Exhaustive Trust Algorithm: A Mitigation to Flood and Loot Attack of Lightning Network

Arman Dave, Catherine Johnson, Gil Banuelos, Evan Seeyave

May 2022

Contents

1 Introduction

The Bitcoin Network was originally invented in 2009 so that people could conduct electronic transactions in an efficient, safe, and anonymous manner. Bitcoin is a universal currency that ideally can be used for both overseas transactions, and an everyday transaction at the grocery store. Today, although it can be used for transactions, Bitcoin generally is used as a store-of-value currency, since while Bitcoin transactions are very secure, they are also very slow and expensive. The benefits of Bitcoin make it worthwhile to attempt to fix these downfalls that currently exist.

In its current state, Bitcoin cannot be used for regular transactions, such as buying coffee, because a transaction on the Bitcoin Blockchain is too slow and too expensive. Approximately 7 Bitcoin transactions can be performed per second - by comparison, Visa can perform approximately 1,700 transactions per second.

As a potential fix to Bitcoin's performance issue, the Lightning Network [7] was invented. The Lightning Network is a network that lies above the Bitcoin infrastructure (refer to section 2 for more information on the Lightning Network). It has been proven that the Lightning Network can increase Bitcoin's transactions per second from 7 to several millions. However, since the Lightning Network is still very new, it has vulnerabilities that compromise the safety of transactions performed on the network - namely, Flood and Loot (see section 3 for details).

The following paper focuses on mitigation strategies for the Flood and Loot Attack [3]. Specifically, this paper outlines Exhaustive Trust Algorithm (ETA), a reputation-based algorithm we proposed that disincentives adversarial users from attacking the Lightning Network. Additionally, we propose an experiment that analyzes the effectiveness of the algorithm implementation.

The paper is structured as follows: In Section 2, we explore the technical construction of the Lightning Network, including the mechanisms by which two parties can transact. In Section 3, we summarize the findings of the Flood and Loot paper, which exploits holes in Lightning's construction to steal large chunks of money. Given that Flood and Loot was only discovered in 2020, proposed solution in the literature are sparse and under-researched; nevertheless, in Section 4, we analyze the merits of different proposals. In Section 5, we propose ETA, describe its design, and give a proof that shows a rational attacker cannot profit when Lightning nodes follow the ETA protocol; we further outline a learning experiment to connect the ETA to broadcast delta. Finally, in Section 6, we conclude our work and discuss future avenues of research.

2 What is the Lightning Network?

The goal of the Lightning Network is to allow for fast transactions of Bitcoin. Blockchain transactions on Bitcoin are currently very slow, so the Lightning Network allows for secure transactions off of the blockchain, which are much faster [1].

2.1 How Lightning Works

The Lightning Network is a layer 2 blockchain that runs on the Bitcoin network. The way the network does this is by allowing for transactions to happen off-chain. A multi-signature payment channel is opened between two users. They can exchange payments quickly back and forth within the channel. Once they are ready to cash out their payments they go to the blockchain to complete the transaction. In this way, the blockchain is only used for the initial opening and the final closing of the channel. They can exchange hundreds of payments at a lightning-fast rate and still use the integrity provided by the blockchain to receive the proper amount owed. In theory, no user has to trust any specific participant of the network because of HTLCs (hash-time-locked-contracts), which ensure that even if one member disappears the other can still close the channel claim their money on the blockchain.

Additionally, the lightning network allows for multi-hop payments, a network of channels that allow you to send money to a user through a specific path. This also speeds up the network and improves scalability because now if two nodes want to transact with one another, they do not have to open up a new channel between them, which is time consuming and expensive. Instead, the two nodes can route funds through an existing path (consisting of multiple third-party channels and nodes) between them. However, these multi-hop payments introduce a potential exploit, the Flood and Loot Attack.

Note: at this point, it is important to make clear the distinction between on-chain transactions (transactions or requests posted to the blockchain) which are time-consuming, and off-chain transactions (not posted to the blockchain), which can happen very quickly and do not consume many resources.

2.2 HTLCs

HTLCs (Hashed Time-Lock Contract) are smart contracts used in the Lightning Network to ensure that nodes can claim payments that are made to them. When money is sent using an HTLC, the funds are locked until the receiver can provide a secret key passphrase, and the receiver confirm the transaction with the issuer of the HTLC. If the receiver enters the wrong secret key, or fails to do both of the aforementioned tasks without a set timeframe, the reciever loses access to the money locked inside the HTLC, and the issuer is free to claim the money. Every time a contract is issued, nodes set a broadcasting delta, which is the time at which an HTLC contract expires minus the time one individual in the channel is allowed to unilaterally close the channel (by going back to the blockchain)[2].

3 Flood and Loot Attack of Lightning Network

The following section describes a typical Flood and Loot Attack.

3.1 Setup and Wait

This is the first stage of the Flood and Loot Attack. To start the attack, the attacker first creates many channels from a source node to many victim nodes, and it locks funds into these channels. The attacker also prepares a target node by opening channels between the target node and many other legitimate nodes in the Lightning Network. This ensures that the target node has enough liquidity to take advantage of a victim. The final step of this stage is to wait for a vulnerable victim node.

3.2 Initiating Payments

This is the stage where the attack is launched. The source node sends multiple HTLC payments to a target node. The goal of the attacker is to maximize the amount of HTLC payments that are sent, as this will make it more difficult for the victim to fend off the Flood and Loot Attack.

3.3 Accepting Payments

Once the target node receives the HTLC payments, it accepts the HTLC and sends back the secret key which is used to gain access to the funds that are locked. After this, the target node no longer has open HTLC payments. The secret keys are passed from the target node to the victim nodes, and the victim nodes attempt to pass the secret key to the source nodes. At this stage, the source node does not response to the victim node, and the victim node is left with HTLC payments that must be claimed on the blockchain. Claiming a payment on the blockchain is a slow process so in many cases, victim nodes will not have time to claim the HTLC before the contract expires.

3.4 Collecting Expired HTLCs

As the HTLCs' timeout time approaches, the victims attempt to close the channels with the source node and claim all HTLCs on the Bitcoin blockchain. This means that they may have to publish many blockchain transactions all at once. Some of these transactions will timeout and thus will fail to be claimed by the rightful owner, which is the vicim node. The attacker follows up by claiming any HTLCs that remain past expiration, using the replace-by-fee policy, and raises the fee of his own transactions to be higher than that of the victims' transactions.

3.5 Is Flood and Loot Plausible?

A previous study [3] concluded that 95% of nodes in the Lightning Network accept a request to open a channel without knowing anything about the node sending the request. If the Flood and Loot Attack is successful even 15% of the time, this has catastrophic effects on the security of Bitcoin transactions. Our goal is to provide nodes in a network with a manner of determining whether a node is trustworthy or not. That way, nodes are way less likely to fall victim to attacks such as Flood and Loot.

4 Proposed Solutions to Flood and Loot in the Literature

The following are proposed solutions to mitigate the consequences of the Flood and Loot attack that can be found in the initial flood and loot paper [3].

4.1 Simplifying Incoming HTLCs

The idea is that if the preimage is sent back from the target adversary node, HTLCs in the channel are resolved immediately. Ultimately simplifying incoming HTLCs does not work because it introduces another attack, despite correcting flood and loot. So this solution does not work.

4.2 Increasing HTLC Fees Via Child Pays For Parent

This is also a bad mitigation strategy, as the only way to prevent or disincentive the attack would be to make the fees higher than the ones locked in the payment channel, which is impractical.

4.3 Reducing the Maximum Number of Unresolved HTLCs

The consequence of this strategy is that the attacker would need more victims in order to steal funds. This is not really an issue for the attacker though, because with the current state of the lightning network most nodes accept payment channel requests, they have no reason not to. Furthermore, this would also greatly slow down the network (more than our proposed solution) because of fewer channels allowed to be open all-around. Because everything is connected via multi-channel paths, this would affect everyone on the network.

4.4 Deciding When to Close Channels

We implement this as part of our solution, changing the broadcast delta for different channels with different reputation scores. Increasing the broadcast delta when faced with an adversarial peer seems to surpass any cons, as it does not slow down the transaction rate of the lightning network in any significant way, but rather just gives the attacked node enough time to claim their own money anyway.

4.5 Reputation Based Behavior

This is the core of our solution. See our algorithm in Section 5 and the trade offs in section 5.4. Our solution combines reputation based behavior with deciding when to close channels. We looked into solutions such as EigenTrust [4], but these global scoring schemes can only defeat malicious collectives when a group of pre-trusted peers are assigned at the start of the network. Since Bitcoin is already over a decade old, and creating a group of pre-trusted peers would be opposite of Bitcoin's democratic values, we ruled out such global solutions.

4.6 Anchor Outputs

Each of the nodes increases the fee of commitment with the child-pays-for-parent policy. However, this does not fix the main problem, which is that the money in the HTLCs is the victim's money, so the attacker probably doesn't care about using this money to pay for the higher fees. The attacker will still make money on each successful attack, and therefore is not properly disincentived.

5 Reputation-Based Scoring: Exhaustive Trust Algorithm (ETA)

Based on the mitigation strategies in Section 4, we have decided to combine reputation based behavior and varying channel closing times in our solution.

The overall idea of the Exhaustive Trust Algorithm is to award nodes reputation scores. These reputation scores represent a metric of the trustworthiness of a node in the Lightning Network. We define trustworthiness by how likely the node is to commit to transactions without backing out. This reputation score gives neighbors an idea of how well the specific node should be trusted. Over time, nodes can grow their scores by completing successful transactions without backing out. The actual numerical scores affect certain parameters of the smart contracts that determine how the network operates. The technical details will be further described in the following sections.

5.1 How ETA Works

ETA works as follows: For a user who is trying to make their i_{th} transaction in a given channel, they cannot transact more than the sum of the first through $i-1$ transactions. For the very first transaction, we start a channel off at a baseline of 1000 Satoshis. If ever one party is forced to return to the blockchain, the max a channel can transact is 1000 Satoshis again. There is a further limitation to the amount one can route: for multi-hop payment, the max a party anywhere in the chain can route is the lowest trust channel in the entire path from source to destination.

In practice, this means a user can only "double" the amount of money they have ever sent in the channel in one transaction. For example, a user who would like to transact many Satoshis at once would first transact 1000 satoshis, 1000 again, then 2000, 4000, 8000, and so on. Suppose a user does not max out the channel on their first transaction and transacts 500 instead of 1000 Satoshis.

Then, on their next transaction, the user is limited to transacting at max 500 Satoshis.

Note that ETA is a **pairwise** score. This has twofold benefit: Firstly, the complexity of implementation is far easier, since any individual node only needs to keep track of its score with its partners at any given time. A node can discover the lowest trust channel on its path from source to destination by simply tacking on ETA scores to the heartbeat messages a node sends when it is trying to discover the cheapest path through the network in vanilla lightning implementations. Secondly, the pairwise scheme defeats a serious problem that would stem from a global score: Attackers could spin up two nodes and transact directly between them honestly, building up his score and never losing any fees. The attacker could then leverage his high ETA score to run a lucrative flood and loot attack. Thus, we should avoid a global scoring scheme.

5.2 How ETA De-Incentivizes Attackers Even as Broadcast Delta Shrinks with Trust

We now show that ETA gives us a way to have low trust channels with high broadcast delta times, and we can afford to have high trust channels with low broadcast delta times. For channels with high trust, we want a low broadcast delta rule so the blockchain isn't unnecessarily flooded with requests, slowing down the network.

Attacker Profit = Funds from "Stolen" channels - Fees Paid to "Good" Channels

Stolen channels refer to those channels where the HTLC expired before the victim could claim it on the blockchain, while good channels refer to the ones where the victim was able to get their HTLC processed on the blockchain in time. For an attacker who establishes N channels, of which $x\%$ on average are stolen, we want

$$\text{(Fees paid per channel)} * (1-x) * (N) >= \text{(Money Routed through Channel)} * (x) * N.$$

On Lightning network, fees are a linear function of money routed through channel, usually just some percentage of the money routed through the channel. We can model fees paid per channel as

$$\text{Fees paid per channel} = \text{Fee Rate} * \text{Money Routed through Channel.}$$

Cancelling out N and Money Routed through Channel, we end up with

$$\textbf{Fee Rate * (1 - x)} >= \textbf{x}.$$

Note that this is for a single attack, however! Hypothetically, an attacker could attempt to keep transacting so they can increase the amount of money in a channel and then obtain a higher payoff. We will show this is to the decrement of the attacker because they burn so much money in fees through these valid transactions. In fact, we will show that as we transact more and more in any given channel, we can afford to shorten the broadcast delta while still keeping attacks de-incentivized.

We propose the following attacker: Attacker will open up **N** channels, transact honestly $i-1$ times, and on the i_{th} cycle, attacker will use flood and loot. By transacting many times previously, the attacker hopes they are growing the amount they can steal in any given channel. However, in reality, the attacker cannot profit from this scheme.

Let us define State 0 through State i as $S_0 .... S_i$. In each state, the attacker has the choice to continue making valid transactions, or attack. We will refer to their choices as "concede" and "attack". In S_0, we have shown that if fee rate **F** is greater than or equal to $x/(1-x)$, the attacker will get 0 from an attack. If the attacker chooses to concede, they will spend

$$(M_0) * (F) * N,$$

where M_0 is the initial amount they are allowed to transact with each victim.

In State 1, the max an attacker is allowed to transact is $M_1 = 2 * M_0$. If the attacker chooses to attack, they will gain

$$(M_1) * (x) * (N) - (M_1) * (1-x) * (N) * (F).$$

However, we must not forget that they spent some money in S_0. This means their actual profit is

$$(M_1) * (x) * (N) - (M_1) * (1-x) * (N) * (F) - (M_0) * (F) * N.$$

Let us define $S_{i,c}$ as the amount an attacker spends in state i when they concede; we define $S_{i,a}$ as the profit of an attacker should they choose to attack in State i. We see that in general, should an attacker choose to attack in S_i, their profit is

$$S_{i,a} = 2^i * M_0 * x * N - 2^i * M_0 * (1-x) * (N) * (F) - S_{i-1,c} - .... - S_{1,c} - S_{0,c}.$$

We want the attacker's profit to be zero. Simplifying the above equation, we get

$$S_{i,a} = 2^i * M_0 * x * N - 2^i * M_0 * (1-x) * (N) * (F) - (2^{i-1} - 1) * M_0 * F * N <= 0.$$

Solving for x, we get

$$x <= \frac{F * 2^i + F * (2^{i-1} - 1)}{F * 2^i + 2^i}.$$

This function is highly non-intuitive, but here we list some values for x (the threshold for attack success above which attacker profits) as i grows from 1 to 20 with a fee rate of 0.01.

1 0.009900990099009901

2	0.014301430143014302
3	0.0187018701870187
4	0.02359124801369026
5	0.029458501405696128
6	0.03684689456599981
7	0.04640834453815752
8	0.05896378389553634
9	0.07557559596837599
10	0.09763882957416303
11	0.12699924160987963
12	0.16610830247616878
13	0.21822828017699905
14	0.2877046259197358
15	0.3803284483378048
16	0.5038193414026211

If ETA uses 1000 Satoshis (currently 0.38 USD) as a baseline, 16 successful cycles already gets us to $.38 * 2^{16} = 24903.68$ USD. Thus, honest nodes very quickly reach a high transaction volume in their channels. It is unlikely a single party would ever need to transact such a large amount in one transaction. For $i > 11$, we see that current "vanilla" lightning implementations of broadcast delta suffice, since probability of theft just needs to be lower than 16%. For values $i <= 11$, we must learn what broadcast delta times correspond the sufficient attack success rate.

Let us now consider two cases:

1) An attacker has malicious intent from the start (aka they started with control of the source and destination node). If they are rational, they will never attempt the attack because it will result in zero profits or a loss on expectation.

2) Some honest node (they only control one node at the start) has many open channels, has transacted a lot, and therefore the broadcast delta has now shrunk in these channels. They suddenly decides to go rogue. BUT, importantly, they don't control a destination node like the attacker from Case 1. When they open up the destination node, they are limited on the amount they can transact on the destination side. Due to ETA working with multi-hop payments (we are bound by the LOWEST trust in the entire pathway), the attacker can't use this to their advantage.

5.3 A Learning Experiment to Optimize Mapping between ETA and Broadcast Delta

In the previous section, we have shown that for a channel with a given ETA, keeping the attack success rate below some threshold will render a single attack non-profitable. By adjusting the broadcast delta, we can alter the probability of a successful attack: A higher broadcast delta will result in lower attack success rate, and vice versa.

Attack success rate is not purely a function of broadcast delta, but also includes several other factors that are impossible to hold constant: namely, general congestion in the Bitcoin blockchain, which is a factor of fees, number of current ongoing transactions, and number of miners active. We suspect most of these factors are cyclical with respect to time of day (good place for a citation), so we propose running the following experiment multiple times, at different times of day.

The experiment has three groups:

1) **Control**: consists of 102 Lightning nodes running LND. Node 1 will open 100 channels and route payments to the last remaining node. These are honest transactions.

2) **Experimental Good**: consists of 102 Lightning nodes running a Lightning implementation with ETA inserted. Node 1 will open 100 channels and route payments to the last remaining node. These are honest transactions.

3) **Experimental Attack**: consists of 102 nodes running a Lightning implementation with ETA inserted, where node 1 and 2 are controlled by the attacker. Attacker will establish channels with nodes 3 through 102 and run a flood and loot attack.

In a single round of experiment, we will set the broadcast delta anywhere from 10 to 40 in Bitcoin block time for Groups 2 and 3, since these are the max and min values allowed by Lightning Network Daemon (LND), the most common implementation of Lightning. Then, we begin routing payments in all groups. After the completion of a single round, we modify the broadcast delta and proceed again. For the control group, we are interested in measuring latency - the time in Bitcoin blocks from when the payment is initiated to its close; we compare this against the latency of the Experiment Good group. For the experimental group, we are interested in measuring the probability of attack success (percentage of channels that had HTLCs expire) and the total amount of Satoshis stolen.

We will use the two metrics for learning as follows: For the Experimental Attack group, we will take the average attack success rates for a given broadcast delta time, and use this to set the broadcast delta rule for channels with given ETA scores. For the control and experimental group, we will use the difference in latency to validate whether ETA is efficient enough to keep Lightning transactions at an acceptable speed.

Unfortunately, we were unable to run this experiment ourselves due to lack of resources. A lightning node is required to be a bitcoin miner, which would require using the resources of 200 computers with GPUs. Purchasing compute time on AWS E2C GPU instances is an expensive proposition, and a tricky technical proposition to implement, especially given we ideated and proved ETA's usefulness with 2 weeks remaining in the project timeline.

5.4 Tradeoffs

We recognize that ETA is not a perfect solution. While the algorithm does mitigate the Flood and Loot Attack, it comes at a cost: lower speeds and more centralization. [6]

Introducing reputation score parameters that all nodes in the network must manage adds overhead to the network. Each node must keep track and update these scores for all of their neighbors, and when using multi-hop payments, the path with the highest reputation score must be calculated. In addition to these calculations, increasing the broadcast delta adds requests to the blockchain, clogging the Bitcoin network and therefore slowing it down. Since ETA also limits how much money can be sent through a channel, nodes may need to execute multiple transactions to send the actual amount of money they desire, broadcasting multiple transactions to the blockchain. The overhead of calculating scores and the increase of on-chain requests ultimately lead to greater latency in the Lightning and Bitcoin networks.

By adding a reputation score that makes some nodes more reputable than others, ETA may lead to centralization of the network [8]. This is because nodes with higher reputation scores can transact more funds and are included more often in multi-hop payments. The popular nodes will continue growing their reputation scores faster, and even more nodes will route through these nodes. In this case, the network uses these nodes discriminately. One way we could mitigate this is to have a logarithmic growth to reputation scores so that nodes do not keep accumulating reputation score as quickly. We can also implement an upper limit such that for multi-hop payments, paths with scores above the limit are treated equally, and a random path is chosen.

5.5 Impacts

For the ETA algorithm to be implemented, the Lightning Network infrastructure would need to be changed, and the reputation score feature would need to be added to the Lightning Network smart contracts. This would have to be done along with Lightning Network developers such as Lightning Labs [5]. For the users of the network, not much would apparently change. On the surface, the amount of money people could send would be limited by their reputation scores. Less apparently, the increased broadcast delta would potentially slow down the Bitcoin Network, negatively affecting people who want to transact on the network in a timely manner. Overall, ETA would not have a large impact on the average user of the Lightning Network or Bitcoin Network.

6 Conclusion and Future Works

Bitcoin and cryptocurrencies have recently gained traction among the public. The technology aims to be fast, secure, and scalable. While Bitcoin itself has had difficulty achieving speed and scalability, the added on Lightning Network provides a solution to these issues by enabling off-chain transactions. However, the network introduces potential for the Flood and Loot Attack, where adversaries can steal Bitcoin from victims on the Lightning Network. There has been some research into possible solutions, which we have discussed. We propose the Exhaustive Trust Algorithm, which uses reputation scores to determine how much money can be sent over a channel, which paths to take in multi-hop payments, and what broadcast deltas to use. These reputation scores are pairwise for all of the nodes in the Lightning Network, and nodes can grow their reputation scores by completing successful transactions. We enforce these rules by adding on to existing smart contracts of the Lightning Network. ETA makes the Flood and Loot Attack unprofitable for the adversary and thus practically mitigates the attack, although it leads to greater overhead, higher latency, and more centralization of the network. Future endeavors include an experimental simulation of an ETA implementation against the current vanilla Lightning Implementation as well as discussing ETA with Lightning Network developers.

References

[1] Lightning network website.

[2] Jake Frankenfield. Hashed timelock contract (htlc), Feb 2022.

[3] Jona Harris and Aviv Zohar. Flood & loot: A systemic attack on the lightning network. *CoRR*, abs/2006.08513, 2020.

[4] Sepandar D. Kamvar, Mario T. Schlosser, and Hector Garcia-Molina. The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th International Conference on World Wide Web*, WWW '03, page 640–651, New York, NY, USA, 2003. Association for Computing Machinery.

[5] Inc. Lightning Labs. Faster, cheaper, global layer two bitcoin.

[6] Stefano Martinazzi and Andrea Flori. The evolving topology of the lightning network: Centralization, efficiency, robustness, synchronization, and anonymity. *PLOS ONE*, 15(1):1–18, 01 2020.

[7] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments, 2016.

[8] Philipp Zabka, Klaus-Tycho Foerster, Stefan Schmid, and Christian Decker. A centrality analysis of the lightning network. *CoRR*, abs/2201.07746, 2022.