A Simulation Model for Analysis of Attacks on 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*—We present a simulation model of the Bitcoin peer-to-peer network, a widely deployed distributed electronic currency system. The model enables evaluations of the feasibility and cost of attacks on the Bitcoin network at full scale of 6,000 nodes. The simulation model is based on unmodified code from core segments of the Bitcoin reference implementation used by 99% of nodes. Parametrization of the model is performed based on large-scale measurements of the real-world network. We present preliminary validation results showing a reasonable correspondence of the propagation of messages in the Bitcoin network compared with simulation results. We apply the model to study the feasibility of a partitioning attack on the network and show that the attack is sensitive to the churn of the attacking nodes.**

I. Introduction

Bitcoin [10] is a distributed electronic currency system that has found a broad user base and is gaining acceptance by established organizations. Financial fraud is prevented using a proof-of-work scheme and cryptographic mechanisms that guarantee consistency of the financial transactions given certain properties of the Bitcoin peer-to-peer network formed by the system's users. For instance, it is assumed that no single entity controls more than 50% of the computational power in the network. To increase the users' trust in the system, it is necessary to study Bitcoin's resistance not only against attacks on cryptographic mechanisms but also on the underlying network itself.

In the literature, a number of attacks as well as appropriate counter-measures have been evaluated analytically using models generated from real-world measurements. However, existing works abstract both from details of the Bitcoin client behavior and from network conditions such as topology, link latencies, and churn. Hence, the real-world feasibility of the studied attacks remains difficult to assess.

In this paper, we present a simulation model for evaluation of attacks on the Bitcoin network. Each node's behavior is modeled using unmodified code from core segments of *bitcoind*, the Bitcoin reference implementation used by 99% of nodes in the real-world network [14]. The model parameters defining the network conditions (e.g., latency) are gathered by measurements covering a broad sample of nodes in the Bitcoin network. We present preliminary validation results that display a reasonable correspondence when comparing the observed information propagation in the real-world network with simulation results. Based on the simulation model, we evaluate the feasibility of a partitioning attack on the Bitcoin network and demonstrate the need to model both the client behavior and the network conditions to obtain meaningful results. An attacker entering the network with nodes of a botnet can successfully disrupt operation of the system by reducing the connectivity of honest peers in the network. However, the attacker requires a large number of stable (i.e., continuously connected) attacking nodes. The main contributions of the paper are threefold:

Bitcoin Simulation Model: To the best of our knowledge, we are the first to demonstrate full-scale simulation-based evaluations of attacks on the Bitcoin network: the simulation proceeds at a factor of 0.34 of wall-clock time (i.e., 10 hours of simulation time require 29 hours wall-clock time) while still retaining high accuracy through the use of segments of unmodified application code. Validation results show that the model approximates the information propagation in the real-world network with reasonable accuracy.
Code Transformation: We describe the main challenges in transforming the Bitcoin application code to a simulation model and propose mechanisms to address the conversion of complex control flows and long-running function calls.
Evaluation of Partitioning Attack: Applying the simulation model, we study the feasibility of an attack aiming to partition the network. We demonstrate the tradeoff in the costs of the attack with respect to the amount of network resources that must be available to the attacker and the time required to perform the attack.

The remainder of the paper is structured as follows: in Section II, we outline related work in evaluation of Bitcoin regarding network and security properties and in modeling approaches for similar distributed systems. Section III describes our simulation model of the Bitcoin network conditions and client behavior. Section IV details the transformation of the client source code to the simulation model. In Section V, we present preliminary validation results with reference to real-world measurements. In Section VI, the simulation model is applied to study the feasibility of a partitioning attack on the Bitcoin network. In Section VII, we summarize our results and discuss future work.

II. Related Work

Most previous studies of the Bitcoin network are based on analytical models parametrized using measurements of key metrics (e.g., information propagation delay) of the network. Measurements of the network have been presented by Donet et al. [3], while results from continuous measurements are published by the project *bitnodes.io*. It was shown that the distribution of IP addresses in the Bitcoin network does not

resemble a uniform distribution, but has a strong bias towards a few autonomous systems [4]. As this distribution affects the connectivity of the network, our simulation model emulates the behavior of the Bitcoin reference client in this matter.

Analytical models were used to assess security properties of Bitcoin in previous studies. The feasibility of *Double Spending* attacks in fast payments has been studied using an analytical model derived from measurements as well as experiments in the real Bitcoin network [7]. Although experiments in the real network are possible, they are not only limited by the researcher's resources but also by legal issues. As the main goal of Bitcoin's peer-to-peer network is maintaining information consistency, several works show the importance of information propagation latency on, e.g., block chain forks [2]. In [5], formal proofs of block chain properties are presented based on the assumption of high degrees of synchronization. The main limitation of analytical studies is the difficulty of modeling the complex peer interactions resulting from the client implementation, inhibiting the evaluation of specific, especially network-based, attacks.

Transformation of native network application's code into simulation models has previously been achieved by providing a virtual operating system interface [12] or full hardware virtualization [13]. Shadow [6] executes the Tor application in a simulation environment and supports Bitcoin using a plugin[1]. However, we argue that executing the expensive block chain interactions and cryptographic operations performed by each Bitcoin client at full accuracy inhibits the scalability of experiments. Instead, for full-scale studies of the network at acceptable performance, one needs to abstract from aspects of the client behavior irrelevant to the given type of study. As this abstraction requires manual effort and increased need for validation, we base our simulation model on large segments of unmodified *bitcoind* client code. Hence, experiments benefit both from scalability through abstraction as well as from high accuracy in the relevant client behavior.

III. Simulation Model of the Bitcoin Network

In this section, we detail the simulation model created based on the reference client code and measurements of the Bitcoin network. The purpose of our model is to study attacks that rely on the network topology to influence information flows in the network, e.g., by isolating groups of peers or by providing spurious information to specific peers. The network topology is predominantly defined by the procedures by which the reference client creates connections to remote peers as well as by the distribution of the peer's lifetimes. As we are interested in information propagation delays in the network, we additionally require an estimate of the link latencies between peers. We model these aspects accurately by integration of unmodified client code and by measurements of the conditions in the real-world Bitcoin network. By focusing on attacks relying on the network topology, we can abstract from computationally expensive cryptographic operations in the client code to allow for experiments at full scale of the Bitcoin network at acceptable performance. In the following, we describe the Bitcoin protocol, the measurements performed to parametrize our model as well as the transformation of the client code to a discrete-event simulation model.

[1]https://github.com/amiller/shadow-plugin-bitcoin/

A. Protocol & Client Behavior

We will now give a brief sketch of relevant aspects of Bitcoin; a more detailed description can be found in [10] or directly in the source code of the reference implementation[2]. The two main objects in the Bitcoin protocol are *transactions* and *blocks*. A transaction contains (among others) one or more payers, one or more payees, and an amount to be paid. Transactions are broadcasted through the network and have to be cryptographically signed by the payer in order to be accepted by other network peers. A transaction can be seen as a right for the payee to further spend a stated amount of Bitcoins. The set of all transactions is often compared to a public ledger, which can be used to cryptographically verify whether the payer of a transaction has the claimed rights. As the set of transactions can be inconsistent among different network peers, blocks are used to reach a consensus on the set of transactions. A block contains a set of transactions and a *proof-of-work*. Each block references a predecessor forming the *block chain* – a set of blocks that are authoritative for accepted transactions. The used proof-of-work mechanism links the ability to create blocks to computation power, which prevents successful sybil attacks: An attacker can join the network with many peers, only limited by the attacker's network resources. However, this does not improve the attacker's chances to create new blocks, as this solely depends on the attacker's computational power. In Section VI we will analyze the influence of such an attack on the network itself.

In order to keep information as consistent as possible, new transactions and blocks are flooded through the network. For this, peers announce the availability of new information using an `INV` message containing the hash of the new information to connected peers. The receivers of these `INV` messages check whether they have already received the announced information, and, if the information is new, request the information by responding with a `GETDATA` message. As a response to the `GETDATA` request, the transaction or block will be sent as a `TX` or `BLOCK` message, respectively.

There are several mechanisms that peers wishing to join the network can use in order to retrieve IP addresses of other network peers. First, a number of DNS servers returning IP addresses of peers are hard-coded in the client code. Once a peer is connected to the network, it broadcasts its own IP address so that other peers are aware of it and can potentially connect to the new peer. By issuing `GETADDR` messages it is also possible to receive IP addresses known by another peer up to a maximum of 23 % of the peer's known addresses or 2,500 addresses. Known IP addresses are held persistently, which allows restarting the client while maintaining a large set of seed IP addresses.

The reference implementation aims at preventing an attacker from gaining a large share of the peers a single Bitcoin node connects to. Based on the assumption that the attacker controls only addresses from a limited IP address space, peers to connect to are organized into buckets according to IP address ranges. In the case of IPv4, the two most significant bytes (/16 network) are used. Hence, an attacker can only fill a limited number of buckets with addresses of peers under his control. Selection of addresses from the buckets for connection

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

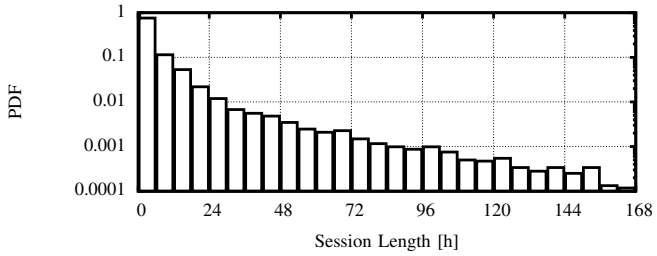

Fig. 1. Measured session lengths of peers in the Bitcoin network.

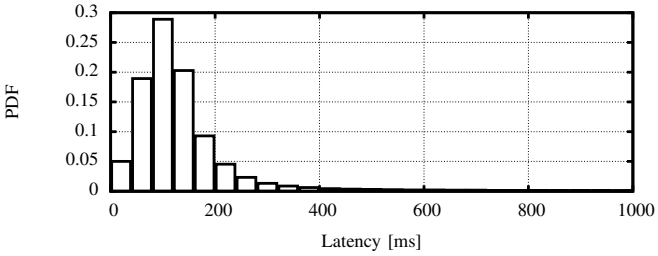

Fig. 2. Measured latency distribution between measurement node and peers in the Bitcoin network.

attempts is performed by a random process with a bias towards peers that have been seen by the network lately.

We assume that all nodes in the network are represented by the behavior of the *bitcoind* reference client. Measurements [14] show that the vast majority of peers is in fact use bitcoind. Still, on the example of a peer connected to all other peers in the network, it was previously shown that a deviation from the default behavior can have a severe impact on the overall network [2].

B. Measurements and Parametrization

In order to parametrize the model correctly, we performed measurements in the real Bitcoin network as well as extracted data from existing measurements. As the behavior is determined by the client implementation, the most influential network parameters are the distributions of the peer's *session lengths* and *link latencies* between peers. The network topology (i.e., the connectivity graph) is determined by the client behavior and the distribution of session lengths, both of which are part of our model. Therefore, we did not perform dedicated measurements of the exact network topology. Depending on the session length distribution not only the stability of the network varies, but the resulting topology can change substantially as well. For instance, a heavy-tail distribution with few peers that stay in the network for a long period of time causes these peers to establish many connections.

The *bitnodes.io* project[3] performs ongoing crawls of all reachable peers in the Bitcoin network and publishes snapshots of the IP addresses of these peers every five minutes. We used this openly accessible data and extracted the points in time when peers joined or left the network. Fig. 1 shows the distribution of session lengths observed based on data from one week. Although the data indicated that a considerable churn takes place, it should be noted that 2,352 peers did not leave the network during the observation time of one week.

Fig. 2 shows the measured distribution of latencies in the Bitcoin network. The distribution was collected by connecting to around 1,000 network peers and observing a total of 183,000 ping/pong messages. The measured distribution only reflects the latency between our measurement node and other peers and not the latency distribution between all peers in the network. We use this as a rough approximation for our simulation and leave a precise model of the latency distribution as future work. As IP addresses of all Bitcoin nodes are known, it will be possible to gather estimations from real-world measurements of Internet path latencies, e.g., using the iPlane service [9].

[3]http://getaddr.bitnodes.io

IV. Implementation: Transformation from bitcoind to DES model

In this section, we present the steps required to transform the source code of the Bitcoin reference client (*bitcoind*) into a model suited for discrete-event simulation (DES). Discrete-event simulation is characterized by a system state that is modified by *events* occurring at discrete points in simulated time. Activities spanning an interval of simulated time must be modeled using multiple events, e.g., representing the start and end of an activity. First, a brief overview of bitcoind's software architecture is given, before challenges arising from the architecture are discussed and a transformation method is presented.

The bitcoind client is a multithreaded application written in C++. Communication between threads is mainly achieved via message queues; each thread may call blocking functions. Recreating the resulting complex timing behavior in the simulation is the main challenge in modeling the client. We will now sketch the functional and time behavior of the three main threads used for networking in bitcoind:

ThreadOpenConnections/ThreadOpenAddedConnections: These two threads try to establish connections to other peers in the network, until the maximum number of outgoing connections (8 by default) is reached. Selection of hosts to connect to was sketched in Section III-A. Connection attempts are performed using blocking calls. Therefore, the timing behavior of this thread depends on how fast connections can be established or timeouts on unsuccessful connections occur.
ThreadSocketHandler: This thread reads data from sockets and writes them into message queues for later processing. It also writes data that was previously stored in designated outgoing queues to sockets. Again, the calls in socket functions are blocking so timing is affected by the blocked duration (e.g., sending a packet on a saturated link). Iteration over all connections is performed at most every 10 ms. Measurements show that blocking calls can increase this interval time to up to around 60 ms.
ThreadMessageHandler: Messages that were stored in incoming queues by ThreadSocketHandler are processed by this thread. It cycles every 100 ms through all connection's queues and performs the protocol handling itself (e.g., reacting to messages, checking for timeouts). Although the timing of this thread is affected by cryptographic processing delays, the effect on the total cycle duration is less severe than for the other threads.

Modeling a multi-threaded application as a DES model is possible by introducing separate sets of event types for each

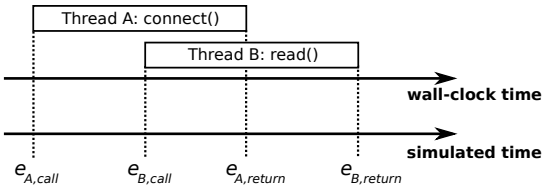

Fig. 3. Transformation of blocking function calls in the original application (top) to a DES representation (bottom).

thread. We identified three main goals that must be taken into account in the modeling process: Explicitness of timing behavior, locality of program state, and functional abstraction.

Timing behavior, which in the original code is implicitly defined by the variable durations of computations and network latencies, must be reflected explicitly in the DES model. For instance, if a simulation event $e_{A,call}$ representing a blocking function call in thread A is executed, a new event $e_{A,return}$ representing a successful return from the function call is scheduled with an appropriate delay in simulated time (cf. Figure 3). In between $e_{A,call}$ and $e_{A,return}$, any number of events representing activities of threads of the same or another client may be executed by the simulator. This transformation is required for every synchronous application code because of DES's inherent asynchronicity.

When modeling a strictly sequential program, continuing after a blocking call is trivial and only requires maintaining the client's state (i.e., variables) across events. However, when introducing complex control structures and function calls, the simulator cannot simply continue *after the call*. Consider the example in Figure 3: If the blocking function read() is called from two different functions in the original code, it must be ensured that the event $e_{B,\mathrm{return}}$ returns to the correct calling function. Therefore, we keep this information (i.e., the *call stack*) for each thread in a separate data structure, enabling the return event to continue at the right caller function.

Locality of program state: The goal of the simulation is to execute several instances of the application in one single simulator. Programmers have several options to keep state in a program. However, if global storage (e.g., static variables) is used, care must be taken to isolate the state of the simulated application instances from each other. Therefore, all state that is stored in global storage must be transferred to a per-instance storage. This transfer is rather trivial and could easily be performed by existing automated model-to-model transformation approaches.

Functional abstractions can on the one hand improve performance and may be necessary to enable simulations of a large number of instances, on the other hand they can potentially lead to deviation of the behavior of the simulation model when compared to the original application code. It should be noted that what is a *suitable* transformation always depends on the ultimate purpose of the simulation model. Hence, different ways to abstract may be suited for different applications and use cases. As we are primarily interested in an analysis of networking aspects of the Bitcoin peer-to-peer network, we abstracted from all cryptographic functions, reducing the computational intensity of the simulation immensely. By omitting procedures such as verification and issuing of signatures as well as block chain maintenance, we make the assumption of a faultless operation regarding these aspects. However, the named procedures do not affect the client's networking behavior that is in our focus. Of course, for studies targeting attacks that depend on a specific behavior of these functions, the corresponding code must be included in the model.

We isolated the simulation model from the underlying operating system by replacing system calls, e.g., for input/output operations, with modifications of in-simulator variables or the scheduling of appropriate event types. As the simulator enables a *god-view* of the network, networking operations can be simplified: for instance, *bitcoind* uses the POSIX select() function to monitor multiple sockets. In the simulation, the reception of packets creates events at the appropriate point in simulated time, making the use of select() unnecessary. Finally, we removed legacy support for older Bitcoin protocol versions and all IPv6-support. As of December 2014, only a small share of peers uses IPv6, enabling a significant performance boost through elimination of the related code. However, as the share of IPv6 nodes is rising, we plan to extend our simulator to also cover IPv6.

V. Validation

In order to validate our simulation model, we compared measurements of how fast information propagates through the real network to the same measurements conducted in simulations. The information propagation delay is well suited for validation as it is affected by many aspects of the Bitcoin network such as client's behavior, network topology as well as networking and processing delays. We will now review information propagation in Bitcoin in detail before we describe our measurement setup and discuss validation results.

As explained in Section III-A, information propagation involves three steps. Forwarding (i.e., sending out `INV` messages to neighbors) is performed either immediately after the reception of the information or some random time later. This mechanism is implemented in order to protect privacy by making it harder to determine which network peer originally issued a transaction. Only 25 % of all `INV` messages are rebroadcasted immediately, the other 75 % are *trickled* out to connected peers. For trickling, every 100 ms one peer is selected out of these peers and the queued `INV` messages are sent. Therefore, the timing of the trickling depends on the number of connections.

The measurements were obtained by maintaining connections to about 1,000 of the 6,000 reachable peers in the network and logging the time stamps of incoming `INV` messages. We deactivated forwarding of `INV` messages in order to avoid *information eclipsing* [2], i.e., situations where peers do not forward information to our measurement peer because they know that we already have this information. As other peers did not receive any `INV` messages from our measurement peer, they were not able to decide whether we had already observed a given message. Hence, the peers forwarded `INV` messages to the monitor node as they became aware of them or during the trickling phase.

For each announced transaction h, the time differences between the first reception of an `INV` message and the subsequent n messages were calculated ($\Delta t_{h,1}$, ..., $\Delta t_{h,n}$). The same

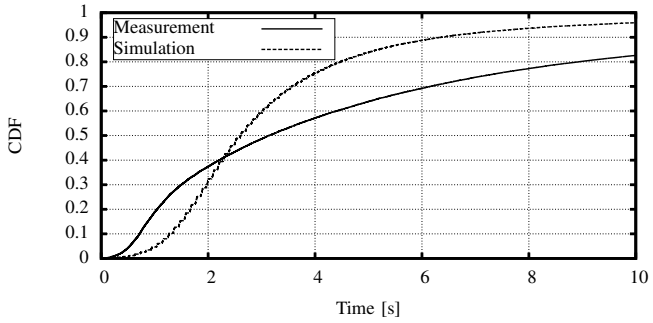

Fig. 4. Comparison of the distribution of $\Delta t_{h,i}$ as measured in the Bitcoin network with simulation results.

method was used in the simulations, where the measurement peer interacted with a simulated network. Obviously, the distribution of these measured time differences $\Delta t_{h,i}$ does not represent the real propagation delay, as the measurements do not indicate when peers receive information, but only when they inform our measurement node. However, it can serve as an upper bound for propagation delay and still exposes the same characteristics required for validation.

Fig. 4 shows the average distribution of $\Delta t_{h,i}$ for the real Bitcoin network and the simulated network. It should be noted that averaging the delays over a large number of individual transactions causes the curve to look very smooth, even though the variance of delays observed for any individual transaction is significantly larger. The results indicate that during the first 2 seconds, information propagation is faster in the real network than in the simulation. However, the propagation rate in the real network declines earlier leading to a worse information propagation after several seconds.

We suspect that the deviation is caused by a number of nodes in the real network that are each connected to large segments of the network. On the one hand, such nodes cause a quick initial acceleration of information propagation due to the large set of peers the message is transferred to initially. On the other hand, the total duration of the subsequent process of trickling messages to the remaining peers increases with larger numbers of connected peers. We leave a further analysis of this issue as future work; nevertheless we state that our simulation model approximates the real network's behavior sufficiently for preliminary evaluations.

VI. Case Study: Partitioning Attack

We will now apply our simulation model to analyze of a potential network partitioning attack.

A. Attack Concept and Attacker Model

We assume that the goal of the attacker is to partition the network into two or more partitions so that no information flow between the partitions is possible. Such an attack does not directly enable the attacker to manipulate actual transaction or block chain data, as this would still require the attacker to be able to solve the required proof-of-work puzzles. Still, such an attack impairs Bitcoin's main functions, potentially causing a decline in users' trust in the system. A possible motivation for an attacker could be speculation on the Bitcoin exchange rate, which is influenced by users' trust in the system.

We assume that the attacker operates a botnet with potentially more than one hundred thousand nodes, which is a realistic size for botnets. Although the number of IP addresses online at a given point in time can be considerably smaller, far more than 10,000 peers may be online simultaneously [11]. Botnets have proven to be able to successfully perform Distributed Denial-of-Service (DDoS) attacks on large services. Hence, we will assume that the botnet is able to also perform DDoS attacks on a limited number of peers in the Bitcoin peer-to-peer network. Although botnets have also been used for mining of Bitcoins, we do not make any assumptions on the botnet's computation power.

The attack itself consists of two phases. In the first phase, the attacker joins the Bitcoin peer-to-peer network with as many nodes as possible and participates conforming to the protocol. However, attacker nodes only announce IP addresses of other attacker nodes, increasing the probability of connections from honest nodes to attacker nodes. The goal of this approach is to thin out the connectivity graph between honest nodes. Once the connectivity graph is sufficiently sparse, the second phase of the attack begins. The attacker nodes in the network stop forwarding transactions and blocks and only respond to messages required to maintain connections to honest nodes. Now, the attacker performs DDoS attacks on honest nodes of the remaining connectivity graph that the attackers believes are on the *minimum vertex cut*. The minimum vertex cut defines the minimum set of peers whose removal causes the connectivity graph to be split into at least two distinct partitions. We will explain in Section VI-B how the minimum vertex cut is determined. In the analysis of this attack, the size of the minimum vertex cut is of interest as it serves as an indicator for the attacker's costs.

As the session length of each node has a severe effect on the number of connections the node establishes, the success of the attack is sensitive to the botnet nodes' churn. Hence, we simulated the attack for varying attacker's session lengths. In order to determine the minimum vertex cut of the network, the attacker needs to be aware of the network topology. It has been shown that it is possible to deanonymize clients by identifying the entry nodes to which clients are connected [1]. We assume that by connecting to a large share of the peers in the network and observing message flows, the attacker can reconstruct an approximation of the network topology. If an insufficiently precise approximation is used and a suspected vertex cut does not actually partition the network, the attacker can repeat the process by observing the remaining connectivity network and selecting additional peers on a newly determined vertex cut.

B. Experiment Setup

In order to evaluate the feasibility of the presented partitioning attack, we performed simulations starting with a number of honest peers. At a certain time stamp the attack begins with the first phase and attacker peers join the network. Periodically, a heuristic approximation of the minimum vertex cut is determined using the *metis* [8] graph partitioning toolkit. It should be noted that in general, there may be both multiple vertex cuts of identical cost and inexpensive cuts that create highly imbalanced partitions. Hence, the choice of a suitable cut depends on the desired imbalance between the resulting partitions. For instance, in a densely connected graph it may

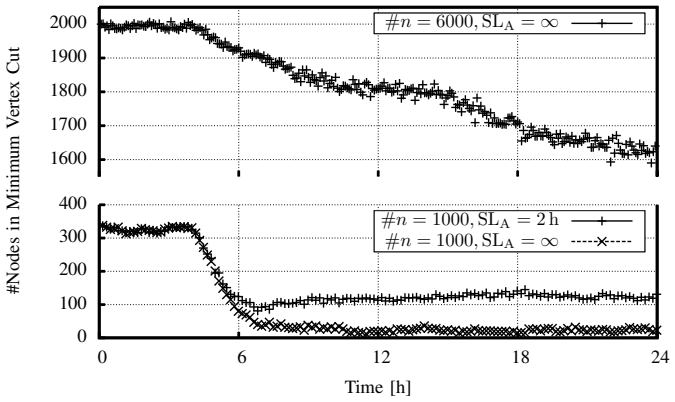

Fig. 5. Number of non-attacker peers on the minimum vertex cut during an attack with 6,000 attacker peers on a network with $\#n = 6,000$ honest peers parametrized as in the real-world network (top) and an attack with 1,000 attacker peers on a network with $\#n = 1,000$ honest peers (bottom); attacks begin at $t = 4\,\mathrm{h}$, attacker's session length $\mathrm{SL_A} \in \{2\,\mathrm{h}, \infty\}$.

be easy to isolate one single vertex by removing all adjacent vertices. In general, the more imbalance is allowed between partitions, the fewer vertices will be part of the vertex cut. In our exemplary attack, the attacker is not interested in creating perfectly balanced partitions, but does aim at creating partitions of non-negligible size. In *metis*, maximum allowed imbalance factors can be specified to represent the attacker's goals.

The simulations were performed on a 2,6 GHz machine equipped with 64 GB of main memory; simulation of 10 hours at real-world scale of around 6,000 peers required 29 hours of computation time and 22 GB of memory.

C. Results

Fig. 5 shows the results of two simulated attacks on a model of the real-world network including latency and churn parameters as presented in Section III ($\#n = 6,000$) as well as on a much smaller network with $\#n = 1,000$ peers. The size of the attacking botnet was chosen to match the number of honest peers in both scenarios. The desired imbalance factor was configured so that the largest partition does not contain more than 60 % of all nodes, hence isolating at least 40 % of nodes. In the large scenario, the attack causes the number of honest peers on the minimum vertex cut to decline from around 2,000 to 1,600 within 24 hours of attack. Although this decline is not yet critical to the stability of the network (as the attacker would still need to perform a DDoS attack on 1,600 peers) attackers with a higher number of peers, a more intelligent behavior or simply more patience might succeed in splitting the network.

The results from the smaller scenario with 1,000 peers illustrate the impact of the attacker's session length SL_A on the success of the attack: Without any churn, the minimum vertex cut declines to less than 50 peers. With an average session length of 2 hours, which equals the session length of honest peers in this small scenario, the minimum vertex cut does remain above 100.

VII. Conclusion

In this paper we presented a first model to enable full-scale simulation of the Bitcoin peer-to-peer network at low runtimes and demonstrated its application with an analysis of a partitioning attack on the network. Validation results indicate a close correspondence between key metrics of the simulated network and the real-world network. The analysis of a partitioning attack showed that the Bitcoin peer-to-peer network is resistant against attackers controlling less than 6,000 bots and an attack lasting several hours. However, attackers with more resources need to be considered.

Currently, our model does not fully model the real-world network's topology, resulting in a remaining deviation when considering the information propagation in the network. Future work will include approximating the network's topology by measurements to improve the accuracy of our model. Additionally, we will extend our model by integrating further aspects of measured user behavior (e.g., realistic generation of transactions). These improvements will allow us to investigate sophisticated attacks relying strongly on the current network state and on protocol specifics.

References

[1] A. Biryukov, D. Khovratovich, and I. Pustogarov. Deanonymisation of Clients in Bitcoin P2P Network. *arXiv preprint arXiv:1405.7418*, 2014.

[2] C. Decker and R. Wattenhofer. Information Propagation in the Bitcoin Network. In *2013 IEEE Thirteenth International Conference on Peer-to-Peer Computing (P2P)*, pages 1–10. IEEE, 2013.

[3] J. Donet Donet, C. Prez-Sol, and J. Herrera-Joancomart. The Bitcoin P2P Network. In *Financial Cryptography and Data Security*, Lecture Notes in Computer Science, pages 87–102. Springer Berlin Heidelberg, 2014.

[4] S. Feld, M. Schönfeld, and M. Werner. Analyzing the Deployment of Bitcoin's P2P Network under an AS-level Perspective. *Procedia Computer Science*, 32:1121–1126, 2014.

[5] J. Garay, A. Kiayias, and N. Leonardos. The Bitcoin Backbone Protocol: Analysis and Applications. Technical report, Technical report, 2014.

[6] R. Jansen and N. Hooper. Shadow: Running Tor in a Box for Accurate and Efficient Experimentation. Technical report, DTIC Document, 2011.

[7] G. O. Karame, E. Androulaki, and S. Capkun. Double-Spending Fast Payments in Bitcoin. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 906–917. ACM, 2012.

[8] G. Karypis and V. Kumar. Metis - Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 2.0. 1995.

[9] 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.

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

[11] B. Stone-Gross, M. Cova, B. Gilbert, R. Kemmerer, C. Kruegel, and G. Vigna. Analysis of a Botnet Takeover. *Security & Privacy, IEEE*, 9(1):64–72, 2011.

[12] H. Tazaki, F. Uarbani, E. Mancini, M. Lacage, D. Câmara, T. Turletti, and W. Dabbous. Direct Code Execution: Revisiting Library OS Architecture for Reproducible Network Experiments. In *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*, pages 217–228. ACM, 2013.

[13] T. Werthmann, M. Kaschub, M. Kühlewind, S. Scholz, and D. Wagner. VMSimInt: a Network Simulation Tool Supporting Integration of Arbitrary Kernels and Applications. In *Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques*, pages 56–65, 2014.

[14] A. Yeow. Bitnodes Project 2014 Q3 Report: The State of Bitcoin P2P Network. https://bitcoinfoundation.org/2014/09/bitnodes-project-2014-q3-report-the-state-of-bitcoin-p2p-network/, 2014.