Decision Tree Based Inference of Lightning Network Client Implementations

Pol Espinasa-Vilarrasa[1], Sílvia Sanvicente[2], Cristina Pérez-Solà[1], and Jordi Herrera-Joancomartí[1]

[1] Universitat Autònoma de Barcelona,
{name.firstSurname}@uab.cat
[2] Independent researcher

Abstract. The Lightning Network (LN) is a second layer payment protocol on top of Bitcoin. It creates a peer-to-peer (P2P) network of payment channels that enable instant payments. The LN can be accessed through different implementations or clients, the most popular being Lightning Network Daemon (LND), Core Lightning Network (CLN), and Eclair. The first step in many known attacks to the LN is to infer the software client the node is running. This paper presents two classification models based on decision trees to infer the implementation of LN clients from either the traffic of the gossip protocol or the announced BOLT #9 features, offering a cost-free means of identification. The accuracy presented by both models in our experiments is high, ranging from 87% to 100% depending on the model and the environment where it is deployed. The application of our inference models on the LN shows a prevalence of LND clients.

Keywords: Bitcoin, Blockchain, Lightning Network (LN), Machine Learning

1 Introduction

The Lightning Network (LN) [1,2] is a layer 2 payment scheme deployed on top of Bitcoin [3]. Its main goal is to address the scalability problems of Bitcoin, allowing for fast transactions outside the blockchain and lower fees. Additionally, by moving transactions out of the blockchain, better privacy properties are provided. The LN operates on a peer-to-peer (P2P) network, where LN nodes exchange the necessary data to create and manage payment channels, over which payments to other nodes can be routed.

Nodes within the LN can operate using various software alternatives. The most popular implementations, namely Lightning Network Daemon (LND) [4], Core Lightning Network (CLN) (former c-lightning) [5], and Eclair [6], come with distinct default parameters and methods for routing payments. Determining the specific LN client running on a node is an area of interest for several reasons. Firstly, previous works have shown that inferring the LN client is a first key step for different attacks on the network. In turn, inferring clients is useful in analysing the resilience of the network to those attacks. Secondly, decentralization in P2P networks often considers the diversity of client implementations [7], thus this information is needed to evaluate decentralization.

In this context, the contribution of this work is twofold. On the one hand, we present two robust classification models to infer LN client implementation without relying on

default-based heuristics. These models address two distinct scenarios: one which is based on network traffic and the other which is based on feature flags announced by the nodes themselves. One the other hand, we apply these models on the live Lightning Network and report the distribution of nodes for each implementation.

The rest of the paper is organized as follows. First, Section 2 introduces the necessary background and Section 3 presents the state of the art, explaining attacks that require client identification and previous methods used to do so. After that, Section 4 describes the threat model and the two inference scenarios. Then, the experimental part of this work is presented: first, Section 5 describes the experimental setup; secondly, Section 6 evaluates the models; and finally, Section 7 presents the results of applying the models to obtain a view of the current distribution of nodes in the LN. Lastly, Section 8 presents the conclusions and lines for future work.

2 Lightning Network Background

The Lightning Network is a layer 2 protocol on top of the Bitcoin blockchain. It was created as a solution to the Bitcoin scalability problem and provides four main benefits: better transaction throughput, fast payments, improved privacy, and lower fees.

The LN is a network of bidirectional payment channels between pairs of nodes. A channel is opened by sending a transaction (called the *funding transaction*) to the blockchain, depositing a certain amount of coins (the capacity of the channel). Once a channel is opened, the two participants of the channel can pay to each other in both directions without needing to send transactions to the blockchain. These payments are made by *commitment transactions* that update the balance each party has in the channel. As long as the peers behave correctly, commitment transactions will not be sent to the blockchain and will only reflect balances. However, if necessary, these can be sent to the blockchain to retrieve funds from the channel or even penalize fraudulent behavior. The channel can be closed by sending another transaction (called the *closing transaction*) to the blockchain, which reflects the final state of the channel.

Payment channels are established between pairs of nodes, allowing payments exclusively between the two nodes. The strength of the LN lies in the ability to perform multihop payments, enabling third parties to route payments between other parties' channels.

Nodes in the LN relay on a P2P network to communicate and operate the channels. This network is mainly used for channel management (opening, updating and closing channels) and routing information propagation (discovering nodes and channels).

2.1 Gossip protocol

One of the main features of the LN is the ability to create multihop payments that use third party channels to make a payment between two users that do not directly share a channel. For nodes to make such payments, they need to be able to find routes in the LN channel graph. The gossip protocol is used for nodes in the LN to share information about the network topology, that is, discover other nodes and existing channels, and create their own view of the LN channel graph.

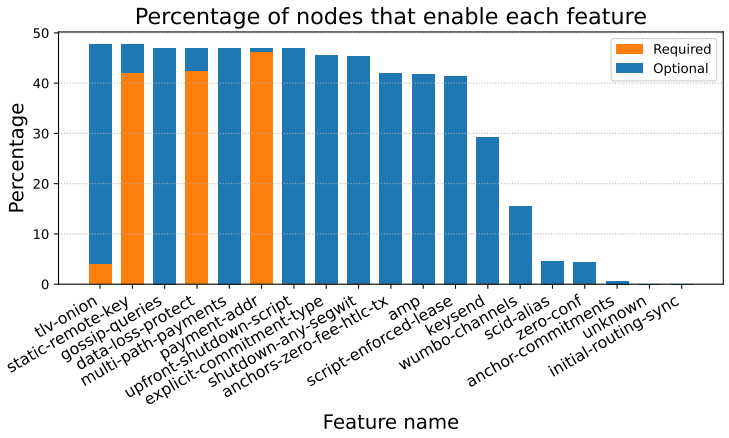


Fig. 1: Features announced by LN nodes (sample obtained by the authors on September 2023)

Gossip protocols are commonly used in P2P networks to disseminate information through the whole network using the direct connections that peers have with each other. The LN gossip protocol supports three different messages: node announcements, channel announcements, and channel updates [8].

Peers can join the LN P2P network and receive gossip messages from their direct peers without having to open any LN channel. Therefore, listening to gossip messages and using their data to construct a local channel graph can be done by any party for free (i.e. without having to pay transaction fees).

2.2 Feature flags

The BOLT[3] #9 - Assigned Feature Flags [9] describes how to identify features within the LN based on flags. Flags are used to show what features a node supports, either as optional or compulsory. Channel and node announcement messages (among others) contain a feature field to describe the features nodes support and require. Figure 1 shows the percentage of nodes announcing each of the features (data from September 2023).

2.3 LN clients

Lightning Network clients are the actors of the LN that open and close channels for sending bitcoin transactions outside the blockchain. In order to coordinate with each other they exchange data using the P2P network and the gossip protocol.

[3] The BOLTs (Basis Of Lightning Technology) are specifications for the LN, used to ensure that different Lightning Network node implementations can interact seamlessly.

Unlike on-chain clients, where nodes are inspired by the same original implementation (Bitcoin Core), the LN has a wider variety of implementations. The most noteworthy are LND [4], CLN [5], and Eclair [6], each one written in a different programming language.

As they do not follow an original implementation, they implement the protocol that is specified in the different BOLTs in order to ensure interoperability.

3 State of the art

Previous works have focused on designing attacks and resilience analysis that needed to infer LN client implementations. Examples of such attacks are the balance discovery attack [10], the lockdown attack [11], the congestion attack [12], the flood and loot attack [13], or route hijacking and DoS attacks [14]. Additionally, other papers that do not focus on security but on describing the network, also use client identification techniques [15].

In any case, these papers tried to infer the client implementation based on the parameters announced in the nodes' policies, assuming nodes were using default values. Inference was quite accurate because different implementations had different default values. However, as new versions of the clients appear with modified defaults, some start to collude and inference losses accuracy.

In this paper, we present two different methods to infer LN client implementations without resorting to manually identify default configurations in the nodes' policies. Unlike previous works, our models are robust to nodes that change the default policy values of their implementation (e.g. `cltv_expiry_delta` or `htlc_minimum_msat`).

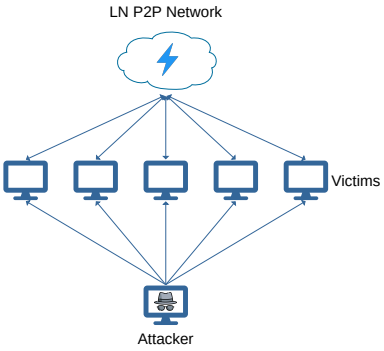
4 LN client inference scenarios

In this article, two different techniques to discover the Lightning Network node implementation are presented. In both scenarios, the attacker (who wants to infer the client used by a target node) only needs to run a single node connected to the LN P2P network (no channels have to be opened by the attacker), so from an economic cost point of view both techniques are free (i.e. no fees need to be paid nor there is any need to own BTC).

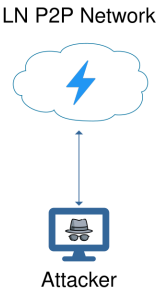
4.1 Scenario 1: traffic based inference

The first scenario bases the classification of the nodes on features extracted from the network traffic between the attacker and the target. Therefore, the attacker must be able to obtain LN P2P network packets from the target. This can be accomplished either by sniffing traffic from a target's node already existing connection, or opening a direct LN P2P connection with the target. In the experimental part of this paper, we opt for the second option (Figure 2a). However, given the extracted features, both settings are equivalent. Note that the attack can be carried out simultaneously for many nodes (by opening connections or capturing traffic from all of them).

The LN P2P traffic is encrypted between the two peers, but as the attacker only uses traffic metadata, the content of the packets is not relevant.



(a) Traffic based scenario.



(b) BOLT #9 features based scenario.

Fig. 2: Topology diagrams for the two scenarios. Note that, for clarity purposes, the representation places the attacker and targets outside the P2P network cloud. Nevertheless, technically speaking, both the attacker and targets exist within this network.

Six features are derived from the traffic between the attacker and the target node, which correspond to the minimum, maximum, and mean packet length in each direction of the connection.

4.2 Scenario 2: BOLT #9 features based inference

The second scenario bases the classification model on the BOLT #9 features that the target node has enabled. These features are public information that nodes share.

Using this information the strategy becomes even easier than the previous case, as now it is not necessary to connect directly with the target node. As shown in the Figure 2b, only a node connected to the LN P2P network is needed by the attacker, and this is enough to infer any node publicly announcing their feature flags.

The underlying assumption of this model is that, as not all features are mandatory and many are still under development, different implementations will announce different sets of features.

5 Experimental setup

Bitcoin clients can be deployed on different networks (and blockchains). The *real* Bitcoin network (in which the coins have actual value in the market) is called the *mainnet*. However, developers often need to test protocols before deploying them in production environments. A *regtest* network is a private Bitcoin network, that replicates the behaviour of the real network, but allows developers to control the nodes that join the network and the mining process. *Regtest* networks thus allow to execute the Bitcoin protocol in a controlled setting.

Our experiments are conducted using two different environments: one based on a *regtest* network and the other on the *mainnet*.

5.1 Regtest network environment

Our *regtest* network environment has been deployed using the Polar [16] software.

Nodes are deployed using docker containers, which facilitates deploying nodes executing different client implementations and versions.

The ground of truth for the *regtest* network environment can be thus directly obtained, since we are in control of all the nodes joining the network and know which implementations they are running.

The **traffic based inference (scenario 1)** requires the attacker to capture LN traffic between the attacker and the target node. In the *regtest* environment, both the attacker and the target are docker containers running in the same physical machine. The traffic sniffing software Wireshark with a lightning-dissector plugin is then used to capture and analyse network traffic between the attacker and the target during 30 minutes.

The scenario is composed by one attacker node using LND and fifteen other nodes divided in three groups of five, each group running one different implementation; LND, Eclair, and CLN.

During the experiments, in order to simulate the real LN, we have emulated some actions such as opening and closing channels or sending payments.

Extracting the data for the **BOLT #9 features based inference (scenario 2)** is easier than the first technique, as we do not need to sniff network traffic using third party programs, nor directly connect with the target node. Only running one LN node is enough to extract the needed information. After connecting to the Lightning Network, the LN channel graph from that node's point of view is obtained. This graph contains information about each node's BOLT #9 features as announced by the nodes themselves. This scenario is composed by one attacker node using the LND implementation and two other nodes, each using a different implementation (CLN and Eclair).

5.2 Mainnet network environment

Our *mainnet* network environment uses the standard Lightning Network environment, running on top of the real Bitcoin *mainnet*. Our environment is composed by a full Bitcoin node and a LN node running a LND client, which acts as the attacker.

To obtain labeled *mainnet* nodes to train and validate our models, we rely on public information provided by the node operators themselves. Particularly, we have used information from the "LN Search and Analysis Engine" [17], where public nodes are listed with their public keys, IP addresses, and other public attributes, and node operators may provide further information about their nodes (such as the software implementation).[4]

With the public key, IP address, and port, our attacker node can connect to those nodes using standard CLI commands and collect data in the same way we did in the *regtest* environment, with Wireshark and the lightning-dissector plugin.

[4] Although these nodes currently share this information voluntarily, the data collected in this project in *mainnet* network environment will not be published, to prevent this critical information from being perpetuated over time.

For the **traffic based inference (scenario 1)**, our node has connected to 81 different nodes[5] (68 of which claim to use the LND, 3 Eclair, and 10 CLN), and has captured traffic from that connection for around one hour.

Data for the **BOLT #9 features based inference (scenario 2)** has been collected by looking at the LN channel graph from the point of view of our LND attacker (again, we only needed the attacker to be connected to the LN P2P network, and no direct connections with the target node where required).

5.3 Model training and testing

In order to determine our classification algorithm, two main characteristics have been taken into account. First, the algorithm must be explainable, that is, its decisions should be easy to interpret by humans. This allows us to understand what properties of the LN protocol are being used to classify, and would facilitate future design of countermeasures. Second, since large datasets are not straight forward to obtain (particularly, for the *mainnet* environment we are limited by the current size of the LN network), we avoid algorithms that require a large training set. Given these characteristics, we select a decision tree algorithm to tackle our classification problem.

A decision tree consists of a set of internal nodes (decision nodes) and leaf nodes. The internal nodes are associated with one of the features and have two branches, each one representing the possible values that the associated feature can take. Leaf nodes are labelled with a class and represent the final output of a series of decisions.

We use k-fold to validate the models. On both scenarios, *mainnet* and *regtest*, we use k-fold with $k = 5$ over the network traffic and BOLT #9 features datasets built from the interactions with the labeled nodes.

Then, the performance of a model is evaluated with a confusion matrix [18], a two-dimensional matrix that compares the labels of the predicted classes with the true labels. Each column of the matrix represents the number of predictions for each class, while each row represents the instances in the actual class.

6 Evaluation of the models

This section presents the analysis conducted on the two models across the two scenarios. Its purpose is to assess the models' accuracy, examine the features used for classification, and determine potential correlations between the *regtest* and *mainnet* scenarios.

6.1 Regtest network environment

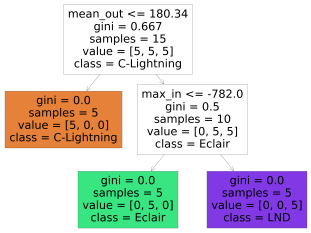
In the following two subsections we present the results over the *regtest* environment: the training and evaluation of the traffic based model, and the analysis of the BOLT #9 features that allow us to differentiate between the three LN node implementations.

[5] This corresponds to all nodes that inform their implementation and accepted our incoming P2P connections.

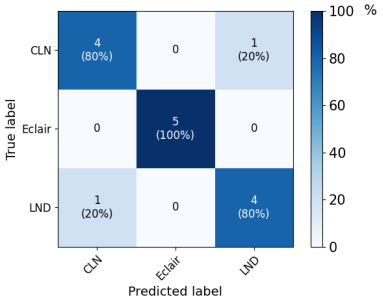
Scenario 1: traffic based inference The *regtest* scenario based on the network traffic has been used to test the model and confirm the viability of differentiating the different implementations based on the P2P network packets.

As mentioned before, we run 5-fold on the traffic captures of the 15 nodes from the environment. The decision tree generated by the model (Figure 3a) only uses two features to differentiate the three classes: first, the mean of the size of the outgoing packets is used to separate CLN nodes from the rest; then, the maximum size of the incoming packets separates Eclair and LND nodes.

We achieved an average accuracy of 86.68% (three executions obtained a 100% accuracy and the other two got a 67%). The confusion matrix in Figure 3b displays the outcomes of these executions, showing that the model misclassified only two nodes.



(a) Decision tree of the sniffer based model.



(b) Confusion matrix of the sniffer based model.

Fig. 3: Traffic based inference over the *regtest* environment.

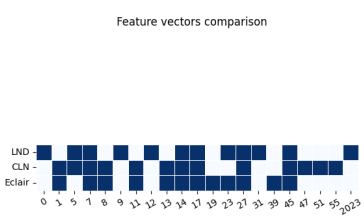
Scenario 2: BOLT #9 features based inference The *regtest* scenario based on the features enabled on each implementation has been used to know and compare the features on each node with the certainty of knowing what implementation they are using (without having to rely on third parties). Therefore, instead of training a decision tree, we analyse the features exhibited by the studied implementations.

Figure 4a displays the features enabled on each implementation. As it can be seen, LND can be easily identified if features 0, 9, 12, 31, or 2023 are enabled.

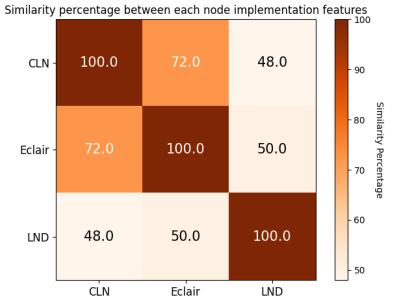
Similarly, the presence of feature 39 distinguishes Eclair, while features 47, 51, and 55 serve as identifiers for CLN.[6] The fact that there are unique features on different implementations seems to indicate that it would be indeed possible to create a decision tree to classify nodes from these features.

[6] These features correspond to data-loss-protect (0), tlv-onion (9), static-remote-key (12), amp (31), and script-enforced-lease (2023), for LND; unknown (39), for Eclair; and scid-alias (47), zero-conf (51), and keysend (55), for CLN .

Figure 4b shows the similarity[7] of the BOLT #9 features configuration between the different nodes implementations. We can see that LND should be the easiest to identify due that the similarity between LND and the other implementations is the lowest one, while Eclair and CLN have a higher similarity.



(a) Feature vector for each implementation.



(b) Similarity between the features that each implementation has enabled.

Fig. 4: Comparison of the features enabled on each LN node implementation.

6.2 Mainnet network environment

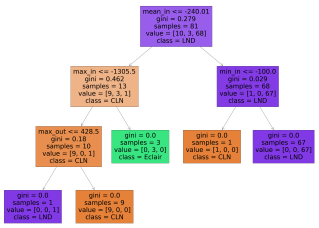
After the favorable results achieved on the *regtest* scenario, we build the two models on the *mainnet* environment. In the following two subsections, the results of these executions are analyzed and compared to the *regtest* models.

Scenario 1: traffic based inference The complexity of the decision tree has increased in comparison with the *regtest* environment, as shown in Figure 5a. The root of the tree divides nodes in two groups with respect to the the mean incoming packet size. Then, maximum packet sizes are used over the first group to differentiate the three classes; and the minimum size of incoming packets is used on the second group to divide CLN and LND nodes (there are no Eclair nodes on this second group).

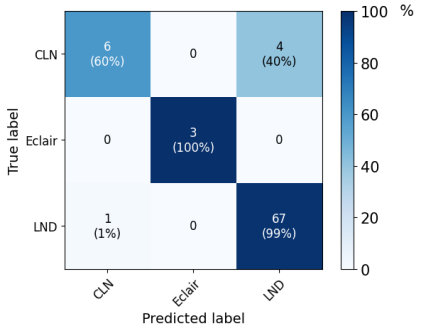
We achieved an average accuracy of 94%. The confusion matrix in Figure 5b displays the outcomes of these executions, showing that the model misclassifies some CLN and LND nodes by confusing them between these two implementations, which is the same kind of error shown in the *regtest* environment.

Scenario 2: BOLT #9 features based inference Figure 6a shows the decision tree the classification model has built. It only uses two features (flags 0 and 5) to infer the implementation.

[7] The similarity is calculated based on the intersection of two lists with the attributes of each implementation.



(a) Decision tree of the sniffer based model.

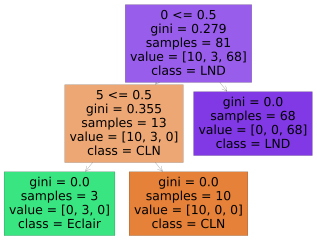


(b) Confusion matrix of the sniffer based model.

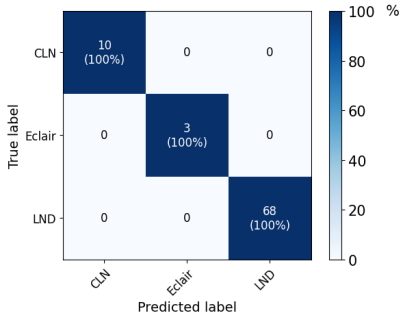
Fig. 5: Traffic based inference over the *mainnet* environment.

This selection of features is aligned with our previous analysis over the *regtest* environment (recall Figure 4a), since feature 0 seems to appear only on LND and feature 5 allows to differentiate between CLN and Eclair.

This model achieves an average accuracy of 100% (Figure 6b depicts the execution results).



(a) Decision tree.



(b) Confusion matrix.

Fig. 6: Features based inference over the *mainnet* environment.

In order to further explore whether *regtest* behaviors can be extrapolated to the *mainnet* environment, we also compared the similarity between the features used by each node in the *mainnet* training dataset with the reference feature vector extracted from *regtest*. The results of this comparison can be seen in Table 1, where we can see that the similarity percentage is quite high. The lowest values are around 75% for CLN and LND, but the mode for these is around 92-93%, so we can assume that the lowest values are from a small group of nodes that are running some old or modified version.

Implementation	Mean	Mode	Median	Range
LND	93.53%	92%	92%	74-100%
CLN	88.25%	93%	90%	75-96%
Eclair	94.67%	96%	94%	92-96%

Table 1: Similarity metrics between *mainnet* and *regtest* nodes

7 Inferring the distribution of LN client implementations

After training and evaluating the models for the two inference scenarios in the *mainnet*, we used them to infer the LN client implementation of all nodes we were able to connect to on the real LN. Since we do not have a ground of truth for this dataset, we cross-checked the predictions of the two models to evaluate their accuracy. The coincidence between both models is 97.63% (288 out of 295 nodes).

Our inference models indicate a prevalence of LND with 85% of the clients, followed by a 15% of CLN, and less than 0.5% of Eclair. Specifically, the traffic based model identifies 250 LND, 42 CLN, and 1 Eclair clients, while the BOLT #9 feature flags identifies 252 LND, 42 CLN, and 1 Eclair.

8 Conclusions and further research

This paper has presented two different techniques to infer the LN client implementation. The two techniques assume different attacker's abilities: one stronger attacker who is able to connect to the LN client of the target node or sniff its already existing connections; and one weaker attacker which only needs to be able to connect to the Lightning Network.

In both cases, the success rates of the inference are high: the traffic based model obtains an accuracy of 86.68% on the *regtest* network and 94% on the *mainnet* network; the BOLT #9 features based model achieves 100% accuracy.

With these results, we have shown that it is possible to infer the implementation of a LN node by observing only the encrypted network traffic or public node information like BOLT #9 features.

Even though the BOLT #9 features model based has a 100% accuracy, using both models in parallel may be useful to validate the results and enhancing resilience to potential future changes in client implementations.

The application of the obtained models to the live Lightning Network has resulted in a client distribution comprising 85% of nodes for LND, 14% for CLN, and less than 0.5% for Eclair. This aligns with previous research works that were based on default configuration parameters [15].

Our work has focused on inferring the LN client nodes are running. Future work could be done to further refine the classifiers in order to distinguish not only the client but also the specific version it is running. Another improvement would be to include additional features like transaction fingerprints, that could be used to classify nodes depending on the funding and closing transactions from Lightning Network channels.

Acknowledgements

This work has been partially supported by the Spanish ministry under grants PID2021-125962OB-C33 SECURING/NET and PID2021-125962OB-C31 SECURING/CYBER; the "Plan de Recuperación, Transformación y Resiliencia" funded by the European Union - NextGenerationEU under the project DANGER C062/23; and the AGAUR grants SGR2021-00643 and SGR2021-01508.

References

1. J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2016.
2. A. Samokhvalov, J. Poon, and O. Osuntokun, "Lightning network in-progress specifications."
3. S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system Bitcoin: A Peer-to-Peer Electronic Cash System," https://bitcoin.org/en/bitcoin-paper, 2009.
4. L. Labs, "lnd: Lightning Network Daemon," https://github.com/lightningnetwork/lnd, 2024, last access: 2024-05-16.
5. ElementsProject, "c-lightning: A specification compliant Lightning Network implementation in C," https://github.com/ElementsProject/lightning, 2024, last access: 2024-05-16.
6. ACINQ, "Eclair: A scala implementation of the Lightning Network," https://github.com/ACINQ/eclair, 2024, last access: 2024-05-16.
7. M. Cortes-Goicoechea, L. Franceschini, and L. Bautista-Gomez, "Resource analysis of ethereum 2.0 clients," in *2021 3rd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*. IEEE, 2021, pp. 1–8.
8. L. Network, "BOLT #7: P2p node and channel discovery," https://github.com/lightning/bolts/blob/master/07-routing-gossip.md, 2024, last access: 2024-05-16.
9. Lightning Network, "BOLT #9: Assigned feature flags," https://github.com/lightning/bolts/blob/master/09-features.md, 2024, last access: 2024-05-16.
10. J. Herrera-Joancomartí, G. Navarro-Arribas, A. Ranchal, C. Pérez-Solà, and J. Garcia-Alfaro, "On the difficulty of hiding the balance of lightning network channels," in *Proc. of the ACM Asia Conference on Computer and Communications Security*, 2019, pp. 602–612.
11. C. Pérez-Sola, A. Ranchal, J. Herrera-Joancomartí, G. Navarro-Arribas, and J. Garcia-Alfaro, "Lockdown: Balance availability attack against lightning network channels," in *Int. Conf. on Financial Cryptography and Data Security*. Springer, 2020, pp. 245–263.
12. A. Mizrahi and A. Zohar, "Congestion attacks in payment channel networks," in *Int. Conf. on Financial Cryptography and Data Security*. Springer, 2021, pp. 170–188.
13. J. Harris and A. Zohar, "Flood & loot: A systemic attack on the lightning network," in *Pro. of the 2nd ACM Conference on Advances in Financial Technologies*, 2020, pp. 202–213.
14. S. Tochner, A. Zohar, and S. Schmid, "Route hijacking and dos in off-chain networks," in *Proc. of the 2nd ACM Conf. on Advances in Financial Technologies*, 2020, pp. 228–240.
15. P. Zabka, K.-T. Foerster, S. Schmid, and C. Decker, "Empirical evaluation of nodes and channels of the LN," *Pervasive and Mobile Computing*, vol. 83, p. 101584, 2022.
16. Polar, "Regtest Lightning Networks, Made Easy." https://lightningpolar.com/, 2022, last access: 2024-05-16.
17. 1ML, "Lightning Network Search and Analysis Engine," https://1ml.com/, 2022, last access: 2024-05-16.
18. X. Deng, Q. Liu, Y. Deng, and S. Mahadevan, "An improved method to construct basic probability assignment based on the confusion matrix for classification problem," *Information Sciences*, vol. 340, pp. 250–261, 2016.