Enhancing Network Privacy in Bitcoin

Aseem R. Baranwal
Ben Armstrong
School of Computer Science
University of Waterloo
Canada

Abstract

We develop new anonymity-preserving protocols for the propagation of transaction messages in the Bitcoin network. *Diffusion*, the spreading protocol currently used in the Bitcoin network has been shown to have poor anonymity properties; in this project, we propose two new spreading protocols and compare them with the existing protocol and recently proposed alternatives.

Keywords bitcoin, privacy, anonymity, spreading

ACM Reference Format:
Aseem R. Baranwal and Ben Armstrong. 2019. Enhancing Network Privacy in Bitcoin. In *University of Waterloo, April 2019*. ACM, New York, NY, USA, 12 pages.

1 Introduction

In the past decade, cryptocurrencies have enjoyed a tremendous surge in popularity. Cryptocurrencies allow for pseudo-anonymous transactions between any individuals within the network. There are many such currencies but Bitcoin is, arguably, the original and is certainly the most popular. A vital component of Bitcoin's success is its decentralization and anonymity. Transactions are spread through a peer-to-peer network to avoid censorship and allow for audits. However, Bitcoin users rely on the link between the cryptographic keys associated with their accounts not being linked to the physical node the user corresponds to in the Bitcoin network. They have a secondary goal of spreading transactions through the entire network quickly. Recent work has shown that current methods of spreading transactions through Bitcoin's P2P network do not adequately obscure the connection between the creator of a Bitcoin transaction and the creator's IP address [1, 2, 9, 13, 16].

April 13, 2019, University of Waterloo

In this project, we develop new methods of propagating transactions through Bitcoin's network with the goal of maintaining privacy without sacrificing performance: it is important that nodes receive transactions quickly as well as securely. Our new spreading protocols are tested in simulations against a novel spy-based adversary, allowing us to compare the level of anonymity afforded to users of existing protocols as well as our new protocols. We show that a new protocol, Synchronized Diffusion, provides similar levels of security while significantly reducing the time taken to propagate a transaction through the network.

We begin with a review of the related literature in this field in § 2, though we find little work specifically focused on the development of new privacy-preserving spreading protocols. This is followed by a description of the model within which we are working in § 3. We then describe our design of the two new spreading protocols: Synchronized Diffusion, and Random Diffusion in § 5. An explanation of various possible attacks by the adversary follows in § 6, and finally, we present the results and analyses of our experiments. We conclude with some discussion and brief thoughts for future work.

2 Related Work

A variety of recent research has focused on the goal of connecting Bitcoin network nodes with IP addresses, and the challenge of avoiding this identification. Research into this area is frequently inspired by graph theoretic approaches, or by existing ideas from security or systems research.

The first analyses of privacy in Bitcoin appeared in 2013. Androulaki et al. showed the feasibility of using clustering techniques to identify similarly behaving nodes and link a user to multiple public keys [1]. They also showed provided evidence that obfuscating coin ownership through mixing techniques is not resistant to analysis. Similar results are obtained by Reid and Harrigan who perform the first privacy-oriented analysis of the Bitcoin network [13]. They demonstrate an alternative method of linking public keys to individual users for a significant fraction of transactions. These results encouraged users to enhance their anonymity and several users began to use Tor to connect to the Bitcoin network. Shortly thereafter, it was shown that using Tor to connect to Bitcoin did very little to enhance privacy and exposed users to man-in-the-middle and denial-of-service attacks [3].

These attacks placed attention on the privacy granted by the spreading protocol in Bitcoin. [10] provided the first analysis of the trickle protocol, the then-current spreading protocol used by the official Bitcoin implementation. Using several months worth of transaction data they were able to map between 250 and 1100 IP addresses to transactions associated with those IPs – a significant fraction of the number of active Bitcoin users at the time. Similar results were provided by Biryukov et al. demonstrating a simple and affordable attack able to identify the IP address of up to one-third of transactions (based on simulation results, varying the resources available to the adversary) [2].

The trickle spreading protocol was then replaced by a diffusion protocol for which Fanti and Viswanath demonstrated an attack, and shortly thereafter suggested an alternative spreading protocol. A graph-theoretic analysis provided a maximum likelihood estimator able to identify the IP of a transaction in 30% to 90% of instances [9]. This paper also suggests a number of attacks in which a global adversary records the timestamps at which it sees transactions and iteratively improves its estimates about the potential identity of a sender over multiple transactions. Followup work has described a new spreading protocol, Dandelion [4]. Dandelion has the goal of spreading messages through the Bitcoin network asymmetrically in a manner which would defeat previously reported deanonymization attacks. After collecting community feedback, Dandelion has been refined and presented as Dandelion++, representing the state-of-the-art in privacy-preserving Bitcoin spreading protocols [8]. Dandelion++ provides a number of small changes that dramatically improve the security it provides. In § 5.2.1, we address these protocols in more detail.

3 System Model

In this section, we describe our model for the relevant components of the Bitcoin network – P2P network topology, transaction propagation, and the adversarial model. We begin, however, with a brief overview of each of the previous spreading algorithms that have been used in Bitcoin to ensure that a discussion about them in the remainder of the paper is clear.

3.1 Bitcoin Spreading Protocol

Whenever a transaction is completed or a new block is formed, it is broadcast over the Bitcoin network. Here we briefly describe the propagation of these messages along with the spreading protocols used to broadcast them.

By default, Bitcoin peers try to maintain 8 outgoing connections (entry nodes). In addition, Bitcoin servers can accept up to 117 incoming connections. To initiate a message transmission to another peer, the sender transmits an `INVENTORY` message with the hash of the transaction (or block). The receiver peer runs several checks on the hash and if the checks pass, it requests the actual transaction by sending a `GETDATA` message. The sender then transmits the transaction in a `TRANSACTION` message. When the receiver gets the transaction, it advertises to its peers with an `INVENTORY` message, continuing the propagation.

3.1.1 Trickle Spreading

This protocol was used in Bitcoin up to 2015. When a client generates a transaction, it puts the message into an outgoing queue to be forwarded to all the neighbours. It then computes a hash of the transaction and a secret salt. If the computed hash has the last two bits set to zero then the transaction is forwarded immediately to all the 8 entry nodes, otherwise, the queue of a randomly chosen neighbour (*trickle* node) for outgoing transactions is flushed. For each neighbour, a hash of the address to be forwarded, a secret salt, current day, and the memory address of the data structure describing the neighbour is computed. The peer then sorts the list of neighbours based on the computed hashes and chooses the first entry as a *responsible* node [2]. For the actual transmission, every 100 milliseconds one neighbour is randomly selected from the list of all neighbours and the queue for outgoing messages is flushed for this neighbour. This chosen node for each new round of 100 ms is called the trickle node.

3.1.2 Diffusion Spreading

In 2015, trickling was replaced by diffusion in the reference Bitcoin implementation [1]. In this protocol, every node (source or relayer) transmits each transaction to each of its neighbours that have not yet seen that transaction, with an exponential distribution on a parameter λ. On receiving a message, the receiver then chooses fresh independent delays for further propagation to its neighbours. Thus, diffusion spreading is modelled as a Poisson point process. This process continues until all nodes in the network are aware of the message.

Recent analysis has shown that diffusion is no better than trickling in terms of preserving anonymity [9]. According to the source code on GitHub, the required Poisson delay is given by,

$$\text{delay} = 0{:}5 \quad \log^1 1 + Q^0 \quad \text{AVG\_INTVL} \quad 10^6;$$

where $Q = 3{:}5527136788 \quad 10^{-15} \quad GetRand^1 2^{48} {}^0$, and the value of AVG_INTVL is 30 seconds for address broadcast and 5 seconds for inventory broadcast messages. Here, $GetRand^1 x^0$ produces a random integer between 0 and x. Thus, the final value is a randomly chosen positive delay interval padded with an offset of 0:5 seconds.

[1] The pull request to replace trickling by Poisson delays was created by Pieter Wuille in November 2015, and is available at https://github.com/bitcoin/bitcoin/pull/7125

3.2 Network Topology

There are two types of nodes in the Bitcoin network: clients and servers. Clients do not accept incoming TCP connections and do not relay transactions. Hence, we only consider servers in all of our analyses. Furthermore, servers are more persistent in the network, and a solution that improves the anonymity of a server can easily be generalized to clients as well. As servers typically persist for moderate to long periods of time (long enough to relay several transactions), we assume for the purposes of our analysis that the network does not change.

The network is modelled as a graph $H = (V, E)$ where the vertex set V denotes the nodes, and the set of edges E denotes the connections between them. Each server is allowed to maintain up to 8 outgoing connections to active Bitcoin nodes, and up to 125 total active connections. Once established, TCP connections are bidirectional. The local neighbourhood of a node v in graph G is denoted by $\eta(v, G)$, the set of all direct neighbours of v.

To model the honest and adversarial nodes, we partition the network in two sets. $V_H \subseteq V$ denotes the set of honest nodes, and $V_A = V \setminus V_H$ denotes the set of colluding adversary nodes. We discuss the adversary model in more depth in § 3.4.

3.3 Transaction Model

In practice, nodes will generate transactions at different rates. In the interest of simplicity, we assume that nodes generate a small number of transactions in any given time interval.

We denote by X the set of all transaction messages, and X_v a transaction message originating from an honest node $v \in V$. The current Bitcoin reference implementation uses the diffusion spreading protocol, which is symmetric in the sense that it does not use any metadata associated with the neighbours to influence forwarding decisions. Thus, in expectation, messages spread evenly outward from a source allowing an adversary to identify the source of messages. The solutions proposed by Fanti et al. [8] attempt to break this symmetry while maintaining resistance to deanonymization. Our work follows similar conceptual ideas.

3.4 Adversary Model

In previous literature, the most commonly studied adversary model is the **snapshot adversary** [11, 15]. In this model, the adversary observes a set of nodes that have already seen the transaction at a single time instant T. Fanti et al. introduce a new kind of adversary in [9], known as an **eavesdropper adversary**. This model utilizes a *supernode* that can make multiple connections to each honest server in the network, each with a different pair of (IP address, port). They also assume that the eavesdropper makes a fixed number $\theta \geq 1$ of connections to each server, and furthermore, these connections are not included in the original graph of nodes H.

In our analyses, we use the less studied **spy-based adversary**. In this model, the adversary observes exact timestamps for the set of nodes V_A, which does not include the source. We parameterize the fraction $p = \frac{|V_A|}{|V|}$ of nodes that are adversarial for evaluation purposes. This model assumes that spy nodes observe the exact timestamp at which each message is received. Yet, unlike the eavesdropper, this model gets rid of the assumption that the adversary is connected to all nodes including the source. Removing this assumption makes the analysis more realistic, but difficult, which is why it has not appeared a lot in the literature. Another crucial assumption we make is that spy nodes can create an arbitrary number of outgoing connections to any node that they want, but cannot force an honest node to create an outgoing connection to a spy node.

The goal of the adversary is to deanonymize nodes by linking their transactions and public keys to their IP addresses, where each IP address is represented by a node in the P2P network. We assume that honest nodes cannot differentiate between an honest and a spy node. When a transaction is broadcast, the spy nodes log two pieces of information: (1) the timestamp at which they received the message, and (2) the set of honest neighbours from which they received the message. Over time, the colluding adversaries learn the network graph H. The actual knowledge of the graph depends on whether the network is static or changes rapidly during the considered time interval. Given that we are considering only a small number of transactions at any time, we assume that the network remains static during our analysis. Adversary nodes can learn their local neighbourhood within this network,

$$\eta(V_A, H) = \bigcup_{u \in V_A} \eta(u, H).$$

After the timestamps are collected, all the spy nodes will work together to attempt identification of the source. To quantify this, we follow the ideas in [4]. A tuple $(x, u, T_u(x))$ denotes that transaction x was relayed by an honest node to adversary u at time $T_u(x)$. For each honest server $v \in V_H$, S_v denotes the set of all such tuples where the transactions are relayed by v. In order to identify the source, the colluding adversary then uses the set of S_v learned by the spy nodes, and the information about the network, $\eta(V_A, H)$ to output a mapping $E : X \to V_H$ between transactions and honest servers. The output mapping is chosen to maximize the utility function of the adversary, which is defined in § 4. The actual attacks are described in § 6.

4 Goals

The broad goal of this work is to improve the privacy of Bitcoin users. Specifically, we aim to increase the difficulty of linking a given transaction with the IP address of the user

that originated that transaction by obfuscating a transaction's path through the Bitcoin network without introducing additional latency into transaction propagation.

The privacy metrics we aim to optimize are precision and recall. [4] used a multiclass extension of precision and recall to quantify the privacy granted by the Dandelion protocol. Precision of an adversarial estimator, E, guessing the source of a transaction X_v is defined as the ratio of the number of transactions for which the estimator correctly guesses v as the source to the total number of transactions for which the estimator guesses v as the source, or,

$$P[v] = \frac{\sum_{X_v} \mathbb{1}\{E[X_v] = v\}}{\sum_{w \in V_H} \mathbb{1}\{E[X_w] = v\}}$$

Precision is then averaged across all nodes to obtain a final precision score for a given estimator. Recall is defined as the fraction of transactions correctly labelled as originating at v divided by the total number of transactions originating at v.

$$R[v] = \frac{\sum_{X_v} \mathbb{1}\{E[X_v] = v\}}{|\{X_w : w = v\}|}$$

Similarly, recall for each node v is averaged to obtain a final recall score.

Maximizing privacy is often at odds with the goal of improving performance. In order to provide a useful contribution, we must also ensure that our protocols do not decrease system performance. In this context, the important metric is the time a transaction takes to spread through the Bitcoin network. Previous work has shown that 95% of nodes typically see transactions in less than one minute; we aim to achieve similar results [6].

5 Design Components

5.1 Building Blocks

Before presenting our algorithms, we provide an overview of how the baseline algorithm works. Both our new protocol proposals build upon these crude ideas. There are 3 major components of our spreading algorithms that we describe here.

5.1.1 Anonymity Graph

From the works of Biryukov, Venkatakrishnan, Fanti, Koshi and others [2, 8–10], we make the following observations regarding anonymity.

1. If the algorithm always propagates messages along the same path for each source, then the adversary can learn the relevant portion of the network structure over time, simply by observing received timestamps. Such a simple attack is described briefly in § 6.1.
2. On the other hand, if the spreading algorithm chooses a node's neighbours randomly for each transaction, then several symmetry-based deanonymization attacks are possible. We explain one of these attacks in more detail in § 6.2 and provide empirical evaluation for the same.

These observations indicate that a balance is required between asymmetry and randomness, which is maintained as follows. All nodes maintain an *epoch* of a preset time interval. In a single epoch, all nodes always relay messages to the same neighbours. A fresh set of neighbours is chosen by each node for propagation only when the current epoch ends and the next one begins. We call this set of nodes (fixed for an epoch) the *anonymity set*, and the corresponding sub-graph the *anonymity graph* for that epoch, denoted by G. Note that the anonymity set may contain adversarial nodes, and our algorithms do not assume otherwise. The complete Bitcoin P2P network graph is denoted by H, and we have $G \quad H$. Based on our assumption of the nodes' clocks being synchronized, the process of re-electing neighbours is simultaneous for all practical purposes. This provides a healthy combination of asymmetry and randomness. We make an assumption here that in a single epoch, an honest node broadcasts only one transaction. However, we provide an empirical analysis that considers the scenario when this assumption does not hold. The time interval for the epoch span can be fine-tuned in response to empirical analysis.

5.1.2 Message Relay

Whenever a source node or a relay node wants to transmit a message, it does so along the same edges as decided in the anonymity graph. Additionally, a relay node has the choice between (1) passing the message to neighbours in the anonymity graph, or (2) starting diffusion. The conditions for making these choices define our main algorithms in § 5.2.

5.1.3 Reliability

Every node keeps track of messages that it sends along with a timer. If the timer expires before the node hears about an `INV` message from other uninformed neighbours, then it begins diffusion as a fail-safe mechanism. This approach provides reliability against *black hole* attacks, discussed in § 6.4. In the implementation, this expiration timestamp for diffusion is set randomly through an exponential distribution with a preset mean value T_0,

$$T_d \quad \text{Time.Now()} + \exp\ \frac{1}{T_0}\ :$$

5.2 Algorithms

We develop and analyze two novel algorithms for propagating transactions through the Bitcoin network, based on the baseline constructs defined in § 5.1. The first is a two-phase protocol that relies on loose timing assumptions that we believe to be realistic in normal circumstances, while the second algorithm attempts to incorporate non-deterministic behaviour in order to maximize the effort required by an

adversary attempting to infer transaction sources. Both algorithms allow for tuning parameters to vary the trade-off between anonymity and performance.

5.2.1 Dandelion and Dandelion++

Before discussing our protocols, we provide an overview of the pre-existing state of the art algorithms, which are also the only alternative spreading algorithms proposed in the research area of privacy-preserving protocols. In [4], the authors present Dandelion, where they address the Bitcoin P2P network's poor anonymity properties through a ground-up redesign of the networking stack. The currently implemented protocol, diffusion, forms the baseline for their protocol. The authors justify this decision based on the good latency properties of diffusion due to exponential spreading. The basic ideas in Dandelion have been used previously in several point-to-point anonymous communication systems, for example by Reiter and Rubin in [14]; but a formal study of the same in the context of anonymous broadcast messaging had not appeared. The main algorithm involves two phases, the anonymity phase, and the spreading phase.

In the anonymity phase, the algorithm relays the transaction message over a line-graph for a random number of hops. In the spreading phase, the message is broadcast using diffusion until the whole P2P network is informed. The authors design a line-graph G H over which the anonymity phase occurs, while the spreading phase (diffusion) occurs over H. They also mention a possible enhancement of making this graph time-varying. Figure 1 shows an overview of the two phases.

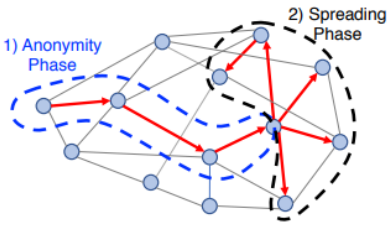

Figure 1. Message forwarding and spreading by Dandelion [4]. Here both phases occur over the same graph, i.e., $G = H$.

Dandelion++ [8] is an improvement on the design of Dandelion. In this proposal, the authors remove several assumptions made by Dandelion. Three of these major assumptions that do not hold in practical scenarios are the following.

1. Adversary nodes are characterized as "honest but curious", and have limited knowledge of the P2P graph.
2. The adversary nodes observe one transaction per node.
3. All nodes obey the Dandelion protocol.

The authors show in [8] how the anonymity guarantees of Dandelion are weakened through a combination of well-crafted attacks that exploit these assumptions. The design is fixed using several enhancements, the most prominent being the replacement of the line-graph by intertwined paths (or *cables* as referred to by the authors), which is a randomly approximated 4-regular graph. Figure 2 provides a view of this design. Through theoretical analysis, the authors show that the new design is fairly resilient to deanonymization attacks even when the above assumptions do not hold.

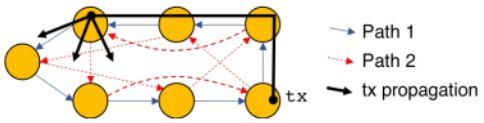

Figure 2. Message forwarding over one of the two intertwined paths on a 4-regular anonymity graph in Dandelion++ [8]. The transaction propagates over the blue path and is ultimately broadcast using diffusion.

We make two key observations from [4, 8]. First, the authors note that the spreading phase can be completely deanonymized, i.e., the node that starts the diffusion process can be identified by an adversary. This motivates us to look for alternative ways of diffusion-based propagation such that the source of diffusion remains anonymous. We present a novel algorithm along these ideas in § 5.2.2. Second, both Dandelion and Dandelion++ involve a two-phase algorithm where phase 2 is regular diffusion. We also look at the possibility of removing this hard constraint, and develop a randomized diffusion algorithm in § 5.2.3 which does not involve two phases.

5.2.2 Synchronized Diffusion

The first algorithm we discuss attempts to enhance Dandelion++ by anonymizing the node that starts the diffusion process in phase 2. The key assumption we make in this algorithm is that honest nodes have access to a clock that loosely corresponds with the value of the clocks of other nodes. Bitcoin already incorporates this assumption, requiring nodes to be within 70 minutes of each other. We extend this to assume that node clocks are within approximately 1-2 seconds of each other. This synchronicity is not required for the algorithm to function, however, it is important for security. Due to the speed of the modern internet, we believe that the vast majority of honest nodes readily meet this assumption. Synchronized diffusion works in two phases similar to Dandelion++.

The *first phase* of the algorithm operates as follows. The source node v first generates a timestamp T_{X_v} in the future at which time the diffusion phase will start. This timestamp

is chosen randomly from a uniform distribution,

$$T_{X_v} \quad \cup^1 T_{min}; T_{max}{}^{\circ}:$$

This timestamp is the distinguishing feature of our algorithm, and it helps in maintaining anonymity of the node that starts diffusion. The values T_{min} and T_{max} are carefully chosen after empirical evaluation. Then, v picks a node from its neighbours randomly in the anonymity graph G (constructed in a way similar to Dandelion++), to which it sends the message X_v along with the timestamp T_{X_v}. We discuss the scenario where this chosen neighbour can be adversarial in § 6. Upon receipt of a message, if T_{X_v} has not yet passed, the receiving node further relays the message X_v to a random neighbour and schedules a timer to start diffusion as soon as its clock hits T_{X_v}.

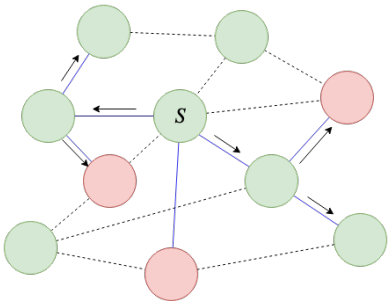

Figure 3. Phase 1 of Synchronized Diffusion: Message relay from source node S along the anonymity graph G (solid blue edges).

The *second phase* begins when a node's clock indicates that time T_{X_v} has passed. In this phase, all the nodes in the anonymity set that have seen the transaction begin to run the diffusion protocol which is currently implemented in Bitcoin, described in § 3.1. After the diffusion protocol starts, there is no timestamp parameter passed with a transaction and all other nodes also run the diffusion protocol, regardless of their current time. In ideal operation, all nodes that have seen X_v (the anonymity set) will begin their diffusion protocols simultaneously.

Figure 3 and Figure 4 show the two phases. The nodes coloured green belong to the set V_H, while the nodes coloured red are adversarial and belong to the set V_A. The example shown considers the case where the relay branching factor $m = 2$. Note that the anonymity set and its subset of active nodes that start diffusion in the second phase may contain adversarial nodes.

The algorithm is enhanced in a number of ways. First, rather than having nodes forward transactions to a single neighbour they forward to m neighbours. We use $m = 2$ in our experiments. Using $m > 1$ greatly increases the speed with which a transaction will spread through the network while resulting in what we expect to be a minimal decrease in privacy.

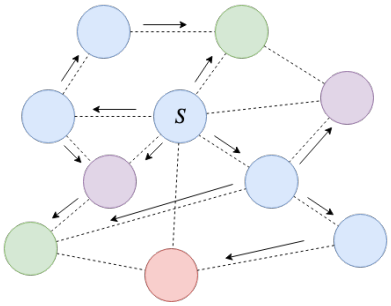

Figure 4. Phase 2 of Synchronized Diffusion: Every node in the active anonymity set (coloured blue) starts diffusion when its clock hits T_{X_v}.

Algorithm 1: SYNC-DIFFUSION

Data:
X_v : The message being broadcast.
T_{X_v} : Random delay to start diffusion.
m : The number of neighbours to select for phase 1.
Result: Broadcasts X_v via Synchronized Diffusion.)

```
neighbours := nil;
if Time.Now() < T_{X_v} then
    // Choose m neighbours randomly.
    neighbours ∪_m ¹η¹this; G⁰⁰;
    // Relay the message to each chosen neighbour.
    for nextHop in neighbours do
        SYNC-DIFFUSION¹nextHop; X_v; T_{X_v}⁰;
    end
end
// Set the callback to start diffusion at time T_{X_v}.
SetCallback¹T_{X_v}; DIFFUSION¹X_v; H⁰⁰;
```

Second, based on results from Dandelion++, this algorithm runs in epochs of approximately ten minutes. Within an epoch, the graph G is fixed, and nodes in the first phase of the algorithm forward all transactions to the same m nodes. These m nodes are re-selected after each epoch to construct a fresh anonymity set. This has the effect of reducing the amount of information that an adversary can acquire when the same node sends multiple transactions. Each transaction within an epoch will follow the same path and reveal minimal topology information. The pseudocode for a single node running an instance of this protocol is given in algorithm 1. Here, we refer to the host node running an instance of the protocol by variable *this*. In the implementation, scheduling of diffusion is done via an asynchronous callback which does not block the node instance.

5.2.3 Randomized Diffusion

Our second proposed algorithm operates in a single phase in a manner similar to a combination of Dandelion++'s first

and second phases. In Dandelion's first phase, nodes forward transactions to neighbours until a transaction reaches a *diffuser* node which then begins the diffusion protocol [8].

Here, each node is able to act as both relayer and diffuser. To begin, the source node v sends X_v to a single neighbour. Subsequent nodes upon receiving a relayed message decide whether they are a relayer or a diffuser based on a combination of their identity and the transaction message X_v. If the hash of this combined value is less than some preset value r, then the node becomes a diffuser. Relayers forward transactions to a single node while diffusers begin the standard diffusion process. Once a node starts diffusion, all subsequent nodes from this point in the graph will diffuse rather than relay messages. As in § 5.2.2, this algorithm can be parameterized to allow making several explicit trade-offs between performance and security. First, it may be useful for relayers to send transactions to m neighbours rather than just one. Second, we can run the algorithm in epochs. After each epoch, nodes can select different neighbours to forward transactions to when they are relayers.

Algorithm 2: RANDOMIZED-DIFFUSION

Data:
X_v : The message to be broadcast.
r : Threshold to decide if the node is a relayer.
m : The number of neighbours to select for relay.
$S_k{}^1u^0$: The private key of node u.
Result: Broadcasts X_v via Randomized Diffusion.)

```
// H_X is the transaction hash of X_v.
H := Hash¹H_X kS_k¹this⁰⁰;
if H < r then
    DIFFUSION¹X_v; H⁰;
else
    // Choose m neighbours randomly.
    neiðhbours  U_m¹η¹this; G⁰⁰;
    // Relay the message to each chosen neighbour.
    for nextHop in neiðhbours do
        RANDOMIZED-DIFFUSION¹nextHop; X_v; r⁰;
    end
end
```

Pseudocode for a single node running an instance of this protocol is given in algorithm 2. The motivation behind this algorithm is the attempt to get rid of the two-phase protocol which partially brings all vulnerabilities of a purely diffusion-based protocol. Our analysis results in § 8 provides quantitative insight into the anonymity properties of this protocol.

6 Attacks

A vast variety of deanonymization attacks have been studied in the literature [1–3, 11, 15]. In [1], Androulaki et al.

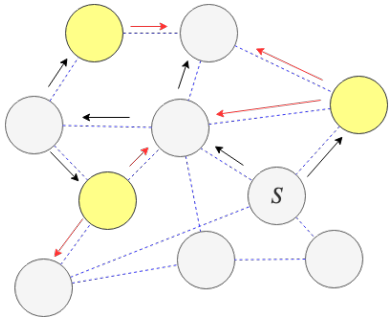

Figure 5. Randomized diffusion starting at source S. Nodes that decide to diffuse and stop relaying are shown in yellow. Black arrows depict relays while red arrows depict diffusion.

show through simulations that about 40% of user profiles can be recovered to a large extent even if they adopt the privacy measures recommended by the original Bitcoin design. In [2, 3], Biryukov et al. design a complete end-to-end method of deanonymizing Bitcoin users. Their method provides successful results even when the clients use Tor. They model the attack in 3 independent phases, (1) disconnecting clients from Tor, (2) learning the network topology, and (3) learning entry nodes of clients when they connect to the network. This attack is based on the fact that whenever a client establishes a connection to one of its entry nodes, it advertises its public address. Hence, if the adversary is already connected to an entry node, with some probability (depending on the number of connections it has) the address will be forwarded to this attacker node. In [11], Lokhov et al. study the problem of estimating the origin of an epidemic spreading on networks of contacts. Other than applications in analyses of infectious diseases and rumour spreading on social media, their methods can also be used to design deanonymization attacks on the Bitcoin network.

In the context of the Bitcoin P2P network, the goal of an attacker is to create an estimator mapping, $\hat{S} : X \to V_H$, based on the set of observation tuples $S = \bigcup_{v \in V_H} S_v$ and the known network graph $\eta^1 V_A; H^0$. Fanti et al. discuss the first-timestamp estimator and maximum likelihood estimator in [9]. Although these methods work well for the eavesdropper model where the adversary knows the complete network graph and maintains multiple connections to each node in the network (including the source), it does not provide interesting results for the attacker in a spy based adversary model, which is more realistic.

6.1 Graph-Learning Attack

The adversary knows the graph construction protocol (the way nodes join and leave the network), but it cannot observe the operation of the protocol outside its own local neighbourhood $\eta^1 V_A; H^0$. The goal of a privacy-preserving spreading

algorithm is to protect the privacy of nodes against an adversary that has learned a significant component of the network graph over time. In ideal operation, the algorithm should provide reasonable resilience guarantees if the path taken by a transaction message in the anonymity set includes at least one honest node.

In this regard, an analysis of the first-spy estimator by Fanti et al. in [8] shows that Dandelion++ is fairly resilient to graph-learning attacks, even in the case where the adversary knows the complete network graph. Whenever nodes shift to a new epoch, the anonymity graph G is reconstructed from scratch. In this epoch, the adversary can no longer use any information learned about G from the previous epoch. Hence, we follow a similar idea of reconstructing anonymity sets in subsequent epochs in our protocols.

Now a natural question arises: why do we need to precompute the anonymity graph and keep it constant for an epoch? We addressed this question briefly in § 5.1.1, and in the next subsection, we describe an attack that focuses on this aspect.

6.2 Intersection Attack

This class of attacks exploits the weak assumption that each user generates exactly one transaction per epoch. As described in [8], when each node creates and relays an arbitrary number of transactions, each spy node has some fixed probability of being the first spy to hear about these transactions. We denote the probability mass function of the first spy nodes for the honest node v by Ψ_v. The probability that a node $u \in V_A$ is the first spy for transactions originating at $v \in V_H$ is denoted by $\Psi_v(u)$. Further, we make the following assumptions.

1. The adversary has complete knowledge of the Bitcoin P2P graph H.
2. For $u, v \in V_H, u \neq v \implies \Psi_u \neq \Psi_v$.

These assumptions provide two advantages: (1) they make the adversary stronger, supporting the robustness of our algorithms, and (2) they make the analysis and simulation easier. Now a two-phase attack can be formed as follows.

The first phase is called *training*, where the adversary simulates the spreading protocol K times for each source node that it wishes to identify. The resulting empirical distribution of first-spies for a given source determines the adversary's estimate of the pmf Ψ_v for each honest node v. The adversary computes such a signature pmf for each candidate source.

The second phase is called *testing*, where the adversary observes $k \ll K$ transactions from a node v and recomputes the pmf signature Ψ for that node based on these k transactions. Next, we match Ψ to the closest $\Psi_v, v \in V_H$ from the training phase. In [8], the authors propose that this can be achieved by minimizing the *Kullback-Leibler* divergence between the two vectors. However, they do not provide any empirical analysis of the same. The minimization based on KL divergence works by finding the closest matching class. This divergence is computed as,

$$D_{KL}(\Psi \| \Psi_v) = \sum_{u \in V_A} \Psi(u) \log \frac{\Psi_v(u)}{\Psi(u)} :$$

Our simulations reveal that KL divergence is not a good metric for empirical evaluation because of the presence of logarithms. In a large enough network, for certain honest nodes, the probability of being the first spy is zero for most adversary nodes which skews the metric against the adversary. The authors in [8] only theorize this attack and do not provide any implementation. Therefore, we use a more popular metric, *cosine similarity*, to measure the closeness between the two pmf vectors. The cosine similarity is given by,

$$C(\Psi, \Psi_v) = \frac{\Psi \cdot \Psi_v}{\|\Psi\| \|\Psi_v\|} = \frac{\sum_u \Psi(u)\Psi_v(u)}{\sqrt{\sum_u \Psi^2(u)} \sqrt{\sum_u \Psi_v^2(u)}};$$

where the sum $\sum_u$ is over all $u \in V_A$. Using this metric, the adversary can try to predict the source of transactions based on the data collected in an epoch as follows.

$$\text{Source} = \underset{v \in V_H}{\operatorname{argmax}} \; C(\Psi, \Psi_v) :$$

6.3 Partial Deployment Attack

It is important to consider the fact that instantaneous deployment of any new protocol on the full network is infeasible. Byzantine nodes can exploit partial deployment of the protocol to perform some serious attacks. For our algorithms to work, we must ensure that the anonymity graph consists of nodes that support the new protocol. Dandelion does not consider this attack [4], but Dandelion++ incorporates a *version-checking* algorithm that checks for nodes that support the protocol while constructing the anonymity graph. Before selecting edges for the graph G, each node first identifies its outbound peers on the P2P graph H that support the new protocol. The rest of the construction is carried out as before on this subset.

6.4 Black-Hole Attack

The purpose of this attack is not to deanonymize users but to stall the network. In the case where an adversary is present in the path taken by a transaction message in the anonymity graph, it can attempt to become a black-hole and stall propagation by not relaying any messages that it receives. However, this is not of much concern for two reasons. First, in the implementation, we always have $m \geq 2$, hence the chances of propagation being stalled are reduced. Second, even if all m of the relay nodes are adversarial, the algorithms have an inbuilt failback mechanism (discussed in § 5.1.3) that enables honest nodes to start diffusion after a preset wait-time interval if they do not receive inventory messages from their uninformed neighbours about a transaction. This ensures

that the whole network receives the broadcast message irrespective of whether a node's neighbours are adversarial.

7 Experiments

We analyze both of our proposed algorithms in a number of ways. Our primary area of interest in this work is enhancing the privacy of nodes in the Bitcoin network so our attention is first focused on studying the efficacy of an adversary that attempts to identify the source of a transaction. However, it is also important that spreading protocols propagate messages quickly, thus we also consider the speed at which our algorithms spread transaction messages.

In our first experiment, we study an adversary performing an intersection attack on a network, as described in § 6.2. We primarily study the effect that increasing the fraction of spy nodes has on the efficacy of the attack, varying this value from 10% to 50% of nodes within a network of 1000 nodes. We keep k set to 10 while running attacks with each K 2 100; 300; 500. For computational tractability, we restrict our adversary to attempting to identify 50 sources for each set of parameters. This experiment is run for the Diffusion protocol, as well as Synchronized Diffusion, and Randomized Diffusion with the chance of diffusion set to 0.01, 0.1, and 0.2.

Our second experiment focuses on the propagation speed of our protocols. For networks ranging in size from 200 nodes to 1800 nodes, we measure how long it takes for a single transaction to spread through 50%, 90%, and 99% of the network. We repeat this in 100 networks at each size and report the average results.

Finally, we also consider the challenges associated with the deployment of our protocols. Partial deployment of spreading algorithms has previously been discussed as a possible attack vector [8]. We run an intersection attack using the previously stated parameters on a network with equal proportions of nodes running each of the three spreading protocols (with Randomized Diffusion having a 0.1 chance of beginning diffusion). We also report the propagation speed in this partial deployment scenario.

In all experiments, the network topology is an Erdős-Rényi random graph model with an average degree of 8 at each node [7]. We believe this provides a reasonable facsimile of the random graph underlying the true Bitcoin network. To run our experiments we make use of the Python simulation framework, `Mesa` [12].

8 Analysis

This section presents and discusses the results of the experiments described in § 7.

8.1 Intersection Attack

The results of our intersection attack analysis are surprising primarily due to their regularity. Figure 6 presents the precision and recall of our spy-based adversary for each spreading protocol and several different fractions of spy prevalence. Remember that precision refers to the ratio of the number of instances in which the adversary correctly identifies a message's source v to the total number of instances in which the adversary guesses a message's source was v. Recall refers to the total proportion of sources that are correctly identified by the adversary. In both cases, a larger value indicates greater success of the adversary at identifying the source of a transaction message.

Most noteworthy in all results is that, in most spreading protocols, a larger fraction of spies does not seem to relate to a more powerful adversary. Diffusion is the exception to this, where the adversary does improve slightly with a larger fraction of spies and has results similar to those found by Fanti et al. [8].

Synchronized Diffusion appears to have very similar resilience to intersection attacks when compared with Diffusion. This suggests that while Synchronized Diffusion may not provide additional security against this form of attack, it does not reduce security either. Given that the protocol is much more tunable than Diffusion, this may be sufficient reason to prefer Synchronized Diffusion.

Random Diffusion, on the other hand, tends to perform more poorly than both other protocols. In particular, with a lower chance of diffusion, the protocol exhibits less resilience to intersection attacks, suggesting that the non-diffusion aspect of the algorithm is much less secure than the diffusion component. This may also provide some indication as to why Synchronized Diffusion is more secure: while there is a component that does not involve diffusion, eventually all nodes participate in diffusion.

All results in Figure 6 exhibit characteristics of noise. Thus, we suggest that these results should be taken as preliminary and more extensive study should be performed in order to fully compare the security afforded by each of these protocols.

8.2 Propagation Speed

Results of experiments measuring the speed at which a transaction spreads through a network are shown in Figure 7. For each protocol, we present the average time taken for 100 transactions to propagate through 50%, 90%, and 99% of the network. There are a few cases which stand out in these results.

First, we note that the Synchronized Diffusion protocol in Figure 7 (b) consistently has the fastest spreading of all protocols we examine. This is to be expected: compared with Diffusion, Synchronized Diffusion in its first phase will send messages more quickly. Thus, when the second phase starts,

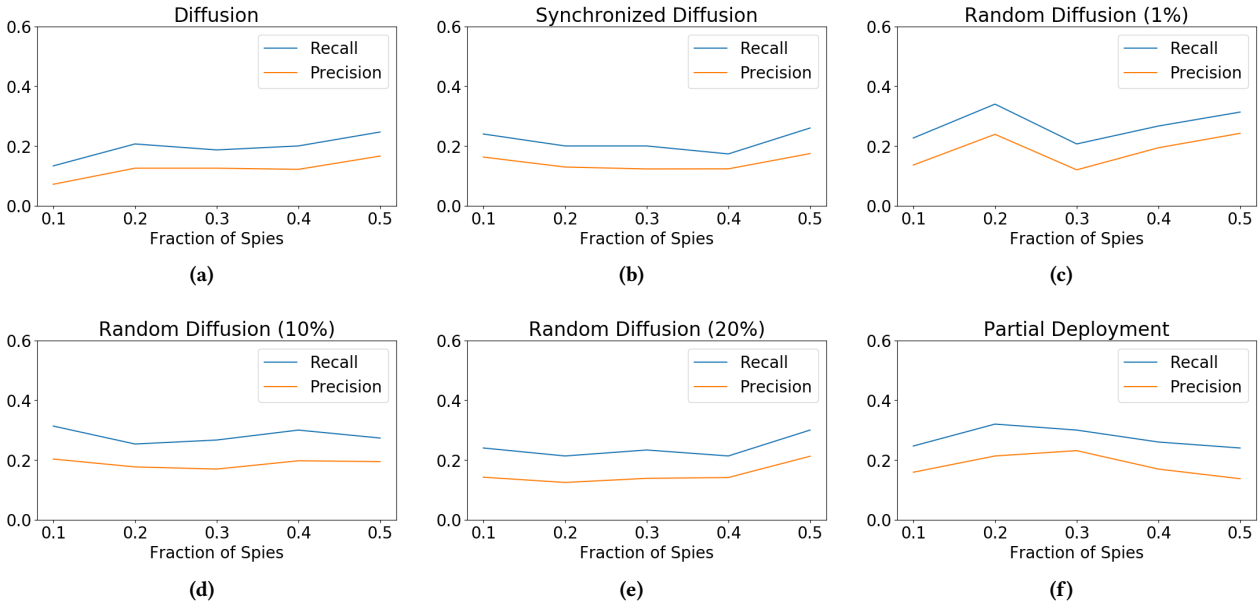

Figure 6. Precision and recall for an intersection attack performed on a network of 1000 nodes with 50 possible source nodes.

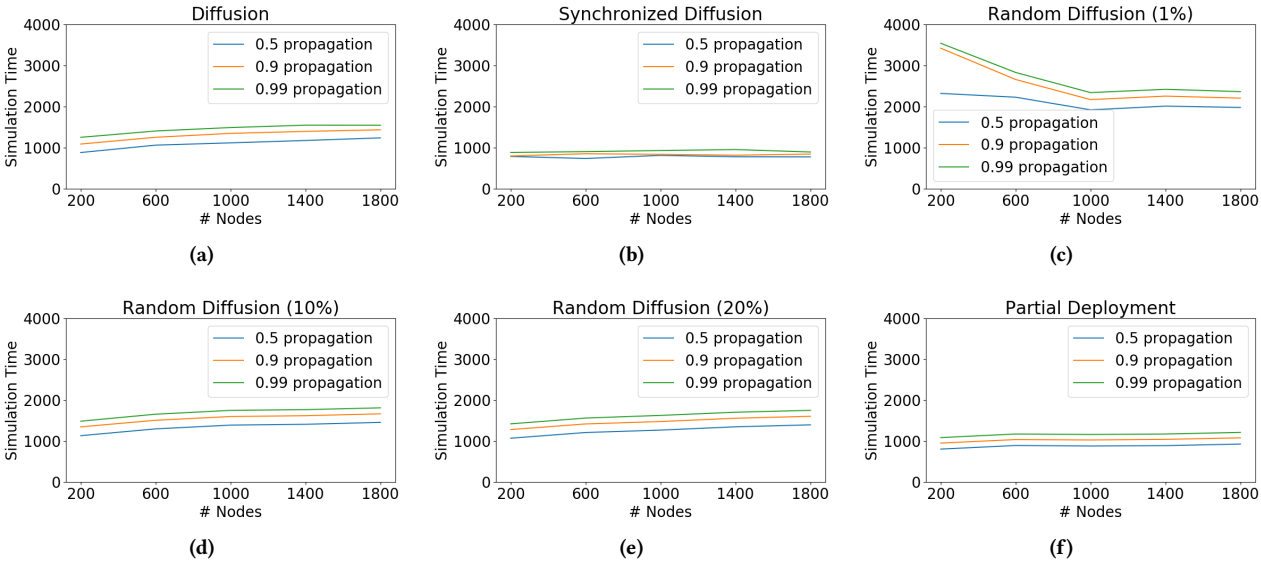

Figure 7. Propagation speeds for a single transaction message using each spreading protocol. Note that time is a measure of simulation duration corresponding to the true time taken by each protocol but does not directly translate to a real time measurement.

many more nodes begin Diffusion. Waiting longer before beginning the second phase, or a shorter average delay in the Diffusion protocol might cause the Diffusion protocol to operate more quickly than Synchronized Diffusion. The reason for Randomized Diffusion not performing as well may be the following: (1) Nodes that do not initially diffuse the message will not later diffuse the message, unlike in Synchronized Diffusion where all nodes eventually run Diffusion, and (2) having a lower chance of diffusing leads to fewer diffusing nodes than other protocols.

It is also interesting to observe the dramatic difference in performance in Randomized Diffusion with a 1% chance of diffusion vs a 10% chance (Figure 7 (c) vs (d)). When nodes are very unlikely to diffuse, the protocol is stuck with nodes

sending to only m neighbours rather than all their neighbours, which greatly reduces speed. This is most evident in the networks with very few nodes where performance is worst with a 1% chance of diffusion but best in all other protocols. Since so few nodes are diffusing, speed is reduced.

8.3 Partial Deployment

The case in which nodes use a multitude of differing spreading protocols is often considered a detriment to security, because it may negate the ability of some protocols to achieve a behaviour that enhances security only when done en masse [4, 8]. In contrast to that idea is an observation by Turing Award winner Barbara Liskov who noted that having many implementations of an algorithm in use reduces the vulnerability of the whole system to software bugs [5]. We cannot test for the presence of bugs in our implementations but this observation suggests that if security is not greatly affected by a partial deployment scenario, then it may actually be better to make use of multiple spreading protocols which will inherently have different implementations and different software vulnerabilities.

Figure 6 (f) and Figure 7 (f) compare the result of our initial test of partial deployment of spreading protocols with the resilience to attack and propagation speed of each protocol. We find that security appears to be very slightly reduced (primarily with the fraction of spies set to 20% and 30%) but is not hugely affected. As might be expected, the time taken for a message to propagate through a network with several spreading protocols lies somewhere between the time taken by each of the three protocols within the partial deployment network. Of course, the time is not lower than Synchronized Diffusion, but it is faster than a network solely using Random Diffusion.

What this indicates is that partial deployment does not greatly increase a network's vulnerability to intersection attacks. If these results extend to other attack methods, this would suggest that protocol designers need not fear that introducing a new spreading protocol will harm privacy.

9 Conclusion

In this paper, we have developed two new algorithms for the spreading of transactions through the Bitcoin network, with the goal of masking the network location of the creator of each transaction. Random Diffusion operates similarly to the existing Diffusion method while Synchronized Diffusion involves an initial spreading phase to generate a large anonymity set, followed by diffusion from a large number of nodes. Synchronized Diffusion is shown to provide similar resilience to intersection attacks from a spy-based adversary with knowledge of the underlying network topology while leading to a significant increase in the speed of transaction propagation through a network. We also show that a network composed of nodes following a variety of spreading protocols does not suffer from greatly reduced privacy or propagation speed.

In order to consider Synchronized Diffusion a fully viable algorithm for deployment within the Bitcoin network, there is additional work to be done. A variety of alternative attacks can be examined, including existing graph-learning attacks as well as new attacks developed specifically to combat this protocol which may, for example, target the first phase of the protocol when nodes are spreading transactions to only a small set of neighbours. As well, the results of tests on our protocol should be compared with the same tests performed using Dandelion++. Further work can also determine the optimal settings for the parameters in Synchronized Diffusion that control spreading in the first phase, and timing the beginning of the second phase of the protocol. However, our initial results suggest that Synchronized Diffusion provides an interesting and distinct alternative to the currently proposed Bitcoin spreading algorithms.

References

[1] Elli Androulaki, Ghassan O Karame, Marc Roeschlin, Tobias Scherer, and Srdjan Capkun. 2013. Evaluating user privacy in bitcoin. In *International Conference on Financial Cryptography and Data Security*. Springer, 34–51.

[2] Alex Biryukov, Dmitry Khovratovich, and Ivan Pustogarov. 2014. Deanonymisation of clients in Bitcoin P2P network. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 15–29.

[3] Alex Biryukov and Ivan Pustogarov. 2015. Bitcoin over Tor isn't a good idea. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 122–134.

[4] Shaileshh Bojja Venkatakrishnan, Giulia Fanti, and Pramod Viswanath. 2017. Dandelion: Redesigning the Bitcoin Network for Anonymity. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 1, 1 (2017), 22.

[5] Miguel Castro, Barbara Liskov, et al. 1999. Practical Byzantine fault tolerance. In *OSDI*, Vol. 99. 173–186.

[6] Christian Decker and Roger Wattenhofer. 2013. Information propagation in the bitcoin network. In *IEEE P2P 2013 Proceedings*. IEEE, 1–10.

[7] P ERDdS and A R&wi. 1959. On random graphs I. *Publ. Math. Debrecen* 6 (1959), 290–297.

[8] Giulia Fanti, Shaileshh Bojja Venkatakrishnan, Surya Bakshi, Bradley Denby, Shruti Bhargava, Andrew Miller, and Pramod Viswanath. 2019. Dandelion++: Lightweight Cryptocurrency Networking with Formal Anonymity Guarantees. *ACM SIGMETRICS Performance Evaluation Review* 46, 1 (2019), 5–7.

[9] Giulia Fanti and Pramod Viswanath. 2017. Anonymity properties of the bitcoin p2p network. *arXiv preprint arXiv:1703.08761* (2017).

[10] Philip Koshy, Diana Koshy, and Patrick McDaniel. 2014. An analysis of anonymity in bitcoin using p2p network traffic. In *International Conference on Financial Cryptography and Data Security*. Springer, 469–485.

[11] Andrey Y. Lokhov, Marc Mézard, Hiroki Ohta, and Lenka Zdeborová. 2014. Inferring the origin of an epidemic with a dynamic message-passing algorithm. *Phys. Rev. E* 90 (Jul 2014), 012801. Issue 1. https://doi.org/10.1103/PhysRevE.90.012801

[12] David Masad and Jacqueline Kazil. 2015. MESA: an agent-based modeling framework. In *14th PYTHON in Science Conference*. 53–60.

[13] Fergal Reid and Martin Harrigan. 2013. An analysis of anonymity in the bitcoin system. In *Security and privacy in social networks*. Springer,

197–223.

[14] Michael K. Reiter and Aviel D. Rubin. 1998. Crowds: Anonymity for Web Transactions. *ACM Trans. Inf. Syst. Secur.* 1, 1 (Nov. 1998), 66–92. https://doi.org/10.1145/290163.290168

[15] Devavrat Shah and Tauhid Zaman. 2012. Rumor Centrality: A Universal Source Detector. *ACM SIGMETRICS Performance Evaluation Review* 40, 1 (June 2012), 199–210. https://doi.org/10.1145/2318857.2254782

[16] Zhaoxu Wang, Wenxiang Dong, Wenyi Zhang, and Chee Wei Tan. 2014. Rumor source detection with multiple observations: Fundamental limits and algorithms. In *ACM SIGMETRICS Performance Evaluation Review*, Vol. 42. ACM, 1–13.