certain GossipSub configurations. This question is articulated via our four formal properties and evaluated on two concrete case studies: Eth2.0, and FileCoin.

5.1. Methodology

As described in Section 4, the ACL2s model is infinite state and faithful to the specification. Any properties we prove hold for all of the infinite instances of the model, with any peers, topology, set of topics, history of events, etc. If a property fails, then there exists a counterexample, but typically infinitely many. When generating counterexamples, we prefer to use minimal networks for human understanding and validation and for generation efficiency. Counterexamples can require simulating minutes of network activity. The amount of possible network activity scales exponentially with the number of peers, as does the simulation cost. To get a sense of this, we tried simulating 10 seconds of network activity on an Eth2.0 network on a 16GB M1 Macbook Air with 100, 200 or 300 honest peers. Running these simulations took 13, 50, and 108 seconds respectively.

Observe that our properties do not depend on the percentage of malicious peers or network topology or size.

Topic	MT	FMD	MMD	IMD	MFP
Blocks	147	194	200	0	0
Agg	42	0	1	0	0
Sub1	141	188	194	0	0
Sub2	42	0	1	0	0
Sub3	135	182	188	0	0

TABLE 1. ETH2.0 PEER TOPIC-COUNTERS VIOLATING PROP. 1

Hence, to find vulnerabilities and eventually exploit them, we can instantiate a small GossipSub Group in our model, using the corresponding app-specific `twp`. The simple Group consists of two honest peers and one attacker, all fully mesh connected on every topic in the `twp`, and allows us to explore the event-space very quickly. Our app-specific `twps` are shown in Tables 4 and 5 and are adapted from open-source implementations of the case studies found on Github. We use ACL2s to try and generate counterexamples with each case study Group, for each property.

If we find counterexamples, then we attempt to generate corresponding attacks, using our event emitter. These attacks are not like those considered in the emulations done by the GossipSub developers. Rather, they describe how a peer can violate one of the properties, *e.g.*, by misbehaving without decreasing its score. Such attacks in a small GossipSub Group can be viewed as the building blocks for crafting stealthier attacks, that evade the defense mechanisms.

5.2. FileCoin Evaluation

We prove that the FileCoin `twp` (Table 4) satisfies all four properties. Unfortunately, the FileCoin `twp` satisfies the properties by violating the GossipSub specification, in that it uses illegal 0-valued weights and thresholds, sacrificing its ability to penalize peers who under-deliver. Since the FileCoin `twp` disables the app-specific score component, the application also cannot signal level misbehavior. Hence the GossipSub layer of FileCoin is (in isolation) less resilient to attacks. (The FileCoin developers inform us that for this reason, FileCoin relies on app-level defenses.)

5.3. Eth2.0 Evaluation

Counterexample Generation for Eth2.0 We next automatically generate Eth2.0 counterexamples to Props. 1 and 2.

• **Property 1.** Our ACL2s formulation 5 of the Prop. 1 is weaker because it is expressed without the temporal operators **F** and **G**. However, Eth2.0 violates Prop. 1 because it has multiple topics and a peer can offset a negative score in one of the (dozens) of subnet aggregator topics by a positive score in all other topics combined, resulting in a positive overall score. A counterexample is shown in Tables 1 and 2. In Table 1, an Eth2.0 peer under-performs in topics AGG and SUB2, *i.e.*, its topic specific counters FMD and MMD are less than the threshold for corresponding deliveries. However, its performance is nominal in the rest of the topics. Table 2 shows indicators computed from topic-counters in Table 1. Note non-zero P_3 and P_{3b} in AGG and SUB2, leading these topics to negatively impact the score.

Topic	P_1	P_2	P_3	P_{3b}	P_4	P_5	P_6	P_7	score
Blocks	147	23	0	0	0	0	0	0	22.21
Agg	42	0	81	81	0	0	0	0	-4.5
Sub1	14.1	24	0	0	0	0	0	0	7.80
Sub2	4.2	0	1	1	0	0	0	0	-25
Sub3	13.5	24	0	0	0	0	0	0	7.78

TABLE 2. ETH2.0 PEER SCORE COMPONENTS VIOLATING PROP. 1. TOTAL SCORE = 8.29.

Topic	MT	FMD	FMD'	MMD	MMD'	IMD	MFP	Score	Score'
Blocks	147	194	3	200	10	0	0	22.21	6.21
Agg	150	194	194	230	230	0	0	13.83	13.83
Sub1	141	188	188	194	194	0	0	7.8	7.8
Sub2	110	180	180	232	232	0	0	7.8	7.8
Sub3	135	182	182	188	188	0	0	7.78	7.78

TABLE 3. PERTURBATIONS IN TOPIC-COUNTERS VALUES FOR AN ETH2.0 PEER WHICH VIOLATE PROPERTY 2. BOTH TOTALS = 32.72.

• **Property 2.** The Eth2.0 $\text{twp TopicCap} = 37.72$. However, the sum of contributions to score from each topic can be well above this limit, violating Prop. 2. A counterexample is shown in Table 3, where, even after perturbations to FMD and MMD give rise to lower values in FMD' and MMD', the overall score remains 32.72. In contrast, FileCoin does not use a TopicCap, and thus satisfies this property.

• **Property 3.** Using property based testing, we show that Eth2.0 satisfies this property for all topics, given that the mesh time is more than the `activationWindow`.

• **Property 4.** This property is proved simply by getting the score function admitted in ACL2. Since ACL2s is a functional language, admitted functions will always have the same output for same inputs, thus validating this property for any possible application running on GossipSub.

Attack Generation for Eth2.0 Next, we generate attacks from our counterexamples. Since our properties are defined for all possible network topologies and number of peers, we consider a simple Group consisting of three peers, forming a mesh in all the Eth2.0 topics. In the counterexamples to Prop. 1 we generated with ACL2s using the Eth2.0 twp , a malicious peer under-performs in one or more topic(s) while performing so well in the others that its cumulative score is positive, and hence it cannot be pruned. Note that an Eth2.0 network can support dozens of these topics, and the topic weight of each scales inversely with the number of them. Consequently, the adversary gets all the benefits of behaving perfectly in the network, while throttling throughput of the network.

When an honest peer publishes a full-message on a mesh-topic, it sends the message to each peer in its topic mesh. In contrast, our adversary uses the gated logic below.

We built a simple event emitter to replay this attack for our simple Group. The emitter takes as input a fully-connected Eth2.0 Group consisting of an honest peer, an honest observer, and an adversary. The Eth2.0 twp initial warm-up period extends above the maximum `ActivationWindow` among all topics during which all peers send at least 10 mesh messages in every topic, per topical heartbeat interval. Ten mesh messages is the maximum mesh message deliveries threshold among all the

Algorithm 1: Misbehaving peer publish

```

1 for peer  $p$  in in mesh for topic  $t$  do
2   if  $\text{targetedTopics.contains}(t) \wedge \text{now}() \geq$ 
      $\text{MaxActivationWindow}$  then
3     if  $\text{rand}(0,10)=0$  then
4       sendRPC( $p,m$ );
5   else
6     sendRPC( $p,m$ );

```

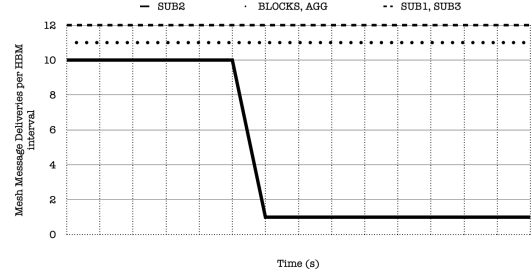


Figure 2. Mesh message deliveries vs. time in mesh for an attacker q as observed by a scoring peer p , over the Eth2.0 topics BLOCKS, AGG, SUB1, SUB2, and SUB3. The attack is generated using our event emitter based on the counterexample shown in Tables 1 and 2. In all topics other than SUB2 the attacker performs consistently well, attaining positive topic-specific score, but in SUB2, after a warm-up period exceeding the maximum activation window among all the topics, it under-performs and soon after achieves a negative topic-specific score. Nevertheless the attacker's score continues to be positive, violating Prop. 1 in perpetuity.

topics, hence it satisfies the threshold for all topics. As illustrated in Fig. 2, right after this warm up period, we begin our attack, with the adversary sending one less than the message deliveries threshold for the targeted topic SUB2 in each heartbeat interval. By under-performing in SUB2, the adversary degrades the performance of the corresponding mesh. Ideally, this under-performance would be flagged by the score function, and the observer would prune the adversary. But this does not occur: the adversary maintains a positive score and remains in all of the observer's meshes. Experimentally, we observed that this attack is not transient. The adversary can forever maintain their positive score, thus making this attack perpetual. The event emitter can be adapted to synthesize such attacks in any Eth2.0 Group.

Scaling to multiple attackers. Based on the insights gleaned from studying counterexamples generated by ACL2s for properties 1-3, we manually crafted a pathological twp (Table 6) with reduced penalties for low `meshMessageDeliveries`. We simulated 1,000 misbehaving peers connected over 5 topics, with a mesh connectivity of 5, for 1,000 seconds of network-simulated time. The misbehaving peers were programmed to never send or forward messages, *i.e.*, to perform a flash attack. In the resulting trace, the misbehaving peers achieved maximal scores from all their neighbors and so would not be pruned – and this situation was permanent, not transient. Simulating the attack took 89s on a 16GB M1 Macbook Air, illustrating how our attacks can be scaled to many malicious peers.

Security implications for Eth2.0 Above, we showed how a single attacker peer could reduce data-forwarding by 90% in a topic without getting pruned. It follows from Prop. 4 that we can scale this strategy to arbitrarily many misbehaving peers, attacking the entire network. In our example, the attacker peer halved their mesh message deliveries in the topic SUB2. According to the Eth2.0 specific parameters shown in Table 5, the average connectivity in SUB2 is eight. Scaling this attack to four misbehaving per honest node, we find that the average number of mesh message deliveries per heartbeat interval drops by 25% below the minimum threshold required by SUB2. Note that this drop is permanent, as we showed in our traces, and has permanently throttles network performance in SUB2. The GossipSub developers attest that there could be ≈ 70 subnet aggregator topics [8]. The attack we describe could scale to multiple subnet aggregator topics at once, so long as the negative contribution to the score from the targeted topics is offset by the positive contribution from the remaining ones. (Note that in Eth2.0 the weight of each subnet aggregator topic is inversely proportional to the number of such topics.) As attackers can actually achieve maximal score while attacking, it follows that they would remain in the honest peers’ meshes. Hence the attack scales from just one malicious peer to a multitude.

6. Related Work

Several prior works applied FM to the automated discovery of attacks against protocols. PROVERIF is an automatic cryptographic protocol verifier based on PROLOG that can automatically generate attacks against confidentiality and privacy [46]. TAMARIN is similar to PROVERIF, and was used to find attacks against the NAXOS [47], 5G AKA [48], the IEEE 802.11 4-way handshake [49], and various other protocols. Von Hippel et. al. reduced the attacker synthesis problem for communication protocols to a Linear Temporal Logic model checking problem, and implemented their approach in an open-source tool called KORG [50], which they applied to TCP and DCCP [51].

A plethora of prior works built and studied executable models of distributed systems using FM tools. Multiple works modeled and proved theorems about CHORD [52], [53]. In one particularly monumental effort, Woo et. al. formally verified 90 properties of the RAFT protocol using VERDI, a tool they built in the COQ proof assistant [54]. Although they did not build an executable model, their framework can generate an executable protocol implementation [55]. Certain high-assurance distributed systems might require formally-verified code at every level of the stack. Such applications could, for instance, be implemented on top of SEL4: a high-performance operating system microkernel that was formally verified against an abstract specification using higher-order logic [56].

Lamport’s modeling language TLA+ [57] and the corresponding TLC model checker [58] have been used to analyze properties of distributed systems including

DISK PAXOS [59], MONGORAFTRECONFIG [60], Byzantine PAXOS [61], SPIRE [62], etc. TLA-style state-machine refinement and Hoare-logic verification are combined in IRONFLEET, which was used to verify a PAXOS-based library and a sharded key-value store [63]. The UNITY [64] computational model, specification language, and proof system was successfully applied to numerous distributed systems including a synchronization scheme for multi-process handshakes [65], Segall’s PIF algorithm [66], distributed sorting algorithms [67], and the Omega Failure Detector [68]. FM was applied to blockchain protocols in multiple works [69], [70], [71]. In industry, Amazon uses a lightweight FM approach to validate new features in their key-value storage node, SHARDSTORE [72].

Many works leverage FM to verify (or find bugs in) communication and network protocols. McMillan and Zuck applied specification-based testing to the QUIC protocol, and found multiple implementation errors, some of which caused vulnerabilities [73]. Wu et. al. formally modeled the Bluetooth stack using PROVERIF, and found five known vulnerabilities and two new ones [74]. Chothia et. al. showed how PROVERIF can be used to verify distance-bounding protocols, *e.g.* those used by MasterCard and NXP [75]. Separately, Chothia modeled the MUTE anonymous file-sharing system using the π -calculus, and proved the system insecure (discovering a novel attack) [76]. Cremers et. al. modeled all handshake modes of TLS 1.3 using TAMARIN, and discovered an unexpected behavior [77]. Although most of these efforts use exactly one of model checking or theorem proving, Manolios et. al. link the two to verify correctness properties of ABP [78].

7. Conclusion

In this paper we rigorously studied GossipSub and its security properties using the ACL2s theorem prover. We created a complete model of the protocol and formalized security properties based on the prose GossipSub specification. We showed that the properties depend on how the score function is configured. Of two well-known applications, FileCoin and Eth2.0, only FileCoin satisfied all properties, but for Eth2.0, we showed that for any network, of any topology and size, we can synthesize attacks where certain peers continuously misbehave by never forwarding topic messages, but are never identified as misbehaving and are never pruned from the network by GossipSub. We ethically disclosed our results to Protocol Labs and the Ethereum Foundation, who agreed with our findings. In addition, the GossipSub developers at Protocol Labs publicly endorsed our model as a formal specification for GossipSub.

Our approach required a formal model of GossipSub, and writing this model required effort comparable to implementing the protocol in a programming language. But compared to an additional implementation, an ACL2s model has many advantages, *e.g.*, providing a formal model that we can reason about with mathematical precision. We did not have to use extensive manual testing because using ACL2s to analyze properties helped us find attacks. Our

work required less manual effort than that expended by the GossipSub developers, and found attacks that they missed. Developers interested in applying our approach to their gossip protocols can build on our formalization, rather than starting from scratch.

Acknowledgements

We sincerely thank Yiannis Psaras, Dimitris Vyzovitis, Nishant Das, Adrian Manning, Max Inden, and others at Protocol Labs and the Ethereum Foundation for their valuable feedback.

References

- [1] “Filecoin: A decentralized storage network,” <https://filecoin.io/filecoin.pdf>, July 2017.
- [2] V. Buterin, “Ethereum: A next-generation smart contract and decentralized application platform,” https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum_Whitepaper_-_Buterin_2014.pdf, 2014, accessed 13 July 2022.
- [3] “Filecoin price,” <https://coinmarketcap.com/currencies/filecoin/>, 11 2022, accessed 20 November 2022.
- [4] “Ethereum price,” <https://coinmarketcap.com/currencies/ethereum/>, 11 2022, accessed 20 November 2022.
- [5] D. Vyzovitis, “GossipSub v1.0: An extensible baseline pubsub protocol,” <https://github.com/libp2p/specs/blob/master/pubsub/gossipsub/gossipsub-v1.0.md>, March 2020, accessed 17 May 2022.
- [6] —, “GossipSub v1.1: Security extensions to improve on attack resilience and bootstrapping,” <https://github.com/libp2p/specs/blob/master/pubsub/gossipsub/gossipsub-v1.1.md>, December 2020, accessed 3 March 2021.
- [7] “GO-LIBP2P-PUBSUB,” <https://github.com/libp2p/go-libp2p-pubsub>.
- [8] D. Vyzovitis, Y. Napora, D. McCormick, D. Dias, and Y. Psaras, “Gossipsub: Attack-resilient message propagation in the filecoin and eth2. 0 networks,” *arXiv preprint arXiv:2007.02754*, 2020.
- [9] D. Lott, “Audit of Gossipsub v1.1 protocol design + implementation for protocol labs,” <https://leastauthority.com/blog/audit-of-gossipsub-v1-1-protocol-design-implementation-for-protocol-labs/>, 2020, accessed 3 March 2022.
- [10] D. Garlan, S. Khersonsky, and J. S. Kim, “Model checking publish-subscribe systems,” in *International spin workshop on model checking of software*. Springer, 2003, pp. 166–180.
- [11] L. Baresi, C. Ghezzi, and L. Mottola, “On accurate automatic verification of publish-subscribe architectures,” in *29th International Conference on Software Engineering (ICSE’07)*. IEEE, 2007, pp. 199–208.
- [12] M. Kwiatkowska, G. Norman, and D. Parker, “Analysis of a gossip protocol in prism,” *ACM SIGMETRICS Performance Evaluation Review*, 2008.
- [13] G. Díaz, M. E. Cambrónero, H. Maciá, and V. Valero, “Model-checking verification of publish-subscribe architectures in web service contexts,” in *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, 2015, pp. 1688–1695.
- [14] M. Oxford, D. Parker, and M. Ryan, “Quantitative verification of certificate transparency gossip protocols,” in *2020 IEEE Conference on Communications and Network Security (CNS)*, 2020.
- [15] P. C. Dillinger, P. Manolios, D. Vroon, and J. S. Moore, “Acl2s: “the ACL2 sedan”,” in *International Conference on Software Engineering (ICSE)*, 2007.
- [16] H. Chamathi, P. C. Dillinger, P. Manolios, and D. Vroon, “The “acl2” sedan theorem proving system,” in *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, 2011.
- [17] D. Vyzovitis, “gossipsub: An extensible baseline pubsub protocol,” <https://github.com/libp2p/specs/blob/master/pubsub/gossipsub/README.md>, accessed 28 Nov 2022.
- [18] J. R. Douceur, “The sybil attack,” in *International workshop on peer-to-peer systems*. Springer, 2002, pp. 251–260.
- [19] H. Yu, M. Kaminsky, P. B. Gibbons, and A. Flaxman, “Sybilguard: Defending against sybil attacks via social networks,” *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 4, p. 267–278, aug 2006. [Online]. Available: <https://doi.org/10.1145/1151659.1159945>
- [20] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao, “Sybllimit: A near-optimal social network defense against sybil attacks,” in *2008 IEEE Symposium on Security and Privacy (sp 2008)*, 2008, pp. 3–17.
- [21] G. Danezis and P. Mittal, “Sybilinifer: Detecting sybil nodes using social networks,” in *Proceedings of the Network and Distributed System Security Symposium, NDSS 2009, San Diego, California, USA, 8th February - 11th February 2009*. The Internet Society, 2009. [Online]. Available: <https://www.ndss-symposium.org/ndss2009/sybilinifer-detecting-sybil-nodes-using-social-networks/>
- [22] D. Yuan, Y. Miao, N. Z. Gong, Z. Yang, Q. Li, D. Song, Q. Wang, and X. Liang, “Detecting fake accounts in online social networks at the time of registrations,” in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1423–1438. [Online]. Available: <https://doi.org/10.1145/3319535.3363198>
- [23] B. Yu, C.-Z. Xu, and B. Xiao, “Detecting sybil attacks in vanets,” *Journal of Parallel and Distributed Computing*, vol. 73, no. 6, pp. 746–756, 2013. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0743731513000191>
- [24] R. Shrestha, S. Djuraev, and S. Y. Nam, “Sybil attack detection in vehicular network based on received signal strength,” in *2014 International Conference on Connected Vehicles and Expo (ICCVE)*, 2014, pp. 745–746.
- [25] B. Blanchet *et al.*, “Modeling and verifying security protocols with the applied pi calculus and proverif,” *Foundations and Trends® in Privacy and Security*, vol. 1, no. 1-2, pp. 1–135, 2016.
- [26] S. Meier, B. Schmidt, C. Cremers, and D. Basin, “The TAMARIN prover for the symbolic analysis of security protocols,” in *International conference on computer aided verification*. Springer, 2013, pp. 696–701.
- [27] F. He, L. Baresi, C. Ghezzi, and P. Spoletini, “Formal analysis of publish-subscribe systems by probabilistic timed automata,” in *International Conference on Formal Techniques for Networked and Distributed Systems*. Springer, 2007, pp. 247–262.
- [28] P.-É. Dagand, D. Kostić, and V. Kuncak, “Opis: Reliable distributed systems in ocaml,” in *Proceedings of the 4th international workshop on Types in language design and implementation*, 2009.
- [29] R. Bakhshi, F. Bonnet, W. Fokkink, and B. Haverkort, “Formal analysis techniques for gossiping protocols,” *ACM SIGOPS Operating Systems Review*, 2007.
- [30] R. Bakhshi, D. Gavidia, W. Fokkink, and M. Van Steen, “A modeling framework for gossip-based information spread,” in *Eighth International Conference on Quantitative Evaluation of SysTems*, 2011.
- [31] H. van Ditmarsch, M. Gatteringer, L. Kuijter, and P. Pardo, “Strengthening gossip protocols using protocol-dependent knowledge,” *Journal of Applied Logics*, 2019.
- [32] G. J. Holzmann, “The model checker spin,” *IEEE Transactions on software engineering*, vol. 23, no. 5, pp. 279–295, 1997.
- [33] P. C. Dillinger, P. Manolios, D. Vroon, and J. S. Moore, “ACL2s: “the ACL2 sedan”,” *Electronic Notes in Theoretical Computer Science*, vol. 174, no. 2, pp. 3–18, 2007, proceedings of the 7th Workshop on User Interfaces for Theorem Provers (UITP 2006).

- [34] H. R. Chamarithi, P. Dillinger, P. Manolios, and D. Vroon, “The acl2 sedan theorem proving system,” in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2011, pp. 291–295.
- [35] M. Kaufmann, P. Manolios, and J. S. Moore, *Computer-Aided Reasoning: An Approach*. Kluwer Academic Publishers, July 2000.
- [36] M. Kaufmann and J. S. Moore, “ACL2 homepage,” 2022. [Online]. Available: <https://www.cs.utexas.edu/users/moore/acl2/>
- [37] H. R. Chamarithi, D. P. C., and P. Manolios, “Data Definitions in the ACL2 Sedan,” *ACL2*, 2014.
- [38] H. R. Chamarithi, P. C. Dillinger, M. Kaufmann, and P. Manolios, “Integrating testing and interactive theorem proving,” in *Proceedings 10th International Workshop on the ACL2 Theorem Prover and its Applications*, ser. EPTCS, D. S. Hardin and J. Schmaltz, Eds., vol. 70, 2011, pp. 4–19.
- [39] H. R. Chamarithi and P. Manolios, “Automated specification analysis using an interactive theorem prover,” in *International Conference on Formal Methods in Computer-Aided Design, FMCAD ’11*, P. Bjesse and A. Slobodová, Eds. FMCAD Inc., 2011, pp. 46–53. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2157665>
- [40] H. R. Chamarithi, “Interactive non-theorem disproving,” Ph.D. dissertation, Northeastern University, 2016.
- [41] P. Manolios and D. Vroon, “Termination analysis with calling context graphs,” in *Computer Aided Verification, 18th International Conference, CAV, Proceedings*, ser. LNCS, T. Ball and R. B. Jones, Eds., vol. 4144. Springer, 2006, pp. 401–414.
- [42] —, “Algorithms for ordinal arithmetic,” in *19th International Conference on Automated Deduction – CADE-19*, ser. LNAI, F. Baader, Ed., vol. 2741. Springer-Verlag, July/August 2003, pp. 243–257.
- [43] —, “Integrating reasoning about ordinal arithmetic into ACL2,” in *Formal Methods in Computer-Aided Design FMCAD*, ser. LNCS. Springer-Verlag, November 2004.
- [44] —, “Ordinal Arithmetic: Algorithms and Mechanization,” *Journal of Automated Reasoning*, vol. 34, no. 4, 2005.
- [45] “TESTGROUND,” <https://docs.testground.ai/>, 2022, accessed 24 July 2022.
- [46] B. Blanchet *et al.*, “An efficient cryptographic protocol verifier based on prolog rules,” in *csfw*, 2001.
- [47] B. Schmidt, S. Meier, C. Cremers, and D. Basin, “Automated analysis of diffie-hellman protocols and advanced security properties,” in *2012 IEEE 25th Computer Security Foundations Symposium*, 2012.
- [48] D. Basin, J. Dreier, L. Hirschi, S. Radomirovic, R. Sasse, and V. Stetler, “A formal analysis of 5g authentication,” in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018.
- [49] R. R. Singh, J. Moreira, T. Chothia, and M. D. Ryan, “Modelling of 802.11 4-way handshake attacks and analysis of security properties,” in *International Workshop on Security and Trust Management*, 2020.
- [50] M. v. Hippel, C. Vick, S. Tripakis, and C. Nita-Rotaru, “Automated attacker synthesis for distributed protocols,” in *International Conference on Computer Safety, Reliability, and Security*, 2020.
- [51] M. L. Pacheco, M. von Hippel, B. Weintraub, D. Goldwasser, and C. Nita-Rotaru, “Automated attack synthesis by extracting finite state machines from protocol specification documents,” in *International Symposium on Security and Privacy*, 2022.
- [52] R. Bakhshi and D. Gurov, “Verification of peer-to-peer algorithms: A case study,” *Electronic Notes in Theoretical Computer Science*, 2007.
- [53] J. Brunel, D. Chemouil, and J. Tawa, “Analyzing the fundamental liveness property of the chord protocol,” in *2018 Formal Methods in Computer Aided Design (FMCAD)*, 2018.
- [54] D. Woos, J. R. Wilcox, S. Anton, Z. Tatlock, M. D. Ernst, and T. Anderson, “Planning for change in a formal verification of the raft consensus protocol,” in *Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs*, 2016.
- [55] J. R. Wilcox, D. Woos, P. Panchekha, Z. Tatlock, X. Wang, M. D. Ernst, and T. Anderson, “Verdi: a framework for implementing and formally verifying distributed systems,” in *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2015.
- [56] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish *et al.*, “sel4: Formal verification of an os kernel,” in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, 2009.
- [57] L. Lamport, “Specifying systems: the tla+ language and tools for hardware and software engineers,” 2002.
- [58] Y. Yu, P. Manolios, and L. Lamport, “Model checking TLA+ specifications,” in *Advanced Research Working Conference on Correct Hardware Design and Verification Methods*. Springer, 1999, pp. 54–66.
- [59] E. Gafni and L. Lamport, “Disk paxos,” *Distributed Computing*, vol. 16, no. 1, pp. 1–20, 2003.
- [60] W. Schultz, I. Dardik, and S. Tripakis, “Formal verification of a distributed dynamic reconfiguration protocol,” in *Proceedings of the 11th ACM SIGPLAN International Conference on Certified Programs and Proofs*, 2022.
- [61] L. Lamport, “Byzantizing paxos by refinement,” in *International symposium on distributed computing*, 2011, TLA+ proof available at <https://lamport.azurewebsites.net/tla/byzpaxos.html>, accessed 29 July 2022.
- [62] E. Koutanov, “Spire: A cooperative, phase-symmetric solution to distributed consensus,” *IEEE Access*, 2021.
- [63] C. Hawblitzel, J. Howell, M. Kapritsos, J. R. Lorch, B. Parno, M. L. Roberts, S. Setty, and B. Zill, “Ironfleet: proving practical distributed systems correct,” in *Proceedings of the 25th Symposium on Operating Systems Principles*, 2015.
- [64] K. M. Chandy and J. Misra, *Parallel Program Design: a Foundation*, 1988.
- [65] M. H. Park and M. Kim, “A distributed synchronization scheme for fair multi-process handshakes,” *Information Processing Letters*, 1990.
- [66] W. H. Hesselink, “A mechanical proof of segall’s pif algorithm,” *Formal Aspects of Computing*, 1997.
- [67] B. Bonakdarpour, M. Bozga, M. Jaber, J. Quilbeuf, and J. Sifakis, “A framework for automated distributed implementation of component-based models,” *Distributed Computing*, 2012.
- [68] Q. Bramas, D. Foreback, M. Nesterenko, and S. Tixeuil, “Packet efficient implementation of the omega failure detector,” *Theory of Computing Systems*, 2019.
- [69] M. Grundmann and H. Hartenstein, “Verifying payment channels with tla+,” in *2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2022.
- [70] K. Babel, P. Daian, M. Kelkar, and A. Juels, “Clockwork finance: Automated analysis of economic security in smart contracts,” *Cryptology ePrint Archive*, Paper 2021/1147, 2021. [Online]. Available: <https://eprint.iacr.org/2021/1147>
- [71] P. Tolmach, Y. Li, S.-W. Lin, Y. Liu, and Z. Li, “A survey of smart contract formal specification and verification,” *ACM Computing Surveys (CSUR)*, 2021.
- [72] J. Bornholt, R. Joshi, V. Astrauskas, B. Cully, B. Kragl, S. Markle, K. Sauri, D. Schleit, G. Slatton, S. Tasiran *et al.*, “Using lightweight formal methods to validate a key-value storage node in amazon s3,” in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, 2021.
- [73] K. L. McMillan and L. D. Zuck, “Formal specification and testing of QUIC,” in *Proceedings of ACM Special Interest Group on Data Communication (SIGCOMM’19)*, 2019.
- [74] J. Wu, R. Wu, D. Xu, D. J. Tian, and A. Bianchi, “Formal model-driven discovery of bluetooth protocol design vulnerabilities,” 2022.

- [75] T. Chothia, J. De Ruiter, and B. Smyth, “Modelling and analysis of a hierarchy of distance bounding attacks,” in *27th USENIX Security Symposium (USENIX Security 18)*, 2018.
- [76] T. Chothia, “Analysing the mute anonymous file-sharing system using the pi-calculus,” in *International Conference on Formal Techniques for Networked and Distributed Systems*, 2006.
- [77] C. Cremers, M. Horvat, J. Hoyland, S. Scott, and T. van der Merwe, “A comprehensive symbolic analysis of tls 1.3,” in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017.
- [78] W.-F. Bisimulation, P. Manolios, K. Namjoshi, and R. Sumners, “Linking theorem proving and model checking with,” in *Computer Aided Verification: 11th International Conference*, 1999.
- [79] “Filecoin in 2021: Looking back at a year of exponential growth,” <https://filecoin.io/blog/posts/filecoin-in-2021-looking-back-at-a-year-of-exponential-growth/>, accessed 20 November 2022.
- [80] “Ethereum stats,” <https://www.stateofthedapps.com/platforms/ethereum>, 11 2022, accessed 20 November 2022.
- [81] “100+ ethereum apps you can use right now [2021 update],” <https://consensus.net/blog/news/90-ethereum-apps-you-can-use-right-now/>, March 2021, accessed 20 November 2022.
- [82] “Phase 0 – networking,” <https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/p2p-interface.md#the-gossip-domain-gossipsub>, published 13 July 2022.

8. Appendix

Implementation Details in ACL2s Defining custom data types in ACL2s is simple and intuitive. Consider the following excerpts from our model.

```
1 (defdata pos-rat (range rational (0 <= _)))
2
3 (defdata topic-counters
4   (record (invalidMessageDeliveries . pos-rat)
5     (meshMessageDeliveries . pos-rat)
6     (meshTime . pos-rat)
7     (firstMessageDeliveries . pos-rat)
8     (meshFailurePenalty . pos-rat)))
```

We define `topic-counters`, a named record to store topic based performance counters. Since these counters can never be negative, and may be rational (due to decay), we specify their types as appropriately defined `pos-rats`. We then define a map `pt-tctrs-map` to store topic-counters per peer per topic as follows:

```
1 (defdata pt (cons peer topic))
2 (defdata pt-topic-counters-map (alistof pt
   topic-counters))
```

Now we need a lookup function to find topic-counters, given a peer and a topic.

```
1 (definec lookup-topic-counters (p :peer top :topic
   map :pt-topic-counters-map) :topic-counters
2   (match map
3     (( (new-topic-counters))
4      (((!p . !top) . tct) . &) tct)
5     ((& . rst) (lookup-topic-counters p top rst)))
   )
```

We use `match` to pattern-match `map` against possible syntactic structures. If `map` is an empty list, we return a new `topic-counters`, with all counters initialized to

Constant or Weight	MESSAGES	BLOCKS
TopicWeight	1	1
TopicCap	0	0
$w_1(t)$	2.78	0.027
$w_2(t)$	0.5	5
$w_3(t)$	0	0
$w_{3b}(t)$	0	0
$w_4(t)$	-1000	-1000
w_5 (global)	1	1
w_6 (global)	-100	-100
w_7 (global)	-10	-10
D	8	8

TABLE 4. FILECOIN’S TWP. ADAPTED FROM GITHUB.COM/FILECOIN-PROJECT/LOTUS.

Constant or Weight	BLOCKS	AGG	SUB1	SUB2	SUB3
TopicWeight	0.8	0.5	0.33	0.33	0.33
TopicCap	32.72	32.72	32.72	32.72	32.72
$w_1(t)$	0.0324	0.0324	0.0324	0.0324	0.0324
$w_2(t)$	1	0.128	0.95	0.95	0.95
$w_3(t)$	-0.717	-0.064	-37.55	-37.55	-37.55
$w_{3b}(t)$	-0.717	-0.064	-37.55	-37.55	-37.55
$w_4(t)$	-140.45	-140.45	-4544	-4544	-4544
w_5 (global)	1	1	1	1	1
w_6 (global)	-35.11	-35.11	-35.11	-35.11	-35.11
w_7 (global)	-15.92	-15.92	-15.92	-15.92	-15.92
D	8	8	8	8	8

TABLE 5. ETH2.0’S TWP. ADAPTED FROM GITHUB.COM/SILESIACOIN/PRYSM-SPIKE.

zero. Otherwise, if the pair of given peer and topic exactly matches the key in the first key-value pair of `map`, we return the corresponding value, otherwise we recurse on the rest of `map`.

Upon admitting `lookup-topic-counters`, ACL2s extends its logic with (1) a definitional axiom: given input arguments satisfy their types, calling `lookup-topic-counters` equals its function body, and (2) a function contract theorem: given input arguments satisfy their types calling `lookup-topic-counters` returns a `topic-counters`, as specified by the function output type. Such axioms could introduce unsoundness if `lookup-topic-counters` did not terminate. So before admitting the function, ACL2s uses termination analysis to prove that `lookup-topic-counters` is indeed terminating. Hence, admitting `lookup-topic-counters` produces theorems about its definition, termination and input-output contracts.

8.1. Use of GossipSub in Applications

GossipSub is used most notably in FileCoin and Eth2.0. FileCoin is a decentralized storage solution based on Proof-of-Space-Time. It is a P2P alternative to the client-server model, where content (e.g., websites) are addressed by their hashes and nodes can earn cryptocurrency by acting as hosts. Peers wishing to publish content find hosts using *ask* orders in a distributed auction house (the hosts respond with *bid* orders). Both types of orders are disseminated using Gos-

Constant or Weight	Value
TopicWeight	40
TopicCap	5
$w_1(t)$	10
$w_2(t)$	10
$w_3(t)$	-1
$w_{3b}(t)$	-1
$w_4(t)$	-1
w_5 (global)	10
w_6 (global)	-1
w_7 (global)	-1
D	5

TABLE 6. OUR PATHOLOGICAL twp . WE RAN THE NETWORK WITH FIVE TOPICS, ALL CONFIGURED THE SAME WAY.

Constant or Weight	BLOCKS	MESSAGES
TopicWeight	0.5	0.5
TopicCap	100	10
$w_1(t)$	0.027	0.027
$w_2(t)$	5	5
$w_3(t)$	-1000	-1000
$w_{3b}(t)$	-1000	-1000
$w_4(t)$	-1000	-1000
w_5 (global)	1	1
w_6 (global)	-100	-100
w_7 (global)	10	10
D	8	8

TABLE 7. OUR GOOD CONFIGURATION twp . IT SATISFIES ALL OUR PROPERTIES

sipSub. FileCoin uses two topics: BLOCKS and MSGS [1]. In 2021, FileCoin had >14 EiB in network storage capacity, $>3,600$ network storage providers, and tens of millions of uploads by tens of thousands of users [79]. Many real-world applications are built on top of FileCoin including the InterPlanetary File System (IPFS), various Non-Fungible Token (NFT) marketplaces, etc. As of November 2022, the market cap of FileCoin is \$1.4B USD [3].

Eth2.0 is the second most valuable cryptocurrency, after Bitcoin, with a November 2022 market cap of \$143B USD [4]. Eth2.0 supports Turing-complete smart contracts with fine-grained control over the amount of value being exchanged, an internal program state, and access to blockchain data such as nonces [2]. Over 2900 applications are built on Eth2.0, some with millions of active daily users, including NFTs, Decentralized Autonomous Organizations (DAOs), Decentralized Finance apps (DeFi), etc. [80], [81] GossipSub is the primary messaging layer protocol in Eth2.0. It is used to disseminate all kinds of data throughout the chain, including newly signed blocks, attestations, payload encodings, etc. over dozens of topics [82].

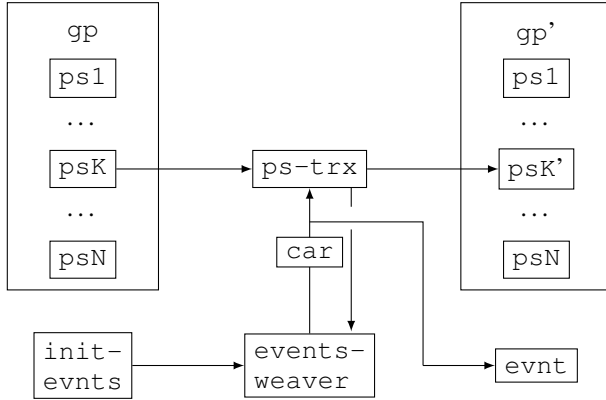


Figure 3. Data-flow diagram of the Group transition function, gs-trx . This function takes as input a Group called gp , a list of events called init-evnts , a twp , and an oracle. The first event coming out of events weaver determines which peer-state psK in the Group gp will get updated this round. The twp and oracle are both passed into the peer-state transition function calls, and in this figure, as in Figure 4, are omitted for simplicity. The events-weaver function splices the init-evnts with the events emitted by the most recent previous application of the ps-trx function. The car function selects the first event from the list of events generated by events-weaver, which serves as input to ps-trx in this round, as well as an output of the overall Group transition function.

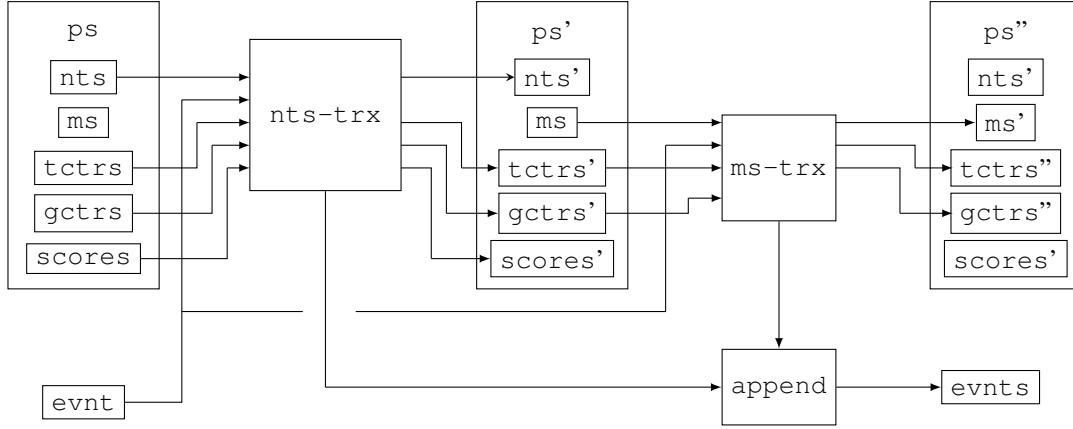


Figure 4. Data-flow diagram of the peer-state transition function, `ps-trx`. The function takes as input a peer-state called `ps`, a list of events called `evnt`, a `twp`, and an oracle. The oracle is used to model nondeterministic decisions, *e.g.*, the specific subset of peers to send gossip to. Both the `twp` and the oracle are omitted from the diagram. The other inputs are shown in on the left-hand side. The function outputs a new peer-state called `ps'`, and a (possibly empty) list of events called `evnts'`, both shown on the right-hand side. The box labeled `nts-trx` denotes a transition function for the `nbr-topic-state`, which outputs an updated `nbr-topic-state` called `nts'` as well as a (possibly empty) list of emitted events. It also generates new topic-counters and global-counters, and calculates peer scores. Next, the `msgs-state` component `ms'` of the peer-state `ps'` is fed into the `ms-trx` function, along with the updated topic-counters and global-counters, and the initial event. The final peer-state `ps''` consists of `nts'`, `ms'`, `tctrs''`, `gctrs''`, and `scores'`. A possibly empty list of events called `evnts` is also emitted by splicing the lists emitted by the `nts-trx` and `ms-trx` functions.

Parameter	Type	Description	Guidance
<code>PruneBackoff</code>	Duration	Duration before pruned peer may re-graft	Default to 1 minute
<code>UnsubscribeBackoff</code>	Duration	Duration before unsubscribed peer may re-subscribe	Default to 10 seconds
<code>FloodPublish</code>	Boolean	Enable/disable optional flood publishing	Default to true
<code>GossipFactor</code>	Float	Fraction of positive-scoring peers to emit gossip to	Default to 0.25, must be in [0, 1]
<code>D</code>	Integer	Desired outbound degree of each mesh	Default to 6
<code>Dlow</code>	Integer	Lower bound for outbound degree of each mesh	Default to 4
<code>Dhi</code>	Integer	Upper bound for outbound degree of each mesh	Default to 12
<code>Dlazy</code>	Integer	Desired outbound degree for gossip emission	Default to <code>D</code>
<code>HeartbeatInterval</code>	Duration	Duration between heart beat maintenances	Default to 1 second
<code>FanoutTTL</code>	Duration	Time-to-live for fanouts	Default to 1 minute
<code>SeenTTL</code>	Duration	Time-to-live for cache of seen message identifiers	Default to 2 minutes
<code>McacheLen</code>	Integer	Number of history windows in message cache	Default to 5
<code>McacheGossip</code>	Integer	Number of history windows to use when emitting gossip	Default to 3
<code>Dscore</code>	Integer	Number of highest-scoring peers to retain when pruning due to over-subscription	4 or 5 for a <code>D</code> of 6
<code>Dout</code>	Integer	Number of outbound connections to keep in a mesh	Default to 2 for a <code>D</code> of 6, must be in [<code>Dlow</code> , <code>D/2</code>]
<code>GossipThreshold</code>	Float	Only emit gossip to peers who score above this threshold	Must be < 0
<code>PublishThreshold</code>	Float	Only send new messages to peers who score above this threshold	Must be $\leq \text{GossipThreshold}$
<code>GraylistThreshold</code>	Float	Ignore control messages from peers scoring below this threshold	Must be $< \text{PublishThreshold}$
<code>OpportunisticGraftThreshold</code>	Float	Opportunistic grafting is triggered when the median score of neighbors in a mesh falls below this threshold	Must be ≥ 0
<code>DecayInterval</code>	Duration	Interval at which counters decay	
<code>DecayToZero</code>	Float	When indicators fall below this value they round down to zero	Should be close to 0.0
<code>RetainScore</code>	Duration	Duration to retain peer scores after they disconnect	

TABLE 8. PARAMETERS USED BY GOSSIP_{SUB}'S DEFENSE MECHANISMS. DESCRIPTIONS AND GUIDANCE ADAPTED FROM THE PROSE SPECIFICATION [5], [6].

```

1 (property (ptc :pt-tctrs-map pcm :p-gctrs-map p :peer top :topic)
2   :hyps (and (member-equal top (strip-cars *eth-twp*))
3             (member-equal (cons p top) (strip-cars ptc)))
4             (> (lookup-score p (calc-nbr-scores-map ptc pcm *eth-twp*)) 0))
5   (> (calcScoreTopic (lookup-tctrs p top ptc) (cdr (assoc-equal top *eth-twp*))) 0))

```

Figure 5. Property 1 definition in ACL2s