Analyzing illicit behavior in the Lightning Network

No Author Given

No Institute Given

Abstract. The Lightning Network is one of the most promising techniques to address scalability issues in Bitcoin blockchain. It proposes to offer unlimited instant off the chain transactions at negligible cost and increased on-chain anonymity, while leveraging consistency and public verifiability of Bitcoin. These off-chain payments pose a serious challenge to law enforcement agencies that track illegal payments as all communication happens off the chain. Existing techniques of tracing transactions are based on heuristical analysis of information that is publicly available on the blockchain. Unfortunately, off-chain payments bypass these techniques resulting in the need for development of new heuristics that exploit specific features of the off-chain payment protocol.
This paper presents a heuristic to estimate illicit behavior of nodes in the Lightning Network. The heuristic attempts to associate a risk factor which we call as *sensitivity score* ranging from 0 to 1 for each node, which represents how likely the node is to involve in illegal activities. The proposed technique analyzes the Hashed Time Lock Contracts (HTLC) that are committed to the main chain for correlation of the hash value and deduce the nodes involved in the same payment before assigning their sensitivity levels.

Keywords: Bitcoin · Illegal payments · Lightning Network.

1 Introduction

Bitcoin [8] is a fully decentralized blockchain based digital cryptocurrency that is widely adopted today as an alternative monetary system. The system does not have a trusted third party like a bank and the users are mutually distrusting. Moreover, the system is permissionless as any user interested in participating need not obtain any authority's permission to enter the system. All one needs is a public key to receive payments and an user can have as many public keys as he/she wants.

While this may prove to be more accessible and less tedious, unfortunately such a lack of authorization has opened the gates to users making payments for illegal goods and services [19]. Silk Road [14], was one such online market place where one could pay in bitcoins for illegal drugs, smuggled artifacts, dangerous weapons, and child pornography, among many others. Law enforcement agencies have toiled hard to trace such payments with the public information

that is available in the Bitcoin blockchain. Heuristics like timestamp correlation, input clustering, transaction graphs are the main techniques [9, 10, 15, 13] used to decipher information about the keys and its users involved in payments.

1.1 Off-chain payments

Owing to the limitations of the permissionless and distributed nature of Bitcoin, the transaction processing speed is far slower than traditional payment systems. For instance, Bitcoin allows processing of tens of transactions per second while Visa can process upto 47,000 transactions per second at its peak [17]. With an ever increasing user base for Bitcoin, the number of transaction requests per unit time is certain to increase.

Scalability, rightly so, is considered today an important issue in the Bitcoin community [1, 18]. Several research and industry efforts are dedicated today to address this concern [1, 2, 5, 11, 12]. One of the most promising solutions is the use of Bitcoin payment channels [3, 11, 6] to realize off-chain payments. Lightning Network (LN) is one of the widely adopted Bitcoin Payment Channel proposal. In a nutshell, a pair of users open a payment channel by adding a single transaction to the blockchain where they lock their bitcoins in a deposit secured by a Bitcoin smart contract, a 2-of-2 multisig address as in [11]. Several off-chain payments can be then performed by locally agreeing on the new distribution of the deposit balance. Finally, the users sharing the payment channel perform another Bitcoin transaction to add the final balances in the blockchain, effectively closing the payment channel.

In this manner, the blockchain is required to open and close a payment channel but not for any of the (possibly many) payments between users, thereby reducing the load on the blockchain and improving the transaction throughput. This simple approach can be extended to payments between users who do not share an open channel but have a path of open channels between them via multiple hops with enough capacity to settle their payments, effectively creating a payment-channel network (PCN) [11].

1.2 Our contribution

Existing heuristic analysis [9, 10, 15, 13] perform poorly in the context of off-chain payments made via payment channels as the amount of communication happening via the blockchain is much less than before. In this paper, we develop a heuristic to analyze nodes involved in the Lightning Network payments. Nodes involved in the same payment share a common attribute in their Hashed Time Lock Contracts (HTLC) which we will discuss in detail in the Background section. More specifically, their HTLCs have the same *hash of pre-image* value. Our heuristic is to assign sensitivity scores to nodes participating in the same payment in Lightning Network. A sensitivity score estimates how likely each node is to involve in illicit transactions. Under the assumption that 7% of nodes are highly sensitive (indulging in illegal activities), we implement our heuristic and apply it to a small sample of transactions in LN. In the topography we

deciphered, we observed that 28% of the nodes are highly likely to involve in illegal activities.

2 Background

In this section we give a brief background on Payment Channel Networks and its popular candidate, The Lightning Network.

2.1 Payment Channels

A payment channel network enables several Bitcoin payments between two users without committing every single payment to the Bitcoin blockchain. The core idea of a payment channel between two users is depositing bitcoins into what is called a multisig address (also known as joint account in the traditional sense) controlled by both users and having the guarantee that all bitcoins would be refunded at a mutually agreed time if the channel expires. In the following, we overview the basics of payment channels.

Consider an example where, Alice opens a payment channel with Bob with an initial capacity of 10 bitcoins by sending a transaction onto the blockchain which is termed as Funding transaction. This opening transaction makes sure that Alice gets the money back after a certain time has passed if the payment channel is not used/closed. Now, Alice can pay to Bob 3 bitcoins by adjusting the balance of the deposit (i.e) 7 bitcoins belongs to Alice and 3 bitcoins to Bob and this happens off-chain. When no more off-chain payments are needed (or the capacity of the payment channel is exhausted), the payment channel is closed with a closing transaction that is included in the blockchain. This transaction sends the deposited bitcoins from the multisig address to each user according to the most recent balance in the payment channel. In this case, the balances are 7 bitcoins to Alice and 3 bitcoins to Bob.

The above example is a case of uni-directional channel where Alice can pay to Bob and not the other way. This can easily be extended to bi-directional payments using standard techniques and we refer the reader to [5, 11, 16] for details.

2.2 Payment Channel Network

We use the notations as described in [6]. A PCN [7] can be represented as a directed graph $G = (V, E)$, where the set V of vertices represents the Bitcoin accounts and the set E of weighted edges represents the payment channels. Every vertex $u \in V$ has associated a non-negative number that denotes the fee it charges for forwarding payments. The weight on a directed edge $(u_1, u_2) \in E$ denotes the amount of remaining bitcoins that u_1 can pay to u_2. Such a network can be used then to perform off-chain payments between two users that do not have an open channel between them but are connected by a path of open payment channels. The success of a payment between two users depends on the capacity

available along a path connecting the two users and the fees charged by the users in such path. Assume that a sender s wants to pay α bitcoins to a recipient r and that they are connected through a path $s \rightarrow u_1 \rightarrow \cdots \rightarrow u_n \rightarrow r$. For their payment to be successful, every link must have a capacity $\gamma_i \geq \alpha'$, where $\alpha' = \alpha - \Sigma_{j=1}^{i-1} fee(u_j)$ (i.e., the initial payment value minus the fees charged by intermediate users in the path). At the end of a successful payment, every edge in the path from s to r is decreased by α'_i. To ensure that r receives exactly α bitcoins, s must start the payment with a value $\alpha^* = \alpha + \Sigma_{j=1}^{n} fee(u_j)$. For example, assume that Alice wants to pay Bob 5 bitcoins. There is a channel between Alice and John, John and David, David and Edward, and finally between Edward and Bob. Each node charges 0.25 bitcoins as fees. So, Alice needs to start a payment for a value of 5.75 bitcoins (5 bitcoins plus 0.75 bitcoin for the fees charged by users in the path). Then the payment is settled as follows: capacity in the link Alice $\rightarrow$ John is reduced by 5.75. Then, John charges a fee of 0.25 bitcoins. Subsequently the capacity of the link John $\rightarrow$ David is reduced by 5.50 instead of 5.75 bitcoins. Following the same reasoning, the link David $\rightarrow$ Edward is reduced by 5.25 and the link Edward $\rightarrow$ Bob is reduced by 5 as originally intended.

2.3 Lightning Network

Lightning Network is a PCN which in its current proposal consists of a smart contract called Hashed Time-Lock Contract (HTLC) [11]. This contract locks x bitcoins that can be released only if the contract is fulfilled. The contract is defined, in terms of a hash value $y := H(R)$ where R is chosen uniformly at random, the amount of bitcoins x and an expiry timeout t , as follows:

HTLC(**Alice**,**Bob**,x, y, t):

1. If Bob produces the condition R such that $H(R) = y$ before t days, Alice pays Bob x bitcoins.
2. If t days elapse, Alice gets back x bitcoins.

Lets consider an illustrative example Fig. 1, Alice wishes to pay 1 bitcoin to Fabi with whom there is a payment channel path in the form of Alice $\rightarrow$ Bob $\rightarrow$ Edward $\rightarrow$ Fabi, where each channel has sufficient capacity. For simplicity, we assume that there are no payment fees in this example. First, the payment amount (i.e., 1 bitcoin) is set on hold from the sender to the receiver and then released from the receiver to the sender. In a bit more detail, the receiver (Fabi) sends the condition y to the sender (Alice), Alice sets an HTLC with her neighbor Bob, effectively setting the payment value (i.e., 1 bitcoin) on hold. Such HTLC is then set at each payment channel (3 in this example) in the path to the receiver Fabi. At this point, Fabi knows that the payment value is on hold at each payment channel and thus she reveals the value R to Edward, which allows her to fulfill the contract with Edward. This enables the intermediaries to learn the value of R and to settle the new capacity at each payment channel in the path.

It is important to note that every user in the path sets the HTLC in the outgoing payment channel with a timeout smaller than the HTLC in the incoming payment channel. This means that the HTLC timeout between Alice and Bob will be greater than that between Bob and Edward, and similarily the HTLC timeout between Bob and Edward will be greater than that between Edward and Fabi. In this manner, the user makes sure that she can pull bitcoins from her predecessor after her bitcoins have been pulled from her successor.

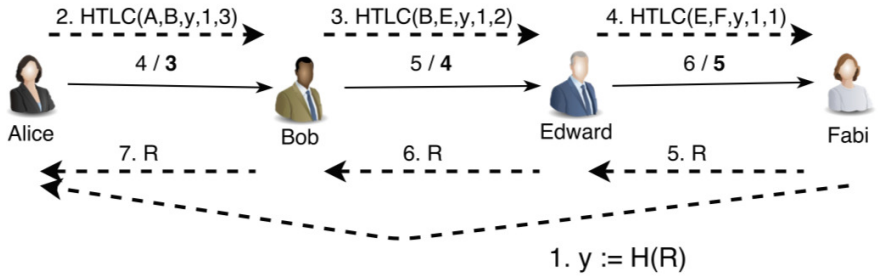

Fig. 1. Illustrative example of a payment from Alice to Fabi for value 1 using HTLC contract. First, the condition is sent from Fabi to Alice. The condition is then forwarded among users in the path to hold 1 bitcoin at each payment channel. Finally, the receiver Fabi reveals R to Edward and subsequently everyone in the channel, releasing the held bitcoin at each payment channel. For simplicity, we assume that there are no payment fees in this example [6].

3 Analyzing illegal nodes in off-chain payments

The recent off-chain payment techniques as discussed before offer solutions to various issues pertaining to the Bitcoin blockchain. As these off-chain payments happen locally between users and do not reflect adequately on the blockchain, it naturally poses a challenge to the law enforcement agencies tracking sensitive transactions based on the information published on the chain. In an attempt to address this challenge, we propose new heuristic that help in tracking nodes involved in LN transactions in Bitcoin. In this section we explain our heuristic and implementation level details.

3.1 Our Heuristic

LN in Bitcoin uses a special contract known as Hashed Time-Lock Contracts (HTLC) as discussed in Section. 2.3. These contracts enable payment between two users through a payment channel upon revealing a pre-image of a hash

value y. In the current implementation, several payment channels can be chained together with HTLCs for each of them albeit with the same hash value y to allow payments across two users who do not share a direct payment channel. It is easy to see that the value of the hash y uniquely identifies the users that took part in a specific transaction. This fact has two main implications. First, any two cooperating users in a path can trivially derive the fact that they took part in the same payment and this can be leveraged to reconstruct the identity of sender and receiver. Second, if the HTLC statements are uploaded to the blockchain (e.g., due to non-cooperating intermediate users in the payment path), an observer can track the complete path used to route the payment, even if he/she is not part of the payment.

On deciphering multiple nodes that have taken part in a payment using the the same hash value y, we order them based on the timeout parameter in each of their HTLCs. We then assign what we call as *sensitivity score* to each node based on prior knowledge and their position in the ordering. On a broader level, a node having a payment channel with another node with high level of sensitivity inherits a higher level of sensitivity than a node that is found to be an intermediary in the payment. Also, the first and the last node in the ordering receive similar level of sensitivity. The rationale behind this heuristic is that if one of the end points, either the sender or the receiver is potentially a miscreant, then the other node too is a miscreant with high probability. Since we cannot be absolutely sure about the end nodes of an ordering being the sender and the receiver as not all the HTLCs involved in the transaction would land up in the blockchain, our heuristic avoids assigning a high level and rather settles for assigning both nodes a similar level. Based on the level at which we have each node, we can conclude the potential misbehavior of the node in case of tracking illegal payments.

3.2 Our Implementation

We collected 36 closed channel information from [4] and validated our heuristic. Closed channel data is of our interest as there could be some channels that are closed/spent via HTLC. Among the collected transactions, we observed 42 unique nodes and 7 unique pre-image hashes. We have implemented our heuristic in Python (3.6.7) using `csv`, `pandas` and `numpy` libraries.

Data Extraction We extracted data from closed channels [4]. Here we extract the following: the public key of the two nodes involved in creating the payment channel, the transaction id of the transaction that creates the channel, the channel id, the capacity of the channel with last update time and the policy details like the minimum HTLC, lock time etc.

We obtained HTLC transactions on the blockchain by analyzing the spend transactions of the extracted closed channel transactions. Briefly, in order to spend an output of transaction Tx, transaction Tx' starts by specifying the hash of Tx (transaction ID) and a witness x that satisfies the output script in Tx.

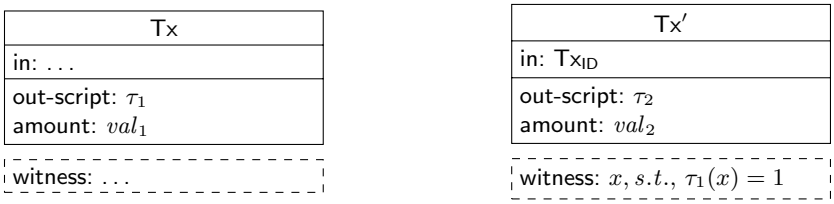

Fig. 2. The structure of a transaction in Bitcoin. The transaction Tx' is spending the output τ_1 of transaction Tx. The transaction ID $\mathsf{Tx}_{\mathsf{ID}}$ is computed as the hash of Tx without its witness. The witness field is also called the spending script.

This is explained in Fig. 2. And Table. 1 shows the different types of output scripts and their corresponding spending scripts. In Bitcoin, the output of the closed channel transactions are of type P2WSH and therefore the corresponding spending transactions need to specify a spending script containing the whole witness script along with the witness. Thus, to collect the required data - public key of the nodes, channel id, transaction id, and hash of the pre-image used to spend with the timestamp, that are specified in the witness script, we analyzed the spending transactions of the extracted closed channel transactions.

Script Type	**Segwit?**	**Output script** (τ)	**Spending script**(x)
P2SH	No	Hash(script)	data_for_script, script
P2PKH	No	Hash(public_key)	signature, public_key
P2WSH	Yes	Hash(witness_script)	witness(es), witness_script
P2WPKH	Yes	Hash(public_key)	signature, public_key

Table 1. Different types of output scripts and the corresponding spending scripts. We consider here the most common types of scripts: Pay to Script Hash (P2SH), Pay to Public Key Hash (P2PKH), Pay to Witness Script Hash (P2WSH), Pay to Witness Public Key Hash (P2WPKH). Segwit compatibility is a feature in Bitcoin where the spending script of a transaction is stored separately with a placeholder called witness. The data in the witness will not contribute to the ID of the transaction.

Implementation of the heuristic After extracting the data, we make an assumption that 7% of the nodes are involved in illicit transactions. Let's call such nodes as *highly sensitive* nodes. The sensitivity score of highly sensitive nodes is given as 1 whereas for other nodes the score is initialized to 0.

We consider two factors to infer the influence of highly sensitive nodes on other nodes. First, nodes having direct payment channel with the highly sensitive nodes have higher chance of involving in illicit transactions. By this rationale, our heuristic assigns a score of 0.7 to nodes that have payment channel with highly sensitive nodes as shown in Fig. 3. Two highly sensitive nodes can have a payment channel with each other, so to avoid updating those scores we choose max(0.7, *self*) where *self* is the sensitivity score of itself.

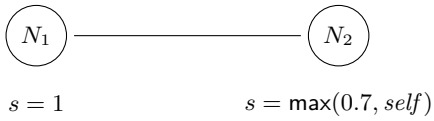

Fig. 3. Payment channel between two nodes N_1 and N_2 where N_1 is highly sensitive with a sensitivity score $s = 1$.

Second, involvement of highly sensitive nodes in multi-hop HTLC transaction path will increase the sensitivity score of other nodes in the path. The path can be reconstructed from the hash of the pre-image from the transactions. Although we might not get the complete path as the transactions happen off-chain, we can infer details of the nodes from the partially constructed path. Fig. 4, 5, 6, 7 shows various scenarios of the reconstructed path.

When the constructed path is continuous as shown in Fig. 4, our heuristic assigns $\mathsf{max}(0.7{*}\mathrm{s}, \mathit{self})$ as the sensitivity score to the start node (sender node) and $\mathsf{max}(0.2{*}\mathrm{s}, \mathit{self})$ to the intermediate nodes where s is the sensitivity score of the end node (recipient node). The rationale behind this assignment is that a node will send amount to a highly sensitive node with malicious intent is highly probable. So, the sensitivity of start node depends on that of the end node. We refrain from assigning the same score as we will not be certain of start and end nodes as not all HTLCs get onto the blockchain. Although the intermediate nodes did not spend money on illicit transaction, they aided it. So, we believe giving them a lower score makes sense rather than leaving the score unchanged or 0. Throughout the above explanation we assumed that the end node is highly sensitive. If it is not the case and one of the intermediate nodes is highly sensitive then there is a fair chance that the transaction is illicit. So, we update the score of the end node to $\mathsf{max}(0.5, \mathit{self})$ and apply the same heuristic as above.

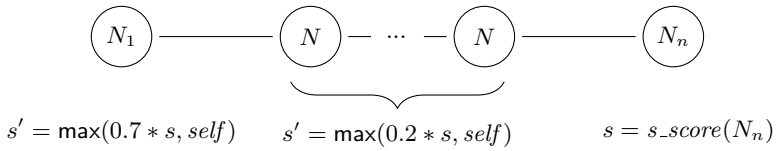

Fig. 4. Sensitivity scores (s') of nodes in a path where the order is known from the start to the end node. If highly sensitive nodes are observed on the path, then $s = \mathsf{max}(0.5, s)$

When we do not know the start node owing to the discontinuity as shown in Fig. 5, we follow the same heuristic as explained above with an additional rule that both the nodes involved in the first transaction will be assigned $\mathsf{max}(0.7{*}\mathrm{s}, \mathit{self})$.

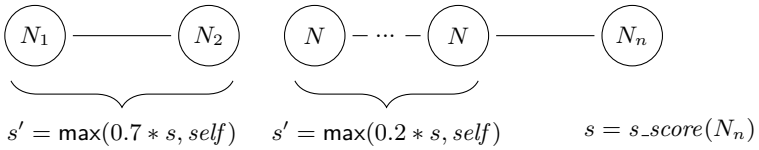

Fig. 5. Sensitivity scores (s') of nodes in a path where the order of the start node is unknown, and highly sensitive nodes are not found in the path. If highly sensitive nodes are found on the path, then $s = \mathsf{max}(0.5, s)$.

For the case when the end node cannot be decided as shown in Fig. 6, we added a rule that the score is the average of scores of the two nodes in the last transaction and we apply our heuristic to update the start and intermediate nodes. We update the score to $\mathsf{max}(0.5,\ s)$ if we see highly sensitive nodes on the path and then apply our heuristic.

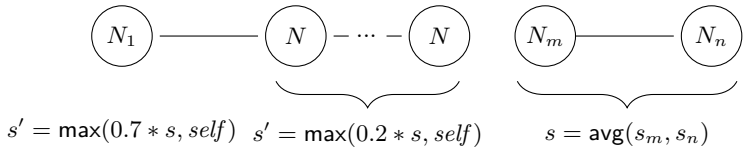

Fig. 6. Sensitivity scores (s') of nodes in a path where the order of the end node is unknown, and highly sensitive nodes are not found in the path. If highly sensitive nodes are found on the path, then $s = \mathsf{max}(0.5, s)$.

In the case of not knowing both the start and the end nodes as shown in Fig. 7, we apply all the individually discussed cases here. The sensitivity score of end node will be the average of the two nodes and it will be updated based on the presence of highly sensitive nodes on the path. The start nodes, namely the two nodes involved in the first transaction will get $\mathsf{max}(0.7$*s, $self)$.

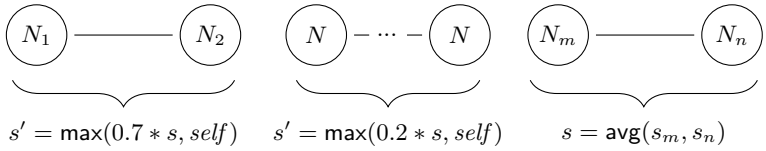

Fig. 7. Sensitivity scores (s') of nodes in a path where the order of the start and the end nodes are unknown, and highly sensitive nodes are not found in the path. If highly sensitive nodes are found on the path, then $s = \mathsf{max}(0.5, s)$.

Intuition We present the pseudo code of our heuristic in Algorithm 1. The inputs to the algorithm are *Transactions* and *Illegal_nodes*, which are lists of transactions and prior knowledge of illegal nodes, respectively. The output is the sensitivity score of all the nodes which is stored in *scores*. Line:1 initializes the score of all nodes to 0 while Line:2 updates the score of all *Illegal_nodes* to 1. Line:3-5 implement the rule shown in Fig. 3 which is updating the score of nodes that have direct payment channel with *Illegal_nodes* to 0.7. Line:6-10 find the unique hashes and the indices of the transactions where those hashes are involved and ordered by *timestamp* which is needed to implement the other rules in our heuristic. Updating of scores as per our heuristic based on Fig.4, 5, 6, 7 happen in Line:11-20.

Algorithm 1: assignSensitivityScore()

input : *Transactions*[] is a list of transactions ordered by timestamp with each transaction of the form ($node_1$, $node_2$, *hash_of_preimage*, *timestamp*)
Illegal_nodes[] is a list of nodes known to involve in illegal activities
output: *scores*[] is a list of sensitivity scores for each node

1: For each node denoted by node, set *scores*[node] = 0
2: For each node denoted by node ∈ *Illegal_nodes*, set *scores*[node] = 1
3: **for** $tx \in$ *Transactions* **do**
4: | **if** $\text{node}_1(tx) \in$ *Illegal_nodes and scores*[$\text{node}_2(tx)$] = 0 **then** Update *scores*[$\text{node}_2(tx)$] = 0.7;
5: | **if** $\text{node}_2(tx) \in$ *Illegal_nodes and scores*[$\text{node}_1(tx)$] = 0 **then** Update *scores*[$\text{node}_1(tx)$] = 0.7 ;
6: *Unique_hashes*[] is the set of unique hash values observed;
7: *hash_tx_dictionary*[] is a dictionary that stores the index of transactions containing the same hash values;
8: **for** *hash_value* ∈ *Unique_hashes* **do**
9: | **for** $tx \in$ *Transactions* **do**
10: | | **if** hash_of_preimage(tx) = *hash_value* **then** Update *hash_tx_dictionary*[*hash_value*] by appending index(tx) to it;
11: s is the value used for updating sensitivity scores of nodes on the path ;
12: **for** *hash_value* ∈ *Unique_hashes* **do**
13: | *indices* = *hash_tx_dictionary*[*hash_value*] ;
14: | **if** is_end_node_known(*Transactions*, *indices*) **then** s = *scores*[end_node];
15: | **else** Let s_m, s_n be the scores of the nodes involved in last transaction;
16: | | $s = \text{avg}(s_m, s_n)$;
17: | **if** is_illegal_node_involved(*Transactions*, *indices*, *Illegal_nodes*) **then** $s = \max(0.5, s)$;
18: | **if** is_start_node_known(*Transactions*, *indices*) **then** *scores*[start_node] = max(0.7∗s, *scores*[start_node]);
19: | **else** Let N_1, N_2 be the nodes of the first transaction;
20: | | *scores*[N_1] = max(0.7*s,*scores*[N_1]);
21: | | *scores*[N_2] = max(0.7*s,*scores*[N_2]);
22: | *scores*[other_nodes] = max(0.7∗s, *scores*[other_nodes]);

3.3 Validation

We validate our heuristic using the extracted data which is pictorially represented in Fig. 8. We show different nodes along with their payment channel connections with other nodes and their sensitivity scores.

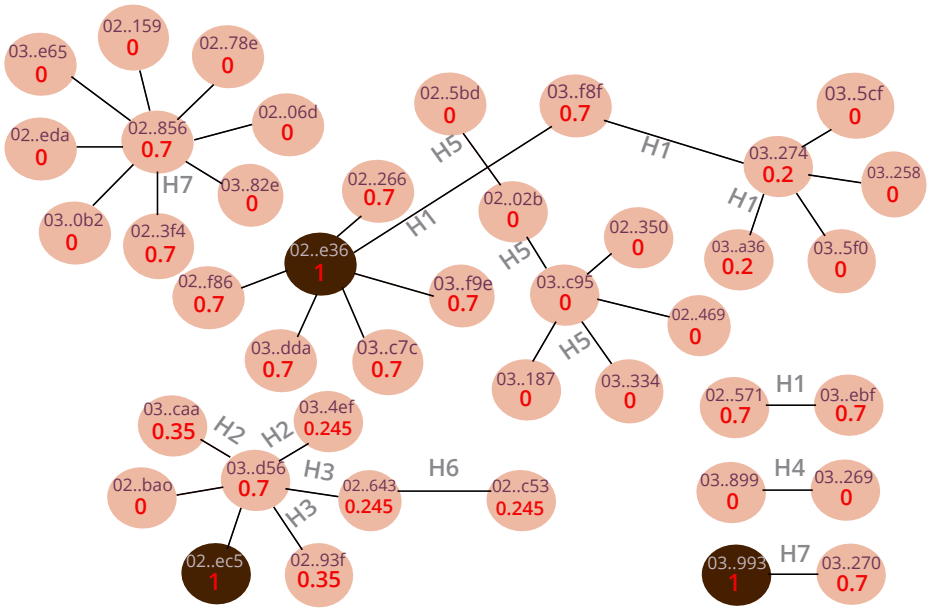

Fig. 8. Each circle represents a node. The public key of the node is the long text within the circle. Full key is not disclosed here as we have made assumptions about their sensitivity. H1,H2,..,H7 are the hashes (of pre-images). Sensitivity of each node calculated as per our heuristic is shown in bold red text within the circle. Highly sensitive nodes are marked as the dark circles.

4 Conclusion and Future work

We briefly present the inner workings of Bitcoin and its facilitation of illegal transactions. Given the lack of authority, nodes indulging in illicit activities exploit the Bitcoin network and make transactions for illegal goods and services. The development of off-chain payments has only made it easier for such behavior as less information is put onto the chain. In this work we develop a new heuristic tailored for the Lightning Network protocol that helps in analyzing behavior of nodes and make informed guesses about its involvement in illegal transactions. We exploit the features of Lightning Network which has a special Bitcoin smart contract, namely the HTLCs. We implement our heuristic by considering a set

of 42 nodes who participated in Lightning Network transactions and decipher each node's involvement in different payments based on their HTLCs. With an initial assumption of 7% of nodes being illegal, we learn that close to 28% of the nodes are *highly likely* to be illegal.

As part of future work we believe our heuristic can be further scaled to a larger set of nodes over a longer span of time. Also, we only use the same hash value across different HTLCs to decipher connections within the same payment. One can exploit other features of the HTLC, for example, the payment time-out can be correlated across different HTLCs and cluster them as being involved in the same payment path. Our heuristic presented here is *passive learning* in the sense that we observe the transactions uploaded on to the chain. One could also setup nodes in the Lightning Network and thereby learn information from the inside by involving in different payments. This type of learning is more *real-time* and can be effective in many scenarios.

References

1. Bitcoin wiki: Bitcoin scalability faq. https://en.bitcoin.it/wiki/Scalability_FAQ.
2. Bitcoin wiki: Block size limit controversy. https://en.bitcoin.it/wiki/Block_size_limit_controversy.
3. Bitcoin wiki: Payment channels. https://en.bitcoin.it/wiki/Payment_channels.
4. Closed Channel Data https://1ml.com/channel?order=closedchannels&json=true
5. Decker, C., and Wattenhofer, R. A fast and scalable payment network with bitcoin duplex micropayment channels. In Stabilization, Safety, and Security of Distributed Systems (2015).
6. Malavolta, G., Moreno-Sanchez, P., Kate, A., Maffei, M., and Ravi, S. (2017, October). Concurrency and privacy with payment-channel networks. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (pp. 455-471). ACM.
7. McCorry, P., Mser, M., Shahandashti, S. F., and Hao, F. Towards bitcoin payment networks. In Australasian Conference Information Security and Privacy (2016).
8. Nakamoto, S. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf, 2008.
9. Ober, M., Katzenbeisser, S., and Hamacher, K. Structure and anonymity of the bitcoin transaction graph. Future internet, 5(2):237250, 2013.
10. Ortega, M.S. The bitcoin transaction graph anonymity. Masters thesis, Universitat Oberta de Catalunya, June 2013.
11. Poon, J., and Dryja, T. The bitcoin lightning network: Scalable off-chain instant payments. Technical Report. https://lightning.network/lightning-network-paper.pdf.
12. Prihodko, P., Zhigulin, S., Sahno, M., and Ostrovskiy, A. Flare: An approach to routing in lightning network. http://bitfury.com/content/ 5- white- papers- research/whitepaper_flare_an_approach_to_routing_in_lightning_network_7_7_2016.pdf.
13. Ron, D. and Shamir, A. Quantitative analysis of the full bitcoin transaction graph. IACR Cryptology ePrint Archive, 2012:584, 2012.
14. Ron, D., and Shamir, A. (2014, March). How did dread pirate roberts acquire and protect his bitcoin wealth?. In International Conference on Financial Cryptography and Data Security (pp. 3-15). Springer, Berlin, Heidelberg.

15. Reid, F. and Harrigan, M. An analysis of anonymity in the bitcoin system. CoRR, abs/1107.4524, 2011.
16. Russell, R. Reaching the ground with lightning. Technical Report. http://ozlabs.org/ rusty/ln-deploy-draft-01.pdf.
17. Stress test prepares visanet for the most wonderful time of the year. Blog entry. http://www.visa.com/blogarchives/us/2013/10/10/stress-test-prepares-visanet-for-the-most-wonderful-time-of-the-year/index.html.
18. Torpey, K. Brock pierce: Bitcoin's scalability issues are a sign of its success. Blog entry. https://bitcoinmagazine.com/articles/brock-pierce-bitcoin-s-scalability-issues-are-a-sign-of-its-success-1459867433/.
19. Turner, A., and Irwin, A. S. M. (2018). Bitcoin transactions: a digital discovery of illicit activity on the blockchain. Journal of Financial Crime, 25(1), 109-130.