Timing Analysis for Inferring the Topology of the Bitcoin Peer-to-Peer Network

Till Neudecker, Philipp Andelfinger, and Hannes Hartenstein
Institute of Telematics & Steinbuch Centre for Computing
Karlsruhe Institute of Technology, Germany
Email: {till.neudecker, philipp.andelfinger, hannes.hartenstein}@kit.edu

Abstract*—Flooding Peer-to-Peer (P2P) networks form the basis of services such as the electronic currency system Bitcoin. The decentralized architecture enables robustness against failure. However, knowledge of the network's topology can allow adversaries to attack specific peers in order to, e.g., isolate certain peers or even partition the network. Knowledge of the topology might be gained by observing the flooding process, which is inherently possible in such networks, and performing a timing analysis on the observations. In this paper we present a timing analysis method that targets flooding P2P networks and show its theoretical and practical feasibility. A validation in the real-world Bitcoin network proves the possibility of inferring network links of actively participating peers with substantial precision and recall (both $\sim 40\,\%$), potentially enabling attacks on the network. Additionally, we analyze the countermeasure of *trickling* and quantify the tradeoff between the effectiveness of the countermeasure and the expected performance penalty. The analysis shows that inappropriate parametrization can actually facilitate inference attacks.

I. Introduction

A number of today's highly distributed internet services employ public Peer-to-Peer (P2P) networks as their main communication infrastructure. One of the challenges in P2P networks is the robustness against failure and adversarial attacks. The network's topology can be the key to both preventing service disruptions as well as performing successful attacks on the network. From an attacker's point of view, knowing the topology of a network can be advantageous, for example if the attacker wants to split the network by performing Denial-of-Service attacks on selected network nodes. Additionally, the anonymity of users might be an application's goal (e.g., Tor), which can be compromised if an attacker is able to reconstruct communication paths.

The electronic currency system Bitcoin [14] uses a P2P network to transmit information on financial transactions through the network in order to reach consensus among all participants on the set of accepted transactions. Information exchange is performed using a gossip protocol: if a peer receives a new transaction, it checks the validity of the transaction and rebroadcasts it to its neighboring peers. An attacker could observe this information propagation process in the network in order to infer connections in the P2P network or link transactions to the originating IP address. This knowledge could be used to perform precise attacks on the network's topology (e.g., eclipsing attack [7], [9]) or as a basis for deanonymization of users.

In this paper, we present a timing analysis method for inferring a flooding network's topology using information obtained by observing the flooding process. In addition, we apply this method in a proof of concept to the real Bitcoin network. First, the proposed method is applied on the basis of an analytical information propagation delay model and validated via simulations. Additionally, the tradeoff between the effectiveness and the performance penalty for the *trickling* countermeasure (i.e., deliberately delaying the forwarding of messages) is quantified. For the real-world validation, the proposed method is then applied using an (empirical) information propagation delay model of the Bitcoin P2P network.

The theoretical and practical contributions of this paper are as follows:

- A network topology inference method using an analytical model for information propagation delay in flooding networks as a basis; validated in simulation.
- A quantification of the generally existing tradeoff between effectiveness and performance penalty for *trickling*.
- A proof of concept of the proposed approach in the real network based on an empirical information propagation delay model of the real-world Bitcoin P2P network.

Although this work has a focus on the Bitcoin network, it is applicable to a wide range of flooding P2P networks including many blockchain based systems [5]. We assume that messages are flooded through the whole network and that each message is uniquely identifiable by all peers. We also assume an *open* network, i.e., each peer is able to connect to arbitrary peers of the network and receives the propagated information from the connected network peers.

II. Related Work

Several previous works aim at extracting information on connections between peers in the Bitcoin P2P network. *Coinscope* [12] uses a client implementation specific behavior to extract the topology of the network. The Bitcoin protocol allows asking peers for IP addresses of other peers in order to establish connections to these addresses. The reference client used to leaked information on neighboring peers by setting the LASTSEEN field in responses to a specific value, if a connection to that peer existed. This leak was removed by March 2015[1], making this technique not feasible anymore.

[1]https://github.com/bitcoin/bitcoin/commit/9c2737901b5203f267d21d728019d64b46f1d9f3

Another approach is to announce made up *marker IP addresses* to other peers in order to infer connections. [4]. Additionally, a technique to identify non-listening peers (e.g., peers behind NATs) by identifying the set of entry nodes was presented. Koshy et al. [10] mapped Bitcoin identities (i.e., public keys) to IP addresses by observing anomalous relaying behavior. All previous approaches have in common that they either use some kind of side channel (e.g., the IP address exchange) or rely on specific, identifiable behavior. The technique presented in this paper, however, makes use only of information that is inherent to a flooding network, i.e., the timing of received messages. Although this paper focuses on the Bitcoin network, numerous similar applications of flooding P2P networks are possible [5]. It has already been shown that attacks on the network such as eclipsing single peers [7], [9] or partitioning the network [15] are possible and information on the network topology can be crucial.

Timing analysis attacks on anonymity-providing networks like Tor have been extensively studied. A common attacker model is the global-passive adversary (GPA, e.g. [13]), which is able to observe the inter-packet intervals on all links between nodes of the network. This attacker model is similar to the one used in this work, as an adversary that participates in a flooding network receives all messages from all of its neighbors and, therefore, can reconstruct message flows.

One difference to the assumptions made in our work is that with onion routing applied, an attacker is not able to link input packets with output packets, as these appear indistinguishable to the attacker. Firstly, this prevents our proposed method from working, secondly, this also allows for other countermeasures. For example, delaying and reordering the outgoing packets is a sound countermeasure against timing analysis if the adversary is not able to link incoming and outgoing packets. Reordering plaintext messages carrying a unique ID (as in Bitcoin) does not prevent the linkage of packets. The same goes with dummy packets that are often proposed for anonymity-providing systems. When focusing on a single mix only, the mix needs to delay packets according to a chosen delay function so that an attacker cannot link incoming packets to their corresponding outgoing packets. In [6] it was shown that the delay function should be exponential. The scenario considered in this paper can be seen as a *dual* scenario to this single mix case: our attacker is able to link incoming and outgoing packets, but there can be several possible delay functions applied to each stream of data and the attacker has to identify the applied delay function. The delay function then represents the number of hops in the network.

III. Problem Statement & Procedure

We will now give the problem statement before describing the procedure and assumptions in detail.

Given: A flooding P2P network with unknown topology. We connect to a large share of the network's reachable peers and observe the time at which messages are received from each of the connected network peers.

Sought: The logical topology of the P2P network, i.e., an answer to the question whether a direct link between any two peers of the network exists or not.

Procedure: We use a propagation delay model of the network and compare the observed propagation delay to the expected delay according to the model, depending on whether a connection exists or not.

We will now discuss several aspects in detail.

Network: We assume a network, which imposes no limitation on the number of connections a peer can maintain. This implies that a single peer (*monitor*) controlled by us can connect to a large share of the network's reachable peers. There are likely still peers in the network that the monitor node is not connected to, however, we restrict the network we reconstruct to the subgraph containing the peers the monitor is connected to. Possible countermeasures could prevent this, e.g., by making the establishment of connections expensive.

Flooding Protocol: The originator of a message broadcasts the message to its neighbors. The neighbors check upon reception of a message whether they have already received the message by checking the message's unique ID. If a message is new to a peer, it rebroadcasts the message to its own neighbors, excluding the peer it received the message from. This way, the message gets flooded through the whole network. This protocol represents a very simple and commonly used gossip protocol, e.g., by Bitcoin.

Observations: The monitor observes the arrival of messages from all connected peers. For each message, a set of tuples *(reception time, sending peer)* is observed – one tuple for each forwarding peer. Intuitively, the reception times in the observations caused by one message correlate with the network topology. However, small reception time differences in two observations do not automatically imply proximity in the network topology, as these peers might have received the message via different paths and coincidentally forwarded the message at the same time to the monitor. It is crucial to only use those time differences that arise from peers on the same transmission path, i.e., time differences with a causal (sender-receiver) significance, and not from parallel actions. As a sender-receiver connection exists between the *originator* (i.e., the peer that initially broadcasts a message through the network) of a message and all other peers, our timing analysis method only uses those time differences. We assume that it is possible to identify the originator of a message: assuming an active attacker model, the attacker could create messages and transmit these messages to one single peer only. This peer would then rebroadcast the message to its neighbors and, from our perspective, can be considered as the originator of the message. Assuming a passive attacker, it is still possible to statistically guess the originator of a message based on the observed reception times.

Delay Model: Based on observations, the latency from our own monitor node to other nodes can be estimated very well. Latencies between foreign peers can be estimated, e.g., based on distance between peers. We are also aware of the client behavior of the network's peers, i.e., whether they instantaneously rebroadcast messages or apply some kind of trickling mechanism that reduces the sending rate.

In this paper, two delay models and two methods for comparing the observation to the model are used: in Section IV we apply a Maximum Likelihood Estimation for comparison, whereas in Section V a hypothesis testing approach is used. An analytical propagation delay model (see Appendix) is used for the analysis of the trickling countermeasure in Section IV, and a simulation-based delay model is used for the real-world validation in Section V.

IV. Topology Inference Model for Flooding P2P Networks

We will now show how to infer the topology of a network based on the comparison of observations of the information propagation delay in the network and a propagation delay model. For this, we will first formalize the problem considered before the approach is presented and validated. Finally, limits of the approach are shown and the *trickling* countermeasure is discussed.

A. Formalization as a Classification Problem

Every time a message is forwarded to an attacker's monitor, a tuple *(reception time tr, sending peer v)* is created at the monitor. Therefore, for each unique message m, the attacker observes a set of tuples $O_m = \{(tr_0, v_0), (tr_1, v_1), ...\}$. As we assumed the attacker to be aware of the latency from the monitor node to other peers, the attacker can subtract this latency from the *reception time* and get an estimate of the *sending time*: $O'_m = \{(t_0, v_0), (t_1, v_1), ...\}$, where the first tuple (t_0, v_0) represents the message's sending by the originator (v_0) of the message.

We convert the absolute timestamps of O'_m to time differences relative to the creation time t_0 ($\delta_1 = t_1 - t_0, \delta_2 = t_2 - t_0, ...$). Each of these time differences δ_i is a sample of the delay between the originator of the message and the peer v_i, which forwarded the message to the attacker's peer. Grouping all time differences of all messages by these two peers results in a set of measured delays for each pair of peers $\Delta_{v_1,v_2} = \{\delta_1, \delta_2, ...\}$. The set contains one time difference for each message that was created by v_1 or v_2. Therefore, the set is empty for all pairs of peers both of which did not create a message during the observation period.

We will now focus on how to estimate the shortest path length C_{min} between two peers in the network (i.e., the shortest sequence of edges between both peers) based on the observations made.

B. Inferring the Shortest Path Length

The following estimation compares the observations made to the analytical propagation delay model described in depth in the Appendix. Here, we will only depict the parameters of the model and how it can be used. The parameters of the modeled network are the number of peers and the probability of existence of each possible connection. The network in our delay model is assumed to match a random graph model [8].

Using the model, we can calculate the a priori probability that the shortest path C_{min} between two randomly chosen peers has length l (i.e., $P(C_{min} = l)$). The discrete random variable D models the propagation delay between two randomly chosen peers (discretized to e.g. milliseconds[2]). The model enables us to calculate the probability of observing a specific delay δ: $P(D = \delta)$. It also allows calculation of the probability of observing a specific delay δ assuming that the shortest path length between sender and receiver equals l: $P(D = \delta | C_{min} = l)$.

We are now looking for a method to assess how likely it is to observe a specific set of time differences, depending on the shortest path length between the two observed peers. A relationship between the unknown shortest path length C_{min} and the observed time difference δ is given by $P(D = \delta | C_{min} = l)$.[3] Evaluation of this formula for each possible shortest path length and all observations allows a comparison between the resulting probabilities and lets us decide, which shortest path length has the maximum likelihood.

The likelihood function for a set of observed time differences Δ_{v_1,v_2} and a length of the shortest paths l follows from the definition of a likelihood function as

$$L(C_{min} = l | \Delta_{v_1,v_2}) = P(C_{min} = l) \cdot \prod_{\delta \in \Delta_{v_1,v_2}} P(D = \delta | C_{min} = l).$$

The maximum likelihood estimation of the shortest path length between v_1 and v_2 is computed by selecting the largest likelihood among all shortest path lengths, resulting in

$$\hat{l} = \arg\max_{l} L(C_{min} = l | \Delta_{v_1,v_2}).$$

For an asymptotically large number of observations, $\hat{l}$ converges to the real value of C_{min}. However, the estimated shortest path length can differ from the real shortest path length, if, for instance, the observation contains only a few values and many of them are outliers. Therefore, some measure of confidence in the guess is required. The quotient of the likelihood function of $\hat{l}$ and the sum of all likelihood functions gives the probability that the guess is in fact correct (*certainty*)

$$P(C_{min} = \hat{l} | \Delta_{v_1,v_2}) = \frac{L(C_{min} = \hat{l} | \Delta_{v_1,v_2})}{\sum_l L(C_{min} = l | \Delta_{v_1,v_2})}. \quad (1)$$

Actually, this equation can be calculated not only for $C_{min} = \hat{l}$, but also for all other values of C_{min}, denoting the probability that each C_{min} is in fact correct. This corresponds to assigning probabilities to each shortest path length.[4]

[2]Although a delay could be modeled as a continuous random variable, we opt for a discrete model to enhance readability and closely match our simulation model.

[3]This problem can be formulated as a very simple Hidden Markov Model (HMM): Each hidden state represents one minimum path length C_{min} between two peers in the network. The observable states of the HMM are the time differences δ. The transition probabilities from each hidden state l to the observable states equal the probability of observing a delay of δ assuming a minimum path length of l: $P(D = \delta | C_{min} = l)$.

[4]The shortest path length C_{min} can also be seen as a probabilistic information source, which outputs l with a probability of $P(C_{min} = l | \Delta_{v_1,v_2})$. The entropy of this information source then equals the uncertainty in the estimation. The difference $P(C_{min} = l) - P(C_{min} = l | \Delta_{v_1,v_2})$ is an upper bound for the information content of the observation Δ_{v_1,v_2}.

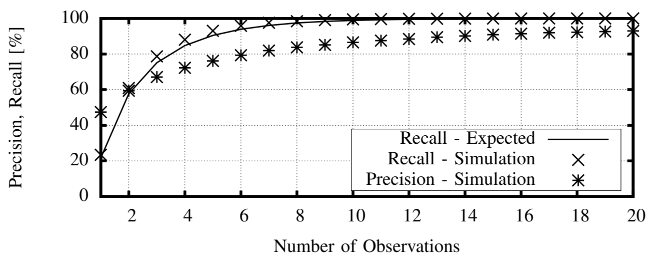

Fig. 1. Precision and recall in a simulated network wrt. the number of observations per pair of peers.

C. Validation

We will now show the effectiveness of the proposed timing analysis by examining the resulting error rates. A flooding network was simulated that generated the observations as input for the timing analysis. The timing analysis resulted in a guess which edges of the network exist. By comparing the estimate to the simulated network, we can judge the quality of the proposed technique.

The estimation can lead to two kind of errors: false positives and false negatives. A false positive occurs if a specific edge is postulated although it does not exist. A false negative occurs, if an existing edge is not detected by the analysis. Obviously, the more observations are in Δ_{v_1,v_2}, the less likely are both kinds of errors, as each observation originates from the *correct* distribution.

Commonly used measures for the quality of classifiers are *precision* and *recall*. Precision is defined as the number of true positives divided by the sum of true and false positives (i.e., all detected elements). Recall is defined as the number of true positives divided by the total number of relevant elements (i.e., the number of elements that should have been detected). Fig. 1 shows how precision and recall increase with the number of observations in the simulation performed. Additionally, the calculated expected recall is depicted. The recall converges quickly to 100 %, whereas the precision rises much slower and reaches 90 % after 12 observations. Both, expected and experimental recall match very well.

Although the error rates look extremely promising, it should be noted that the simulation experiment makes some idealized assumptions that cannot be matched in the real world: firstly, the delay distribution as assumed by the attacker equals the real delay distribution used in the simulation. In reality, an adversary has to estimate the delay distribution, which will only be an approximation. Additionally, the network and the delay distribution is static in the simulation, whereas churn and jitter are known to occur in real-world networks. We will leave a sensitivity analysis of the delay distribution estimation used by the adversary as future work and give a proof of concept of the proposed method in the real Bitcoin P2P network in Section V.

D. Limits & Countermeasures

As just shown, there exists a relationship between the number of observations and the quality of the estimation.

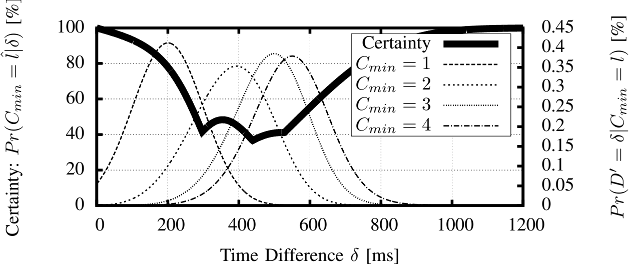

Fig. 2. Conditional delay distributions and certainty wrt. to observed delay - Scenario: $6,000$ nodes, 16 connections per node on average, single-hop latency distribution according to a normal distribution ($\mu = 200\,\text{ms}, \sigma = 100\,\text{ms}$).

Intuitively, the shape and especially the overlap of the conditional delay distributions for each shortest path length also affect the estimation's quality: highly overlapping delay distributions impede correct estimations, whereas observations from non-overlapping distributions are easy to map to shortest path lengths. We will first illustrate this relationship before analyzing the effectiveness and tradeoffs of a countermeasure against timing analysis.

Fig. 2 shows the probabilities $P(D = \delta|C_{min} = l)$ depending on the observed time difference δ for $C_{min} \in \{1, 2, 3, 4\}$ for a given scenario. Additionally, the *certainty*, as calculated by Equation 1, of such an observation is shown. It can be seen that observing small time differences (below 200 ms) leads to the highest certainty, because of the fact that these delays result from $C_{min} = 1$ with overwhelming probability. As the conditional probabilities overlap between 200 ms and 600 ms, such observations do not help much in reconstructing the network, as various minimum path lengths are almost equally likely. For delay differences higher than 600 ms the certainty rises again. However, this is only because of the limited considered shortest path length of 4 for this calculation. Larger shortest path lengths result in transition probabilities similar to $C_{min} = 4$ but slightly shifted (the difference between $C_{min} = 3$ and $C_{min} = 4$). We will exploit the fact that small delays cause a higher certainty in the real-world validation in Section V.

A common countermeasure against timing analysis is deliberately delaying the forwarding of messages (*trickling*) instead of instantaneously rebroadcasting all messages (e.g., used in Bitcoin). The idea is to increase the overlap in the conditional delay distributions, which reduces the certainty and hinders the adversary from reconstructing the network. However, this also increases the overall information propagation delay in the network, which is not desirable for most applications, for example Bitcoin, where reaching consistency is the main purpose of the network.

The existence of a general tradeoff between traffic analysis resistance, performance, resistance to catastrophic DoS and bandwidth cost has already been identified in [1]. We will now apply the propagation delay model and the timing analysis technique presented in order to quantify the tradeoff between traffic analysis resistance in terms of precision and recall, and

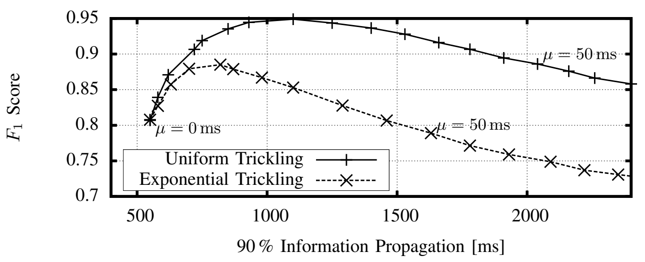

Fig. 3. Tradeoff between low consistency delay and timing analysis resistance when applying *trickling*. Trickling is performed using *a)* a uniform distribution of varying size, and *b)* an exponential distribution with varying mean. The x-Axis shows the overall delay until 90 % of peers received the propagated message. F_1-Score after 4 observations.

performance in terms of *consistency delay*, i.e., the delay until a message has been flooded through a certain share of the network. We assume that the analyzing adversary is aware of the fact that trickling is performed and how it is parametrized.

Fig. 3 illustrates the effect of applying the trickling countermeasure on the consistency delay and the precision and recall of an adversary for our exemplary scenario. Precision and recall are combined into the F_1-Score, which is a common measure of accuracy and is calculated as $F_1 = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$. A perfect predictor results in $F_1 = 1$, whereas worse predictors result in smaller F_1-Scores. In Fig. 3 trickling is performed by randomly delaying the redistribution of messages for a certain length of time according to a) a uniform distribution and b) an exponential distribution. Both distributions were parametrized with a set of mean values: at $\mu = 0$, effectively no trickling happens, therefore, the result corresponds to what has been shown in Fig. 1. With increasing mean values, both the time until information is propagated to 90 % of peers, as well as the F_1-Score increases. Only for higher delays, the F_1-Score starts to decline.

Although one might expect trickling to always have a positive effect on traffic analysis resistance, the results show that trickling, if inappropriately parametrized, can actually reduce the resistance to traffic analysis, i.e., it can improve an attacker's precision and recall. This is caused by trickling's negative effect on propagation speed. Its goal is to increase the overlap of the transition probability distributions. On the one hand, trickling broadens the shape of the conditional delay distribution, on the other hand it also increases the difference between the mean of the different distributions. For example, a constant trickling distribution that delays all packets by one second makes it much easier for an attacker to guess the packet's hop-count, as the constant delay only increases the gap between the different conditional probabilities, but does not broaden each distribution's shape. Fig. 3 also shows that trickling according to an exponential distribution can increase the timing analysis resistance if properly parametrized, whereas trickling with a uniform distribution has a negative effect for the parameters considered.

Please note that although trickling may be detrimental for preventing timing analysis, it also can have positive effects on the timing analysis resistance that were not discussed here. For example, trickling makes it substantially harder for an attacker to identify the originator of a message in the network. The attacker cannot decide whether the peer it received the message from first created the message or just relayed the message. An attacker can, however, either statistically guess or use an active approach where the attacker creates messages and transmits them to one single peer only.

V. Proof of Concept: Bitcoin Network Timing Analysis

We already showed that the proposed timing analysis method is feasible in theory and simulation under idealized conditions. In order to analyze its real-world feasibility, we will now apply the proposed method to the Bitcoin P2P network. After introducing the fundamentals of Bitcoin, a mapping between the proposed models and the Bitcoin scenario will be established, e.g., by parametrization. Finally, the results of a real-world ground truth validation are presented.

A. Fundamentals: Bitcoin Protocol

The Bitcoin network is a peer-to-peer network [14] currently consisting of 5,500-7,000 (depending on source) reachable and an unknown number of non-reachable unique peers. The main purpose of the network is to broadcast *blocks* and *transactions* through the network. A transaction transfers bitcoins from one or more source addresses to one or more destination addresses. In order to create a transaction, a user has to sign the transaction with the private key corresponding to the source address. A block is formed in the process of mining and contains a set of transactions. We omit the details of mining and transactions here, but emphasize that the P2P network floods blocks and transactions through the network in order to reach consistency.

Flooding is implemented as a three-step process: peers that receive a new transaction or block announce the hash value of that object, which serves as an identifier, to their neighbors using an inventory (`INV`) message. The informed neighbors check whether they have already received the announced object, and can request it using a `GETDATA` message. The transaction or block is then sent using a `TX` or `BLOCK` message, respectively. In order to perform a timing analysis, a peer connects to a large number of other peers and monitors the reception of `INV` messages. This peer does not need to request the data objects itself and it should not send `INV` messages to neighboring peers, because peers keep track of which data objects their neighbors already have and do not issue `INV` messages to peers that already have the data object in question [7].

B. Bitcoin Network Model

The Bitcoin network protocol matches our definition of a flooding protocol from Section III. For parametrization, we need to approximate three distributions:

1) Network Latency Distribution: For the analytical model we assumed that the network delay between all peers in the network follows one single, known latency distribution. In the Bitcoin network, peers are geographically distributed and

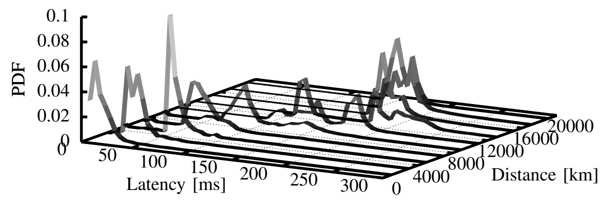

Fig. 4. Latency distribution broken down by geographical distance between measurement node and foreign peer. Binsize = 2000km.

connected through links of various, unknown, qualities. Hence, for each pair of peers a latency estimate based on known parameters is required.

2) Client Delay Distribution: For the analytical model we assumed that clients immediately forward messages without accounting for any possible delay, e.g., due to computation time. The Bitcoin reference client *bitcoind*[5], which is used by the vast majority of peers, delays rebroadcasting due to the polling software architecture as well as due to *trickling*, which randomly delays forwarding and tries to *protect privacy*, as the source code comment says.[6]

3) Node Degree Distribution: For the analytical model we assumed a random graph network model. As previous studies have shown [12], the node degree distribution is much more heavy-tailed.

We will now discuss how each of the distributions has been modeled and how the resulting distribution looks.

1) Network Latency Distribution: It is easy to measure the latency distribution from one location to other peers, but much harder to estimate the latency between two foreign peers. Approaches like iPlane [11] provide a latency estimate by modeling the Internet's connection structure. However, predictions were only possible for a subset of IP addresses that participated in the Bitcoin P2P network. Additionally, iPlane only estimates a mean latency and not a latency distribution. Therefore, we estimated the latency distribution between foreign peers based on our own measurements and the geographical distance between these peers.

We used a measurement node that establishes connections to network peers by connecting to announced IP addresses and monitors the latency to these peers. The latency is measured using ICMP pings or TCP SYN pings for those hosts that do not respond to ICMP messages.[7]

The ICMP/SYN ping measurements result in a precise estimate of the network latency distribution from our monitor node to each connected network peer. This estimate can be used to estimate the sending time of messages from the reception times (cf. Section IV). Figure 4 shows the observed delay distribution from the monitor node to all peers broken down by

[5]https://github.com/bitcoin/bitcoin

[6]https://github.com/bitcoin/bitcoin/blob/dd1304ec216c7d4bdb302195e184b15503819f67/src/main.cpp#L5543

[7]A TCP SYN packet is sent to a host and the time until either a RST or a SYN/ACK packet is received is measured as a round-trip time. This method is commonly used by network scanners, such as nmap, for example.

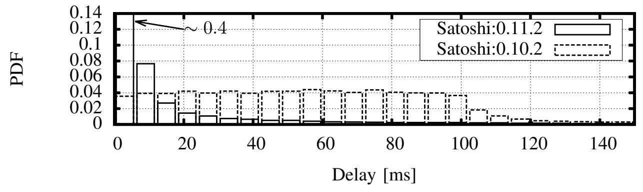

Fig. 5. Unintentional client delay for the bitcoin reference client version 0.10.2 and 0.11.2.

the geographical distance to the remote peer. The correlation between the distance between peers and the resulting latency distribution can be clearly seen from the plot. Therefore, we chose to estimate the latency between two foreign peers solely on the distance between them. The distance between peers can be easily calculated after obtaining the location of the peer based on its IP address using the Maxmind GeoIP Database.[8] Although errors in the localization may occur, the sensitivity of our approach to such errors is very low.

We acknowledge that this model has certain limitations: the model does not account for latency introduced by temporal behavior such as link saturation. It also cannot make any statements on the connection quality between the specific user and its ISP, as the empirical model relies solely on the distance. Furthermore, the distance between two peers on the earth's surface may not reflect the routed distance, e.g., through submarine optic fiber cables.

2) Client Delay Distribution: We define the time between the reception of a message and the forwarding of the message to its peers as the *client delay*. These delays can be unintentional, such as computation delays or delays introduced by blocking behavior of the software architecture. However, the client may also intentionally delay forwarding of messages in order to impede timing analysis or reduce network load. We will first model these unintentional delays empirically and model the intentionally introduced delays in the Bitcoin network analytically.

For the empirical model of the unintentional client delays we leverage the possibility of sending two different types of ping messages to peers: those that are processed by the operating system's network stack (ICMP and SYN as used above) and those that are processed by the application code (Bitcoin protocol `PING`). Although processing by the operating system can be severely delayed, it usually takes only a negligible amount of time in the microsecond range [3]. By subtracting the averaged network latency from the observed delay for answering a Bitcoin `PING`, we receive an estimate of the application's processing delay.

Fig. 5 shows the observed unintentional client delay distributions for clients using one of two selected client versions. Whereas the older client version (0.10.2) employs a blocking message processing architecture with a fixed $100\,\mathrm{ms}$ sleep, the newer version (0.11.2) uses an event driven message processing that minimizes delays. Our delay estimates clearly reflect these changes; especially the behavior of the older

[8]http://dev.maxmind.com/geoip/

version is correctly modeled with a uniform delay distribution between 0 and 100 ms. Please note that although most delays are less than 100 ms, a substantial number of much longer delays were observed: More than 7.5 % of all delays were longer than one second, 2.6 % of all delays were even longer than 10 seconds.

Two different causes for these very long delays could be identified: on the one hand, there are peers that constantly have a very high client delay (i.e., more than 1 second). These peers are probably permanently overloaded. On the other hand, there are peers that exhibit small client delays in most cases, but long delays for a few measurements only. These peers are probably only temporarily busy, e.g., they could be verifying a block at this particular point in time. In order to model this dependency, we use our measurements to generate one client delay distribution per peer instead of one single averaged distribution for all peers.

The Bitcoin reference client implements a trickle mechanism to intentionally delay the forwarding of `INV` messages to neighboring peers. The following description of the trickling mechanism applies to version 0.10.x. We will later discuss changes to current versions. Upon reception of a transaction, the client randomly decides whether to apply trickling on this transaction. Trickling is performed for 75 % of all transactions, 25 % of all transactions are immediately forwarded. Transactions that were not immediately forwarded to all neighboring peers are forwarded to one neighbor at a time during each of the following message processing cycles, which are executed every 100 ms. With the source code changes in version 0.11.x, the fixed 100 ms interval vanished, effectively increasing the number of processing cycles and, therefore, accelerating message forwarding even with trickling. As of version 0.12.x, this behavior was changed again so that trickling is performed on a per-host basis with a Poisson distribution.[9] As this version was released after the experiments were performed, we did not model it here but emphasize, that the changed trickling mechanism can be modeled as shown and does not invalidate this timing analysis (cf. Section IV-D).

3) Node Degree Distribution: Although most Bitcoin network peers maintain a relatively small number of connections, some peers maintain an extremely large number of connections [12]. This node degree distribution not only affects the a priori probabilities required for the timing analysis (cf. Section IV), but also the overall propagation speed in the network. For instance, a single peer that is connected to all other peers in the network may substantially increase the propagation speed of flooded messages. The proposed inference approach could show the real node degree distribution to us, however, we require the distribution in order to perform the inference in the first place.

In order to approximate the node degree distribution in the Bitcoin network, we simulated the network and compared the information propagation as observed in the real network to the simulated information propagation. Minimizing the deviation (i.e., the squared error sum) between these two information propagation delay distributions for different node degree distributions results in an approximation of the correct parametrization.

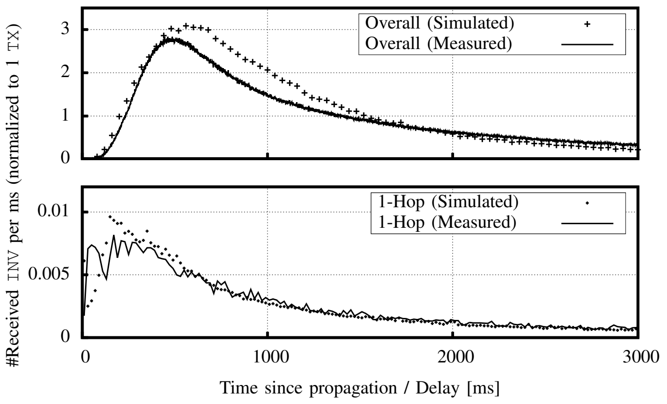

Fig. 6. Comparison between measured and simulated `INV` propagation delay as histogram data; limited to direct neighbors of originating peer (bottom) and for the complete network (top). Both networks parametrized with $\gamma = -2.3$. Normalization: shown values correspond to the creation of *one* transaction.

Known parameters are the number of reachable peers, their geographical distances from each other and, therefore, an estimate of their latencies, the distribution of client delays and the assumed behavior of all clients. Unknown parameters are the node degree distribution and the number of non-reachable nodes that are connected to reachable nodes. What is also unknown is the number of peers with anomalous behavior and this behavior itself. An example of such behavior is the *BitcoinRelayNetwork*[10], which transmits transactions and blocks in one step instead of relying on the three-step protocol described earlier.

In order to determine these unknown parameters, we first approximated the node degree distribution of the reachable nodes: we generated transactions in the real Bitcoin network and observed how fast the transactions were relayed by the neighbors of the generating peer. As the generating peer was controlled by us, we knew the neighboring peers, which were randomly chosen from all network peers. The measured propagation delay is independent of the number of non-reachable nodes and only depends on the node degree distributions of the neighboring peers, which affects the trickling mechanism.

As many similar networks were shown to resemble a scale free network [2] and previous data [12] also supports this assumption, we assume that the node degree distribution follows a power law: $P(k) \sim k^{-\gamma}$. However, as the client is configured to establish eight outbound connections, we modified the minimum node degree to be eight as well, while scaling the node degrees for eight or more connections.

The bottom part of Fig. 6 shows the measured propagation delay distribution between the creation time of a transaction and the reception of the corresponding `INV` messages by our monitor node from the neighbors of the originator of

[9]https://github.com/bitcoin/bitcoin/commit/5400ef6bcb9d243b2b21697775aa6491115420f3#diff-7ec3c68a81efff79b6ca22ac1f1eabba

[10]http://bitcoinrelaynetwork.org

the transaction (*1-Hop*). Additionally, the simulated delay distribution for the same number of reachable peers and a node degree distribution following the adapted power law parametrized with $\gamma = -2.3$ is shown. Various values for the parameter γ were simulated with $\gamma = -2.3$ resulting in the smallest deviation between measurements and simulation, i.e., showing the least square error sum between the two distributions.

In the Bitcoin network there are at least two more known *classes* of non-reachable peers: standard clients that maintain a low number of connections (i.e., 8) and peers that are specifically used to maintain a high number of connections and perform fast message forwarding. The first class of peers could represent a standard user behind a NAT, whereas the second class of peers could be run by a mining pool, for example. We added peers of these two classes to our model and, again, varied the parameters (i.e., the number of peers per class and the number of connections held by the second class of peers) and compared the simulated information propagation delay distribution to the measured one. In contrast to the first measurements, that were restricted to direct neighbors of the generating peer, the information propagation delay now covers the complete propagation. Although additional peers and connections were added to the model, the node degree distribution of the reachable peers was not affected, as connections to additional peers substituted already existing connections in the simulation.

The upper part of Fig. 6 shows the measured propagation delay distribution of the whole network compared to the simulated one with 16,000 non-reachable standard peers and 70 non-reachable peers with 200 connections each. Again, we simulated a broad range of values and the parametrization shown resulted in the smallest square error sum between measurements and simulation. Although this specific parametrization represents the information propagation in the real network well, one cannot draw the conclusion that the real network consists exactly of these types of peers in these quantities. It is possible that there are numerous other parametrizations (including additional peer classes or anomalous behavior) that also lead to the same information propagation delay distribution. Probably, the remaining deviation in the overall propagation delay is also caused by anomalous clients that were not modeled here.

C. Application to the Real-World Network

We parametrized the timing analysis approach presented in Section IV with the model of the Bitcoin P2P network and validated its feasibility in the real network. We will now first describe how the parametrization was done, before validation results are shown and discussed.

As the analytical model presented assumes a random graph model, it cannot be directly used to obtain the required probabilities $P(D = \delta | C_{min} = l)$ (i.e., which shortest path length results in which delay distribution) for the real network. However, these probabilities can also be obtained by performing simulations of the network and monitoring not only the overall propagation delay (Fig. 6), but the propagation delay depending on the shortest path length (which is a known property in the simulation).

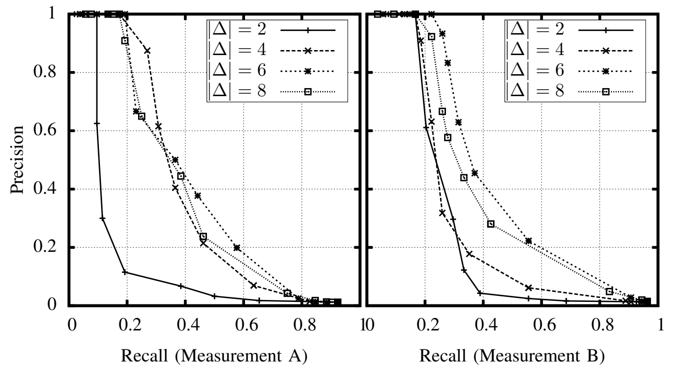

Fig. 7. Precision vs. recall of two estimations for measurements performed on Jan 26th (A) and Jan 28th (B) 2016 for varying number of observations $|\Delta|$.

In order to perform a ground truth validation, two off-site peers running *bitcoind 0.11.0* with known neighboring peers ($\sim$ 50) were used to create transactions and initially broadcast them to the network. Our monitor peer observed the propagation of transactions through the network. As the high certainty of observations with small delays has already been pointed out (cf. Fig 2), we now use an estimation that focuses on the smallest observed delay only: assuming the set of time differences for a pair of peers Δ_{v_1,v_2} contains n observations, an edge between the two peers is detected, iff the theoretical probability (i.e., the probability as derived from the model) that all n observations are larger than the smallest observation δ_{min} is higher than a certain threshold s ($P(D > \delta_{min} | C_{min} = 1)^n > s$). The threshold represents the sensitivity of the estimation and affects the false positive and negative rates.

Fig. 7 shows the resulting precision and recall depending on the chosen sensitivity and the number of observations for two different measurements. Each data point corresponds to one setting of the sensitivity threshold s. Depending on the analysis' goal, the threshold can be configured achieve either a high recall or a high precision. An increasing number of observations also increases the quality of the estimation up to about 6 estimations, where no further improvement can be seen.

With the appropriate sensitivity, the estimation can achieve a recall of 40 % while maintaining a precision of also about 40 %. This corresponds to an F_1-Score of 0.4, which is substantially lower than the theoretically achievable scores, however, much better than simply guessing the connections of a peer, which would result in an F_1-Score of 0.0125. The information gained can be useful for an attacker that wishes to eclipse certain peers. For instance, an attacker could choose a sensitivity that leads to a high recall, DDoS the detected neighbors of a specific peer (many of them can be false positives) and then repeat the timing analysis in order to identify remaining neighbors. Even if this attack does not completely eclipse a peer, it still reduces the propagation speed and the robustness of the network.

Our data shows that 44 % of peers that were online for at least one hour publish at least 5 transactions per day, making a passive attack possible. For the remaining 56 %, actively inserting transactions as sketched in Section IV-D is required. Obviously, the network topology changes over time (although churn among reachable Bitcoin network peers is quite low), which can also impede a correct analysis.

VI. Conclusion

In this paper we presented a timing analysis method for inferring the topology of flooding P2P networks. The real-world validation in the Bitcoin network showed that the proposed method can be used to infer network links at a substantial ($\sim$ 40%) recall and precision. We also showed that the commonly used *trickling* mechanism can, if inappropriately parametrized, actually reduce the resistance to traffic analysis.

The possibility for an attacker to observe the flooding process is inherent to P2P flooding networks. Therefore, countermeasures against the presented method either reduce the propagation speed (trickling) or raise the cost of establishing connections, which also affects legitimate peers negatively. Future work may include finding an optimal combination and parametrization of countermeasures depending on the network's objectives and the threat model.

Acknowledgement

This work was supported by the German Federal Ministry of Education and Research (BMBF) within the project *KASTEL_IoE* in the Competence Center for Applied Security Technology (*KASTEL*). The authors would like to thank the anonymous reviewers for their valuable comments and suggestions.

References

[1] A. Back, U. Möller, and A. Stiglic. Traffic analysis attacks and trade-offs in anonymity providing systems. In *Information Hiding*, pages 245–257. Springer, 2001.

[2] A.-L. Barabási et al. Scale-free networks: a decade and beyond. *science*, 325(5939):412, 2009.

[3] A. Beifus, D. Raumer, P. Emmerich, T. M. Runge, F. Wohlfart, B. E. Wolfinger, and G. Carle. A study of networking software induced latency. In *Networked Systems (NetSys), 2015 International Conference and Workshops on*, pages 1–8. IEEE, 2015.

[4] A. Biryukov, D. Khovratovich, and I. Pustogarov. Deanonymisation of clients in bitcoin p2p network. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014.

[5] P. Brody and V. Pureswaran. Device democracy: Saving the future of the internet of things. *IBM, September*, 2014.

[6] G. Danezis. The traffic analysis of continuous-time mixes. In *Privacy Enhancing Technologies*, pages 35–50. Springer, 2004.

[7] C. Decker and R. Wattenhofer. Information propagation in the bitcoin network. In *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*, pages 1–10. IEEE, 2013.

[8] P. Erdős and A. Rényi. On random graphs. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.

[9] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 129–144, 2015.

[10] P. Koshy, D. Koshy, and P. McDaniel. An analysis of anonymity in bitcoin using p2p network traffic. In *Financial Cryptography and Data Security*, volume 8437 of *Lecture Notes in Computer Science*, pages 469–485. Springer Berlin Heidelberg, 2014.

[11] H. V. Madhyastha, T. Isdal, M. Piatek, C. Dixon, T. Anderson, A. Krishnamurthy, and A. Venkataramani. iPlane: An Information Plane for Distributed Services. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, pages 367–380. USENIX Association, 2006.

[12] A. Miller, J. Litton, A. Pachulski, N. Gupta, D. Levin, N. Spring, and B. Bhattacharjee. Discovering bitcoin's public topology and influential nodes, 2015.

[13] S. J. Murdoch and G. Danezis. Low-cost traffic analysis of Tor. In *Security and Privacy, 2005 IEEE Symposium on*. IEEE, 2005.

[14] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. 1(2012):28, 2008.

[15] T. Neudecker, P. Andelfinger, and H. Hartenstein. A simulation model for analysis of attacks on the bitcoin peer-to-peer network. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 1327–1332, May 2015.

Appendix
Analytical Propagation Delay Model

The proposed timing analysis method infers the number of hops a message traveled through the network by comparing the observed delay to the delay that is to be expected for several path lengths. Therefore, a model for the propagation delay depending on the path length is required. Given two randomly chosen peers as the sender and the receiver and their (hop) distance in the network graph, we want to know the delay distribution between sending and receiving a message. This analytical approximative model accounts for multi-hop propagation through the network and can be used to infer how many hops a specific message took.

We will now introduce the notions used in the propagation model. The network is represented as a connected and undirected graph $G = (V, E)$, where V is the set of peers ($v \in V$) and E is the set of connections ($e \in E$) between peers. We define $x(e)$ as the random variable modeling the existence of edge e. We assume a random, *Erdos-Renyi* graph network model [8], which means each possible edge exists independently with a certain probability $P(x(e))$. We also assume that the zero-hop delay distribution, represented by the random variable D_1, between peers in the network, is known.

A path R between two peers is a sequence of edges $(e_1, ..., e_c)$ connecting the peers with a length of $|R| = l$. As the graph is assumed to be connected, at least one path between any two vertices of the graph exists. We define a class of paths $c(l)$ as the set of paths with a length of l. The probability of existence of a certain path in class $c(l)$ is given as $p_l := P(e)^l$. Each class is characterized by a different delay distribution, represented by the discrete random variable D_l. All random variables in this model are regarded as discrete random variables. The delay distribution defined by $P(D_l = t)$ states the probability of receiving a message via a single, isolated path of length l time t after sending. For now, we assume that for each class there is a global distribution independent of the sender and receiver. In this case, each delay distribution can be calculated by convolution from the zero-hop ($l = 1$) delay distribution, which is assumed to be known.

A. Approximative Propagation Delay Model

We will now formulate the analytical propagation delay model and begin by deriving some basic properties of the network that will be required later. We define Z_l as the *maximum possible number* of paths in class $c(l)$ between two peers and z_l as the actual number of paths in class $c(l)$ between two peers. Then Z_l is calculated as

$$Z_l = \binom{|V| - 2}{l - 1} \cdot (l - 1)!. \qquad (2)$$

The probability that *exactly* k paths in a given class $c(l)$ exist can be approximated using the probability of existence p_l for each path by the binomial distribution as

$$P(z_l = k) \approx \binom{Z_l}{k} \cdot p_l^k \cdot (1 - p_l)^{Z_l - k}. \tag{3}$$

Please note that the existences of two paths are not statistically independent. Hence, this formula serves only as an approximation. For $l < 3$ the solution is exact. As only short path lengths are of interest for the proposed timing analysis technique, we accept the resulting error and leave a more precise model to future work.

Given these basic graph properties, we now focus on the flooding protocol in order to develop an information propagation delay model. For now we ignore all but one class $c(l)$. We also ignore the probability of existence for the paths in this class and assume that exactly $z_l = n$ paths in class $c(l)$ between the sender and receiver exist. We define $\hat{D}_l$ as the random variable modeling the delay between broadcasting a message over these n paths and the reception of the *first* message via any path. There are n paths that can be the fastest transmitting path. The fastest path has probability $P(D_l = t)$ of resulting in time t and all other $(n-1)$ paths must be slower than t, resulting in

$$P(\hat{D}_l = t | z_l = n) = P(D_l = t) \cdot P(D_l > t)^{n-1} \cdot n. \tag{4}$$

We now take the probability of existence for paths into account and want to know the latency distribution of the fastest arriving message over any existing path from a class $c(l)$. For this, we apply the law of total probability to equation 4:

$$P(\hat{D}_l = t) = \sum_{k=1}^{Z_l} (P(z_l = k) \cdot P(\hat{D}_l = t | z_l = k))$$

The distribution defined by $P(\hat{D}_l = t)$ equals the delay distribution of the fastest message that took a path of length l through the network. However, we are interested in the delay distribution of the fastest message over any path of any length. Therefore, we will no longer ignore all classes but one and acknowledge the fact that messages might be transmitted faster on paths with different lengths. When choosing one class $c(l')$, the probability that none of the other classes results in a delay smaller than t is given by the product of probabilities of all other classes resulting in a delay larger than t, resulting in

$$P(\hat{D}_{l \neq l'} > t) = \prod_{k \neq l'} P(\hat{D}_k > t).$$

We now define D as the random variable modeling the overall delay of the fastest message transmitted over any of the possibly existing paths. The probability of a transmission resulting in a delay of t is calculated as the probability that the fastest transmission over a path of one length results in latency t and no other transmission over any path of another class results in a smaller latency:

$$P(D = t) = \sum_{k=1} (P(\hat{D}_k = t) \cdot P(\hat{D}_{l \neq k} > t)). \tag{5}$$

With $P(D = t)$, we now have an analytical propagation delay model. However, for use in the timing analysis, we need to calculate the propagation delay distribution depending on the shortest path length C_{min}. The minimum path length C_{min} between two peers can be formally specified as $C_{min} = l \Leftrightarrow (z_l > 0 \wedge \forall d < l : z_d = 0)$. For each pair of peers in a connected network, exactly one value for C_{min} exists. Using this definition, we will now adapt Equation 5 to calculate the probability of observing a time difference δ, depending on the minimum hop-count l between the two senders. From the definition of $C_{min} = l$, it follows that no paths shorter than l exist. Hence the probability of existence for such paths has to be set to 0 ($p_i = 0, i < l$). Equation 5 uses Equation 3 to calculate the probability that exactly k paths exist in a given class $c(l)$. Adapting Equation 3 to assume that at least one path exists leads to

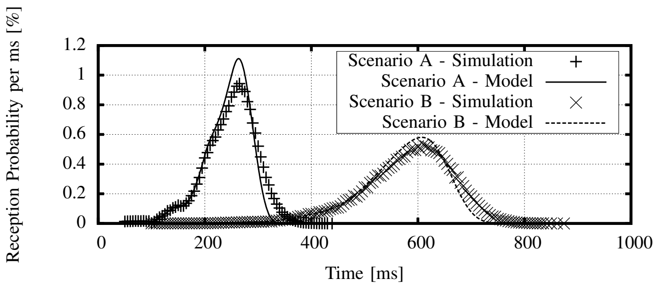

Fig. 8. Propagation Delay: Comparison of Model to Simulation

$$P(z_l = k | z_l > 0) = \frac{P(z_l = k \cap z_l > 0)}{P(z_l > 0)}. \tag{6}$$

Obviously, the numerator equals $P(z_l = k)$ for $k > 0$ and the denominator is calculated as $1 - (1 - p_l)^{Z_l}$.

We also need the *a priori* probability of C_{min} as a starting point of the estimation. Assuming independence, the probability of a shortest path length of l is calculated as the joint probability that at least one such path exists and that no shorter path exists, resulting in

$$P(C_{min} = l) = (1 - (1 - p_l)^{Z_l}) \cdot \prod_{i<l} (1 - p_i)^{Z_i}. \tag{7}$$

As discussed before, the formula is only exact for shortest path lengths smaller than 3. These two adaptions enable us to calculate $P(D = t | C_{min} = l)$.

B. Validation and Limitations

In order to validate the model, we compare the propagation delay as calculated from the model to the propagation delay measured in simulations. We consider two exemplary scenarios: *Scenario A* comprises a network with 1,000 peers, an average of 8 connections per peer and a uniformly distributed latency (D_1) between 50 and 100 ms whereas *Scenario B* consists of 6,000 peers, each with 16 connections on average and a latency distribution (D_1) between 100 and 300 ms. The delay distribution for multi-hop transmission ($D_2, D_3, ...$) was calculated by convolution from D_1 in both scenarios. Figure 8 shows the resulting delays for these two scenarios.

It can be seen that during the beginning of the propagation our model perfectly matches the simulation outcome, whereas a deviation becomes visible afterward. This deviation is caused by an abstraction made in the model: we treat two paths between sender and receiver as independent although the paths may share common hops (i.e., they may not be vertex-independent). Obviously, this situation can only occur for paths with a length of three or more, as one hop must be the joining hop and one hop must differ in order to result in different paths. We use the model to distinguish between a shortest path length of 1 and any longer path length. Therefore, a precise modeling of the propagation's beginning is required, whereas the further propagation is not crucial.

Another, more practical, limitation of the presented model can be imposed by the possible extremely large number of possible paths Z_c, which can reach values of more than 10^{100} pushing the following calculation of $P(z_c = k)$ against common numerical limits. However, the use of arithmetic libraries as well as limiting the path length bypasses these issues. Lastly, the assumption of an ER-graph does not hold in most real-world networks. We argue, however, that for some network models an adaption is possible analytically, whereas for others the use of simulation and fitting techniques allows the development of practically usable models, which is done in Section V. The model derived here can be especially useful for theoretically assessing the timing analysis presented.