Exploiting Transaction Accumulation and Double Spends for Topology Inference in Bitcoin

Matthias Grundmann, Till Neudecker, and Hannes Hartenstein

Institute of Telematics, Karlsruhe Institute of Technology, Karlsruhe, Germany
matthias.grundmann@kit.edu, till.neudecker@kit.edu
hannes.hartenstein@kit.edu

Abstract. Bitcoin relies on a peer-to-peer network for communication between participants. Knowledge of the network topology is of scientific interest but can also facilitate attacks on the users' anonymity and the system's availability. We present two approaches for inferring the network topology and evaluate them in simulations and in real-world experiments in the Bitcoin testnet. The first approach exploits the accumulation of multiple transactions before their announcement to other peers. Despite the general feasibility of the approach, simulation and experimental results indicate a low inference quality. The second approach exploits the fact that double spending transactions are dropped by clients. Experimental results show that inferring the neighbors of a specific peer is possible with a precision of 71 % and a recall of 87 % at low cost.

1 Introduction

Bitcoin [9] is a digital currency system that stores transactions in a blockchain. Participants are connected via a peer-to-peer (P2P) network in order to exchange transactions and blocks. The topology of the P2P network is an important aspect in ensuring anonymity of users and robustness against denial of service attacks [5], double spending attacks [6], and attacks on mining [3, 10]. For instance, knowledge of the network topology can enable network based attacks on anonymity [4, 1, 7].

In this work we present and analyze two approaches that aim at inferring the topology of the publicly reachable Bitcoin network. Peers that are not reachable (e.g., peers that do not accept incoming connections) as well as *private* networks such as FIBRE[1] or mining pool networks are not covered by our work. Neither of the presented approaches rely on the existence of side channels (e.g., peer discovery), because they exploit properties of the implementation of the flooding protocol used for transaction propagation.

2 Related Work

Topology inference in Bitcoin has been the subject of several previous works. Peer discovery in Bitcoin allows clients to query their neighbors for IP addresses

[1] http://bitcoinfibre.org/

of other peers in order to establish connections to them. The queried neighbor then sends a list with IP addresses along with a `lastseen` timestamp. Until March 2015 the timestamp was not randomized sufficiently and allowed Miller et al. [8] to exploit this mechanism and infer the network topology. Peer discovery can also be exploited for topology inference by sending marker IP addresses to remote peers [1].

Neudecker et al. [11] performed a timing analysis of the propagation of transactions in order to infer the network topology. By connecting to all reachable peers of the network and observing the timestamps of receptions of certain transactions, the path of the transaction and thereby the connections between peers can be inferred. This approach requires connections to all reachable peers and requires the adversary to actively create transactions if he is unable to determine the creator of a transaction. Furthermore, changes made to the propagation mechanism of the reference client Bitcoin Core[2] in 2015 render this method much more difficult nowadays.

3 Fundamentals

For this work the networking code and especially the forwarding of transactions is relevant. After a peer creates or receives a transaction, it sends an `INV` message containing the hash of the transaction to each of its neighbors. A peer receiving an `INV` message checks whether it has already received the transaction, and, if the transaction is new, sends a `GETDATA` message to the peer it received the `INV` message from. This peer then replies with a `TX` message containing the actual transaction.

When a peer receives a transaction, it validates the correctness of the transaction. This includes checking the correct format, checking whether the sum of input values is at least as large as the sum of output values, and checking whether the inputs of the transaction are actually spendable. Because every transaction output can only be spent once, a transaction with an input that was already spent by a transaction received earlier is regarded invalid and dropped silently. We will demonstrate how to exploit this behavior regarding *double spends* for topology inference in Section 6.

In order to enhance privacy by impeding timing analysis, `INV` messages are not sent out immediately after receiving and validating a transaction, but are delayed according to a non-deterministic function. Bitcoin Core maintains one outgoing queue for each connected peer. When a new transaction is received or created, this transaction is added to all queues. Therefore, each queue contains all transactions that are to be announced to that peer. At certain times all messages in a queue are announced to the neighbor via a single `INV` message.[3] These times are chosen according to an exponential distribution[4] to model a

[2] `https://github.com/bitcoin/bitcoin`

[3] If there are more than 35 transactions in the queue (which occurs only infrequently), only 35 transactions are announced at once.

[4] The next time is calculated as: $\mathit{current\_time} + \ln\left(1 + X \cdot \frac{-1}{2^{48}}\right) \cdot \mathit{average\_interval\_seconds} \cdot -1000000 + 0.5$ (all timestamps are in microsec-

Poisson process. Every time the elements of the queue are sent to the neighbor, a new sending time is determined. This mechanism has the property that all transactions received between two sending timestamps are sent in one single `INV` message. We will demonstrate how to exploit this *transaction accumulation* for topology inference in Section 5.

4 Problem Statement & Assumptions

Let $G = (V, E)$ be the undirected graph modeling the peers (V) and connections (E) of the Bitcoin network. Given a subset $R \subseteq V$ of the reachable peers of the network, the adversary tries to infer all connections between all peers in R. The inference can lead to false positives (i.e., inferring a connection although no connection exists) and false negatives (i.e., not inferring a connection although a connection exists). We will use precision (i.e., the share of inferred connections that are true positives) and recall (the share of existing connections that were inferred) as metrics to describe the success of the inference.

We assume that the adversary can run a small number of peers, which can connect to as many other peers as possible. This number is limited by the number of reachable peers and the network capabilities of the adversary.[5] We also assume that the adversary is able to precisely estimate the latency between its own peers and remote peers, e.g., based on the observation of Bitcoin `ping` messages or ICMP ping messages. Furthermore, we assume that the adversary is able to create a large number of transactions. These transactions can transfer funds between addresses controlled by the adversary, however, transaction fees still have to be paid. The adversary is not assumed to have information that an ISP or state actor organization might have about connections and traffic of other peers. We do not consider stronger adversary models (e.g., ISPs), as these adversaries could simply monitor the network traffic in order to infer the network topology.

5 Exploiting Transaction Accumulation

We will now describe how the transaction accumulation mechanism implemented in Bitcoin Core (cf. Section 3) can be used for topology inference.

5.1 Description

Assume for now that the adversarial monitor peer v_M is connected to all peers $v_i \in V$ of the network. The adversary creates one transaction $t_i \in \tau$ for each connected peer v_i. All transactions are independent and not conflicting in any way (i.e., they are spending different outputs). All transactions are sent to the peer they were created for (i.e., t_i to v_i) so that they arrive at all peers at the

onds, $X \in \mathcal{U}[0 : 2^{48} - 1]$, *average_interval_seconds* is 5 s for incoming connections and 2 s for outgoing connections).

[5] Our measurements show that maintaining connections to ≈10,000 peers consumes about 20 Mbit/s.

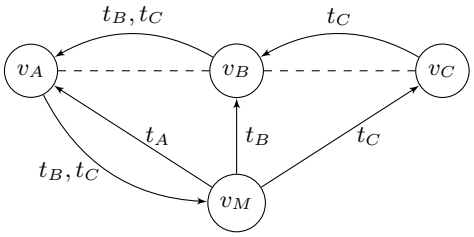

Fig. 1. Exploiting transaction accumulation for topology inference. Dashed lines indicate existing connections. Solid lines indicate the transmission of transactions.

same time. Afterwards the adversary monitors the first `INV` messages that will be received by v_M from all connected peers, and infers information about the topology by using the following inference rules:

1. If the first `INV` message that peer v_A sends to v_M contains only t_B (i.e., the transaction sent to v_B) and no other transaction from the set of created transactions τ, then v_A and v_B are connected.
2. If the first `INV` message that peer v_A sends to v_M contains more than one transaction from the set of created transactions τ, at least one of the peers associated with the announced transactions is connected to v_A.

Let us consider the scenario depicted in Fig. 1 to demonstrate the correctness of the rules. v_M is connected to v_A, v_B, and v_C. v_A is connected to v_B, v_B is connected to v_C. After the transactions were sent by the adversary, each peer has only the transaction designated for itself (and transactions created by other participants, which can be ignored). Statement 1 is equal to *If v_A and v_C are **not** connected, then v_A will **not** send an `INV` message that contains only t_C and no other transaction from the set of created transactions τ.*

Because v_A and v_C are not connected, t_C has to be relayed by another peer (v_B) to v_A. As we assumed that the adversary is connected to all peers, the adversary is also connected to v_B and has sent a transaction t_B to v_B. Because of the queuing mechanism described in Section 3, v_B's queue for v_A already contains t_B. Therefore, t_B and t_C will be announced together to v_A, which announces them together to v_M. It is also possible that t_B will be sent earlier than t_C, because the queue at v_B is sent between the reception of t_B and t_C by v_B. However, it is not possible that t_C arrives earlier than t_B at v_A.

This scenario also explains the second statement: If v_A sends an `INV` message that contains t_B and t_C, the adversary does not know whether v_B, v_C, or both are directly connected to v_A. The transactions initially sent to all peers serve as identifiable flags that the remote peers attach to the first group of transactions they forward after receiving their transaction. This allows the adversary to reconstruct the path of transactions and thereby infer connections between peers.

5.2 Discussion & Variants

While this topology inference approach is possible under perfect conditions, there are several issues that can arise when not all assumptions are met.

If there are peers on the network that the adversary is not connected to, false positives can occur. Consider again the scenario depicted in Fig. 1, but let us assume that v_M is not connected to v_B. Then, v_B would not have received a transaction t_B, and the `INV` message sent by v_A would only include t_C, which would lead to the wrong conclusion that v_A and v_C are directly connected.

False positives can also occur when the adversary cannot guarantee that all transactions arrive at all peers at the same time. While the latency measurement might be precise in general, temporal changes, e.g., due to bandwidth peaks, are possible and hard to foresee by the adversary. Furthermore, sending several thousand transactions within a few hundred milliseconds in a coordinated way can require much bandwidth and computational effort.

Even when all assumptions are met, the success of the approach depends on the order in which the remote peers forward transactions to their neighbors. This order is determined by the sending times of the respective queues and is unknown to the adversary. Therefore, repeating the approach many times is required in order to infer a large number of connections.

Another issue with the approach is that it is not possible to explicitly target a specific remote peer in order to infer the connections of that peer only. Instead, the inferred connections are a mostly random subset of all existing connections, which cannot be influenced by the adversary.

Variant *DS*: We will now present a variant of the discussed approach that reduces the cost by reducing the incurring transaction fees. Assuming 10,000 peers on the network and transaction fees of \$1 per transaction, the cost for one run of the approach is \$10,000.[6] A possibility to reduce this cost is to still create one transaction per peer, but to create these transactions so that they are all double spends of only a few different outputs. The number of different inputs among all transactions is a parameter freely chosen by the adversary (e.g., DS_3 denotes variant *DS* with three different inputs). Each transaction is still unique (e.g., by having different outputs), which enables the mapping of one transaction to one remote peer. That way, the adversary has to pay only for those transactions that get included in the blockchain. However, this approach can cause transactions to be dropped, which can cause false positives. We will evaluate the effect of double spendings on this approach in the next subsection.

5.3 Simulation Results

We will now briefly describe the used simulation setup before the results of the simulation are presented and discussed. Simulations were performed using a discrete event simulation. The network topology was generated by creating eight outbound connections to uniformly chosen peers for each simulated peer.

[6] Due to extreme fluctuations in transaction fees and exchange rates, this calculation is just an example.

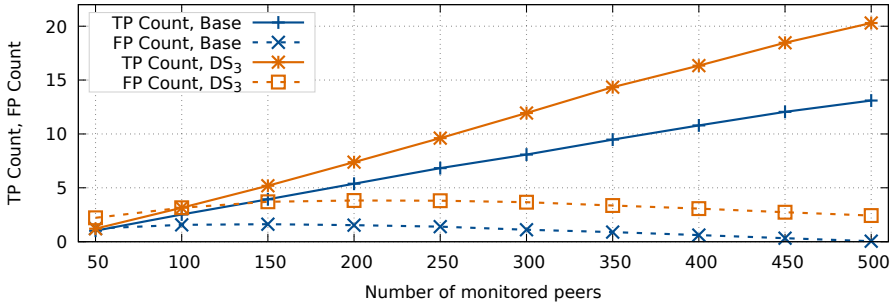

Fig. 2. Number of true positives and false positives per run for the base approach and the variant *DS* with three different inputs.

This results on average in eight incoming and 16 total connections per peer. The adversary was modeled as a specific peer that establishes a large number of connections (depending on scenario) and sends and receives the transactions according to the presented inference strategy.

While the simulation matches the general behavior of the Bitcoin client, several simplifications were made. First, we model the three-step transaction propagation process (`INV - GETDATA - TX`) as one single event. Secondly, the latencies between peers are chosen according to a normal distribution ($\mu = 100\,\text{ms}$, $\sigma = 50\,\text{ms}$, truncated to $[1\,\text{ms}, 6000\,\text{ms}]$). Thirdly, when peers forward transactions and have more than 35 transactions in their queue, they choose the transactions to forward uniformly at random, but prefer transactions created by the adversary[7]. Therefore, our simulation is not a precise model of the Bitcoin network or testnet and the results should be seen as a proof of concept.

Fig. 2 shows the true positive (TP) and false positive (FP) count depending on the number of connected peers for the base variant and variant DS with three different inputs for one run of the approach. The total number of peers on the network is 500. If the adversary is connected to all remote peers, one run of the approach results in about 8 correctly detected connections for variant DS, and in about 5 correctly detected connections for the base variant. Reduction of the share of connected peers leads to a decline in the true positive count. While we expected the false positive count of variant DS to be higher than that of the base variant, variant DS also results in a higher true positive count compared to the base variant. Double spends limit the propagation of individual transactions, because they are dropped at all peers that already received another transaction with the same input. This limitation of propagation is actually beneficial for the approach, because only single-hop propagation of each transaction (i.e., from one remote peer to another and back to v_M) is required and leads to the correct detection of a connection.

[7] This models the scenario that the adversary pays higher transaction fees than the fees for the other transactions.

Simulations of larger network sizes showed a linear relationship between the number of peers and the TP and FP counts, i.e., a network with twice the number of peers results in about twice the number of true positives and false positives (cf. Appendix Fig. 7).

5.4 Experimental Results

In order to perform a ground truth validation of our simulation results, we set up several peers on the Bitcoin testnet: Two peers perform the role of the adversary peers and connect to all reachable public peers (around 520 connections during the experiments in November 2017[8]). Another five peers running Bitcoin Core (0.15.0.1) serve as validation targets. These peers establish eight outgoing connections and are reachable to the adversary peers via IPv4 and IPv6. In this setup the adversary peers are connected to all neighbors of the validation targets, which is a best-case scenario for inference.

During the experiments one of the adversarial peers sends transactions to other peers so that they arrive at the same time. The latency to remote peers was measured using ICMP ping, TCP SYN packets, and Bitcoin ping messages.

We performed 50 runs of variant *DS* of the described inference approach using transactions with three different inputs. A total of 632 unique connections were detected, which roughly conforms to our simulation results. Out of these 632 connections, only 9 connections were connections from or to one of our validation peers. From these 9 detected connections, only 6 actually existed, which corresponds to an observed recall of 67 %.[9] Roughly estimating the total number of connections on the testnet to be 4,160[10], and assuming a precision of 67 % results in a recall (with respect to all connections of the network) of about 10 % after 50 runs for a total cost of $50 * 3 = 150$ transaction fees.

Although the small sample size only allows very rough estimates of the real quality to be expected, and a more extensive ground truth validation could result in more precise estimations of the expected precision and recall, the results still help in assessing the topology inference approach. While the discussed approach is in fact possible to perform, we believe its execution to be hardly practical. For scientific purposes the approach is too invasive and lacks validation possibilities. For adversarial purposes the lack of influence on which connections are inferred prevents targeted attacks, especially taking into account that topology inference is only an intermediate goal for further attacks.

6 Exploiting Double Spends

One major drawback of the approach presented in Section 5 is that it is not possible to infer the connections of a specific peer only, rather than inferring

[8] Peers were found using `https://github.com/ayeowch/bitnodes/`

[9] Because of the small sample size, the real recall can strongly deviate from the observed recall.

[10] 520 peers with 8 connections each.

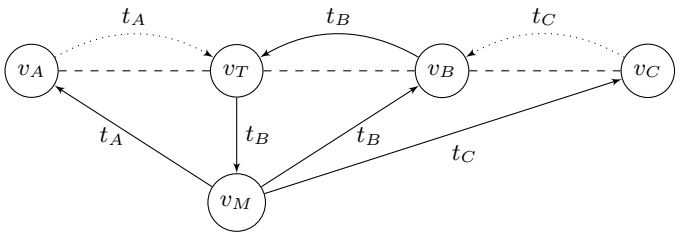

Fig. 3. Exploiting double spends for topology inference. Dashed lines indicate existing connections. Solid lines indicate the transmission of transactions. Dotted lines indicate dropping of transactions by the receiver because of an earlier reception of a conflicting transaction.

random connections of the network. This is not only problematic for adversaries, but also makes validation a challenge. We will now describe and analyze an approach that allows inferring the connections of a specific peer v_T by exploiting the fact that clients drop transactions that double spend bitcoins.

6.1 Description

Again, assume for now that the adversarial monitor peer v_M is connected to all peers $v_i \in V$ of the network. One of the connected peers is the target peer v_T, the connections of which the adversary wants to infer. The adversary creates one transaction $t_i \in \tau$ for each connected peer v_i, except for the target peer v_T. All transactions have the same input, i.e., they are double spends, but all transactions are unique, e.g., by specifying different output addresses. Again, all transactions are sent to the peer they were created for (i.e., t_i to v_i) so that they arrive at all peers at the same time. Then the adversary monitors which transaction the target peer v_T forwards to the monitor peer v_M and can conclude that the peer associated with the forwarded transaction is directly connected to the target peer v_T.

Let us consider the scenario depicted in Fig. 3 to demonstrate the correctness of the strategy. The monitor peer v_M is connected to v_A, v_T, v_B, and v_C. The target peer v_T is connected to v_A and v_B, while v_B is also connected to v_C. After the transactions were sent by the adversary, every peer only has the transaction designated for itself, and v_T has no transaction received yet. Every peer will only accept and forward *exactly one* of the created transactions, because they are all double spends of the same output. Therefore, if v_C forwards t_C to v_B (dotted line), v_B will drop t_C because of the earlier reception of the conflicting transaction t_B. Because the target peer v_T has not yet received any of the conflicting transactions, it will accept exactly one transaction forwarded by one of its neighbors (transaction t_B in Fig. 3). This transaction gets forwarded to the monitor peer v_M and indicates a neighbor of the target peer v_T.

6.2 Discussion & Variants

If the adversary is not connected to all peers of the network, or if the transactions are not received by all peers at the same time, false positives can occur. The reason is basically the same as for the approach exploiting transaction accumulation discussed in Section 5.2: A neighbor of v_T that did not receive its double spending transaction from v_M will accept another double spending transaction t_i from another neighbor v_i and forward that transaction to v_T, which may forward t_i to the adversary causing the false inference of a connection between v_T and v_i. Obviously, if the adversary cannot establish a connection to v_T, the connections of v_T cannot be inferred using the discussed approach. We will now discuss three variants of the presented approach that aim at optimizing the inference even when not all assumptions are met.

Variant *Count*: When repeating the approach several times, one would expect the transactions associated with real neighbors (true positives) to be sent to the adversary by v_T more often than those of peers that are not connected to v_T (false positives), because those transactions have to be relayed by another peer and should be slower. In order to reduce false positives, the approach can be repeated and connections are only identified, if the number of transactions indicating a specific peer as a neighbor of v_T is larger than a certain threshold.

Variant *Ignore*: Assume that t_A is forwarded by v_T to v_M. If the adversary was unable to synchronize the reception of all transactions at all remote peers (e.g., due to bad latency estimation or bandwidth limitation), it is possible that t_A is *also* forwarded to v_M by another peer, say, v_B. As such a reception indicates the violation of a key assumption and v_T might have received t_A from v_B rather than directly from v_A, the adversary can opt to ignore the result without concluding a connection between v_T and v_A.

Variant *Suppress*: The cost for a single run of the approach is one transaction fee. However, one single run reveals at most one connection of the target peer. In order to infer more connections, additional runs are necessary, which each come at the cost of one transaction fee. Which connection can be inferred depends on which transaction arrives first at v_T, which is determined by the sending times of the remote peers and the latencies between peers. With bad luck (or single clients being very fast), multiple runs of the approach can all result in inference of the same, already known, connection. Variant *Suppress* slightly modifies the approach to eliminate the repeated inference of the same connection. Consider again the example depicted in Fig. 3 and assume that the adversary inferred the connection between v_T and v_B in the first run of the approach. For the next run, we (1) want the transaction t_B to be dropped at v_T and (2) we do not want v_B to forward any other transaction t_i. While simply not sending any transaction to v_B would satisfy the first requirement, it would make v_B a hidden node and violate the second requirement. Therefore, we modify the way the double spending transactions are created. Assume there are two unspent outputs i_1 and i_2 that will be used as inputs to the transactions in the following way:

- All peers v_i, except for v_T and v_B, receive transactions t_i spending i_1 only.

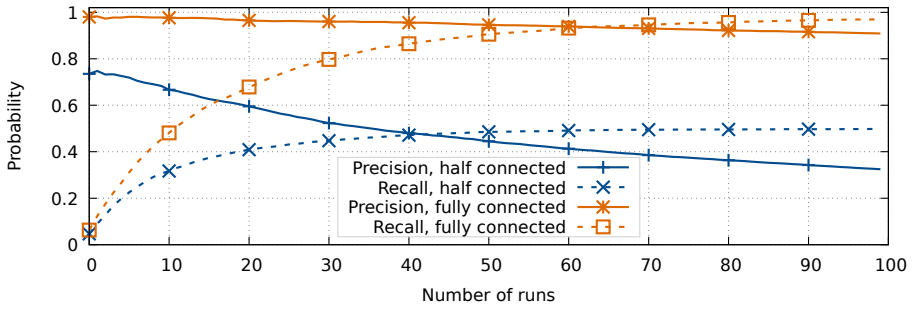

Fig. 4. Precision and Recall depending on the number of runs with v_M being connected to 250 (half connected) and 500 (fully connnected) of 500 peers.

- v_T receives a transaction t_T spending i_2 only.
- v_B receives a transaction t_B spending i_1 and i_2.

This approach satisfies both requirements: v_T will drop t_B because it is a double spend of i_2. Any transaction t_i will be dropped by v_B because they are double spending i_1. Yet, any transaction t_i will be accepted by v_T because they are spending different outputs (i_1 and i_2).

6.3 Simulation Results

We simulated the approach exploiting double spends with the same simulation setup as described in Section 5.3. Fig. 4 shows how recall and precision develop depending on the number of runs for the base version of the approach. If the monitor peer v_M is connected to all peers of the network, the recall reaches 95% after 100 runs while the precision decreases slowly. The precision decreases because once all neighbors of the target are detected, the precision can only fall by detecting more false positives. Detecting the same neighbor multiple times is used in the variant *Count*. While variant *Count* can maintain a high precision, the recall is worse than for other variants (cf. Appendix Fig. 8).

If the adversary is connected to only half of the peers of the network, the maximum possible recall is 50 %, because the adversary is on average only connected to half of the neighbors of v_T. As described above, the target's neighbors being not connected to the adversary cause false positives and thus the precision is lower than for the fully connected scenario.

Fig. 5 shows precision and recall of the approach when using the variant *Suppress* with v_M being connected to all peers. Using only this variant results in the recall growing faster, because this variant prevents neighbors from being detected multiple times. However, not only true neighbors are detected faster, but also false positives, which results in a faster declining precision. If v_M is not connected to all peers, the precision falls even faster, because the likeliness that a detection is a false positive is higher.

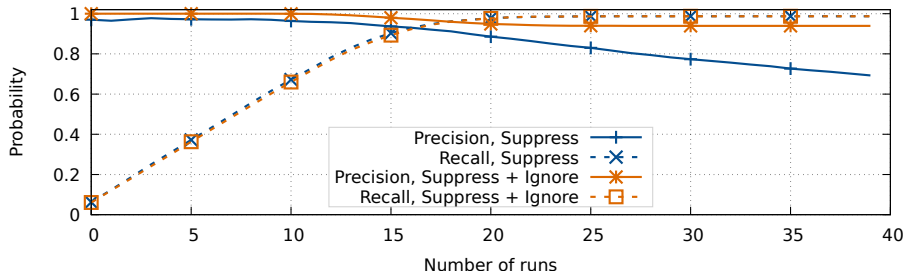

Fig. 5. Precision and recall depending on the number of runs for variants *Suppress* and *Suppress + Ignore* with v_M being connected to 500 of 500 peers (fully connected).

The precision can be improved by combining the variants *Suppress* and *Ignore*, for which precision and recall are also shown in Fig. 5. Combining both variants results in a recall of 96 % after 25 runs with a precision of about 94 % if the monitor is connected to all peers.

6.4 Experimental Results

We validated the approach in the testnet with the setup described in Section 5.4 with the exception that the adversarial peers did not send any transactions to the IPv6 addresses of the validation targets. The reason for this exception is that otherwise the presented approach infers connections between the IPv4 and IPv6 addresses of the same peer. While this might also be an interesting application for the approach, it would impair our validation.

We ran the approach six times against each of the five validation targets with 50 runs each. Analyzing the data generated during the experiments in different ways results in various combinations of precision and recall. Two of them using *Suppress* and *Ignore* are shown in Fig. 6. The combination of the variants *Suppress* and *Ignore* results in a recall of 60 % and a precision of 97 %. The recall can be improved though by relaxing the restrictions imposed by *Ignore* by using only the variant *Suppress*. This combination results in a recall of 87 % and a precision of 71 % (also shown in Fig. 6) for a total cost of 99 transaction fees.

While these results indicate a high inference quality, we emphasize again that the adversarial peer was connected to all neighbors of the validation peers in our experiments and hidden neighbors could impair the inference quality.

Because not only our validation peers were dual-stacked (i.e., connected via IPv4 and IPv6 to the network) but also other peers on the network are, it is possible that a peer v_N is connected via IPv4 to one of our validation targets and via IPv6 to the adversary peer. In this situation the approach might infer a connection between v_N's IPv6 address and the validation target's IPv4 address. While this is technically a false positive, it is still correct that both peers are connected. As this situation might have occurred several times, some connections that were categorized as false positives might actually be correctly inferred.

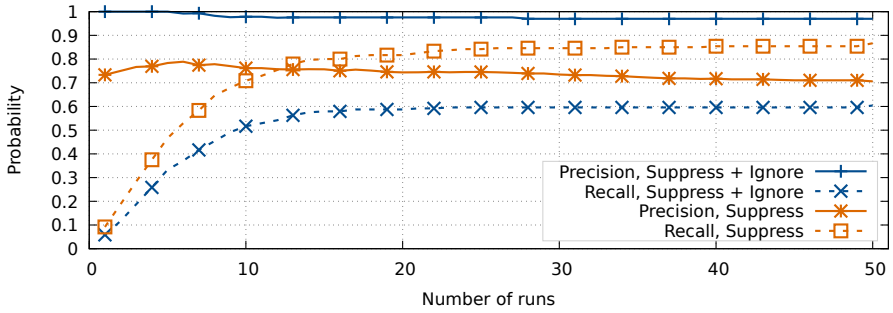

Fig. 6. Experimental Result: Precision and recall depending on the number of runs using variant *Suppress* and *Suppress+Ignore*.

7 Conclusion

While the presented approach exploiting transaction accumulation is likely not suitable for topology inference, the approach exploiting double spends showed sufficient detection quality at reasonable cost. We emphasize that transaction accumulation still leaks information that might be exploited in more advanced approaches. A simple countermeasure that prevents these kinds of attacks would be to mix the order of transactions, e.g., by regularly sending only a subset of the transactions in the outgoing queues.

An obvious countermeasure against the approach exploiting double spends would be to forward double spends, which, however, would create the potential for DoS attacks. Another countermeasure could be to not always forward the transaction that was received first, but randomly deciding which double spending transaction will be forwarded, which could affect security against double spending attacks in fast payments [6, 2]. Furthermore, individual peer operators may choose to deny incoming connections, which prevents the discussed approaches from working, but is not desirable from an overall network's perspective. On the other hand, operating a reachable peer with a large number of incoming connections from unreachable peers also impedes the presented inference approaches. Finally, because of the large number of transactions created, such attacks can be observed by monitoring large parts of the network.

While we presented some optimizations and variants of the approaches, many more variants and combinations (e.g., including timing information, making use of more than one adversarial peer, continuous sending of transactions, further combination of double spending inputs) are possible and might result in better inference quality. Although the validation in the Bitcoin testnet gives an idea of the general feasibility, a validation in the real Bitcoin network promises more insights, but is currently not feasible for the presented approaches due to high transaction fees. Finally, while our approaches aimed at topology inference, similar approaches exploiting the same mechanisms might be used against the anonymity of users.

Acknowledgements

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. Biryukov, A., Khovratovich, D., Pustogarov, I.: Deanonymisation of clients in Bitcoin P2P network. In: Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. ACM (2014)
2. Decker, C., Wattenhofer, R.: Information propagation in the Bitcoin network. In: Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on. pp. 1–10. IEEE (2013)
3. Eyal, I., Sirer, E.G.: Majority is not enough: Bitcoin mining is vulnerable. In: International Conference on Financial Cryptography and Data Security. pp. 436–454. Springer (2014)
4. Fanti, G., Viswanath, P.: Anonymity properties of the Bitcoin P2P network. arXiv preprint arXiv:1703.08761 (2017)
5. Heilman, E., Kendler, A., Zohar, A., Goldberg, S.: Eclipse attacks on Bitcoins peer-to-peer network. In: 24th USENIX Security Symposium (USENIX Security 15). pp. 129–144 (2015)
6. Karame, G.O., Androulaki, E., Capkun, S.: Double-spending fast payments in Bitcoin. In: Proceedings of the 2012 ACM conference on Computer and communications security. pp. 906–917. ACM (2012)
7. Koshy, P., Koshy, D., McDaniel, P.: An analysis of anonymity in Bitcoin using P2P network traffic. In: Financial Cryptography and Data Security. Lecture Notes in Computer Science, vol. 8437, pp. 469–485. Springer Berlin Heidelberg (2014), `http://dx.doi.org/10.1007/978-3-662-45472-5_30`
8. Miller, A., Litton, J., Pachulski, A., Gupta, N., Levin, D., Spring, N., Bhattacharjee, B.: Discovering Bitcoin's public topology and influential nodes (2015)
9. Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System (2008)
10. Nayak, K., Kumar, S., Miller, A., Shi, E.: Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In: Security and Privacy (EuroS&P), 2016 IEEE European Symposium on. pp. 305–320. IEEE (2016)
11. Neudecker, T., Andelfinger, P., Hartenstein, H.: Timing analysis for inferring the topology of the Bitcoin peer-to-peer network. In: 2016 Intl IEEE Conference on Advanced and Trusted Computing (ATC). pp. 358–367 (July 2016)

Appendix

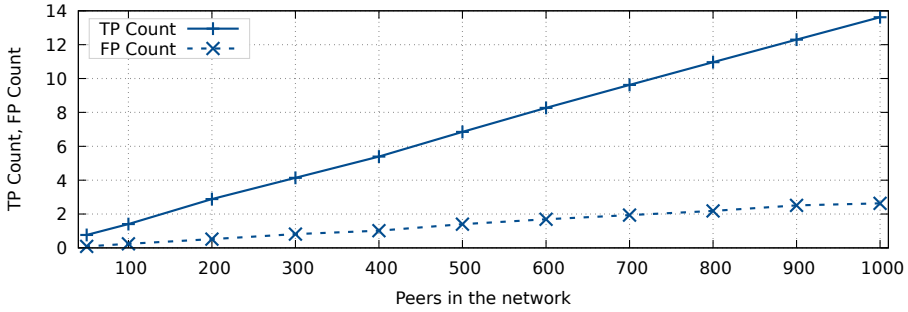

Fig. 7. Exploiting transaction accumulation: Precision and recall depending on the network size for v_M being connected to half of the peers.

Fig. 7 shows that the approach exploiting the accumulation of transactions scales linearly with the network size.

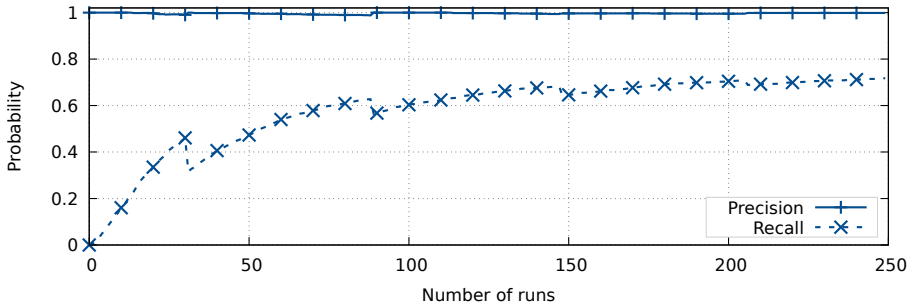

Fig. 8. Precision and recall depending on the number of runs for variant *Count* and v_M being connected to 375 of 500 peers.

Fig. 8 shows precision and recall for the variant *Count* of the approach exploiting double spends. As can be seen, the recall increases in steps. These steps are caused by adjusting the threshold for the required number of receptions. While this variant can be used to reach high precision, the recall is limited even after more than 200 runs.