

Ca' Foscari
University
of Venice

Master Degree
in Engineering Physics

Final Thesis

A Complex Systems Analysis of the Lightning Network for Resilience and Revenue Optimization

Supervisor
Prof. Peppino Fazio

Graduand
Damiano Bacchin
Matriculation Number 883998

Academic Year
2025 - 2026

Contents

1 Introduction **4**
- 1.1 Overview of this work . . . 4
- 1.2 Thesis structure . . . 5
- 1.3 Acknowledgements . . . 5
- 1.4 Declaration on the use of Artificial Intelligence (AI) tools . . . 6

2 Bitcoin fundamentals **7**
- 2.1 The Bitcoin blockchain . . . 8
  - 2.1.1 Keys, addresses and wallets . . . 8
  - 2.1.2 Transactions . . . 14
  - 2.1.3 Blocks and mining . . . 20
- 2.2 The Blockchain Trilemma . . . 25
- 2.3 Scalability solutions . . . 26

3 Lightning Network **30**
- 3.1 Payment Channels . . . 31
  - 3.1.1 Sending payments . . . 32
  - 3.1.2 Routing payments . . . 34
- 3.2 The Gossip protocol . . . 35
  - 3.2.1 Procedure . . . 35
  - 3.2.2 Messages types . . . 36

4 Data collection and network analysis **38**
- 4.1 Data collection . . . 38
  - 4.1.1 Introduction . . . 38
  - 4.1.2 Installing a Lightning Node . . . 38
  - 4.1.3 Running a Bitcoin full node . . . 43
  - 4.1.4 Data structure . . . 45
- 4.2 Graph Analysis . . . 46
  - 4.2.1 Mathematical Model . . . 46
  - 4.2.2 Metrics and statistics . . . 47

4.2.3 Graph visualization . . . 57
4.2.4 Clustering and communities . . . 58

5 Simulations and optimization **61**
5.1 Robustness analysis . . . 61
5.1.1 Graph generation . . . 63
5.1.2 Robustness optimization . . . 66
5.2 Throughput optimization . . . 69
5.2.1 Introduction . . . 69
5.2.2 Rebalancing strategies . . . 69
5.2.3 Generating payments data . . . 75
5.2.4 Simulation results . . . 76
5.3 Related work . . . 79

6 Routing node economics **81**
6.1 Taxonomy of channel rebalancing costs . . . 81
6.1.1 On-Chain Transaction Fees . . . 82
6.1.2 Off-Chain Lightning Routing Fees . . . 83
6.1.3 Third-Party Service Provider Fees . . . 84
6.1.4 Opportunity Costs . . . 84
6.2 Operational cost breakdown by transaction type . . . 84
6.2.1 Channel opening: the funding transaction . . . 84
6.2.2 Channel closure mechanisms: cooperative vs. unilateral . . . 85
6.2.3 Submarine Swaps . . . 85
6.2.4 Channel Splicing . . . 87
6.2.5 Circular rebalancing . . . 88
6.3 Breakeven analysis . . . 89

7 Conclusion **90**

Chapter 1

Introduction

1.1 Overview of this work

Bitcoin is, to date, the most relevant example of a fully operational decentralized monetary system. Its security and censorship resistance stem from the adoption of the *Proof-of-Work* consensus mechanism, which enables a robust decentralization of the validation process. However, the base protocol exhibits well-known scalability limitations, which reduce its suitability as a global payment infrastructure and, in particular, for small-value transactions. These limitations are a consequence of structural design choices, including the on-chain handling of finality and transaction capacity.

Over the years, multiple solutions have been proposed to improve scalability, based on different approaches. Among them, the implementation that has achieved the greatest traction in terms of adoption is the Lightning Network, a second-layer protocol built on top of the Bitcoin blockchain, which uses payment channels to enable a large number of off-chain payments with reduced latency and limited costs.

This thesis investigates the topology of the Lightning Network as a complex system using real-world network graph snapshots, evaluating its structural properties, robustness, and resilience against various attack and failure scenarios. Experiments demonstrate that a broader distribution of payment channels, counteracting the network's inherent tendency toward centralization for efficiency, significantly enhances resilience, even in the event of failures among highly influential nodes.

Furthermore, the thesis addresses the critical challenge of channel rebalancing, essential for optimizing network efficiency and overall transaction throughput. To this end, the main rebalancing strategies are described and compared, both theoretically and through simulations. The analysis includes proactive and passive techniques, considers the interaction between on-chain and off-chain operations,

and evaluates the effectiveness of the methods also in relation to the economic costs associated with rebalancing. In addition, it investigates the impact of cooperative approaches among adjacent nodes in the rebalancing process.

1.2 Thesis structure

The thesis is structured into the following sections. The first, introductory part presents the fundamental concepts underlying the Bitcoin blockchain. Although this protocol is not the primary focus of the thesis, it provides the infrastructure on which the Lightning Network is built; consequently, several elements are essential to understanding the mechanisms used to rebalance payment channels.

A theoretical section follows, devoted to the operation of the Lightning Network, describing the operating principles of payment channels and the processes used to route transactions. The subsequent chapters adopt a more analytical approach and are based on simulations conducted on real-world data: after presenting the methodology used to reconstruct the dataset, the analysis examines the network's characteristic parameters and metrics, referring to a snapshot at a specific point in time.

The final sections investigate, through analysis and simulation, two aspects that are crucial for the system's correct operation: network robustness and the efficiency of payment-routing mechanisms.

In conclusion, a detailed cost breakdown analysis of the various rebalancing techniques was conducted, focusing on the breakeven threshold for node operators to highlight the economic viability and sustainability of the compared rebalancing strategies.

1.3 Acknowledgements

This thesis aims to transparently describe the algorithms and strategies employed in the analysis and simulation activities, with the goal of ensuring the replicability and verifiability of the results.

The Python code used for analysis and simulation is freely available and can be downloaded from the following public repository:

`https://github.com/damianobacchin/lightning-network-simulator.git`.

With regard to the data for the analyzed network, a snapshot of the network was acquired on March, 30 2026. The collection procedure is documented in the dedicated chapter so as to enable its reproduction. It should be noted that the Lightning Network can be modeled as a dynamic graph that continuously evolves:

new nodes join it every day, others cease operating, and channels are constantly opened, closed, and updated. Consequently, downloading the graph on different dates may lead to variations, albeit limited, in the results obtained.

Throughout the thesis, particularly in the introductory section providing the theoretical framework and in the section devoted to the analysis of network metrics, figures were included to represent the temporal evolution of several quantities. Since such historical series cannot, in many cases, be reconstructed ex post, figures provided by Glassnode (`https://glassnode.com/`), a platform for analyzing and visualizing data related to blockchains and digital assets, were also used. Some visualizations are available in the free plan, whereas others require a subscription. Glassnode is gratefully acknowledged for making the figures required for this thesis available free of charge.

1.4 Declaration on the use of Artificial Intelligence (AI) tools

I, the undersigned Damiano Bacchin, author of this Master's thesis, hereby declare that I have utilized generative artificial intelligence technologies to support various aspects of the research and writing process, specifically:

- Gemini (model 3.6): Accessed via the institutional account provided by the University, utilized for translation assistance, textual refinement, and stylistic revision.
- Claude Code (model Opus 4.8): Utilized for reviewing and optimizing the Python source code developed for the simulation framework.

I further declare that all AI-assisted outputs were thoroughly verified, critically evaluated, and revised. I assume full personal responsibility for the overall quality, factual accuracy, and scientific integrity of this work.

Chapter 2

Bitcoin fundamentals

Bitcoin represents the first successful implementation of a decentralized digital currency based on blockchain technology, remaining its most prominent application to date. While preceded by several attempts to establish distributed payment systems, Bitcoin was formally introduced in 2008 through the whitepaper *Bitcoin: A Peer-to-Peer Electronic Cash System* [21]. The document proposed a framework for digital transactions that bypasses financial intermediaries, eliminating the need for a central coordinating authority.

Although originally conceived as an electronic cash system, Bitcoin's evolution has led the market to categorize it primarily as "digital gold" rather than a medium of exchange. This shift is driven by its properties as a store of value and its algorithmic scarcity, which mirrors the extraction dynamics of precious metals. This classification is further reinforced by the technical characteristics of the base layer:

- Latency: A transaction typically takes 10 minutes to be included in a block. For high-assurance finality, a standard waiting period of six confirmations (approximately one hour) is generally required.
- Fee Structure: Transaction fees are decoupled from the transacted value; instead, they depend on the data size (in bytes) recorded on the distributed ledger. Consequently, small-value transfers may become uneconomical if the required fees exceed the transaction amount.
- Throughput Constraints: Due to limited block space, the network faces a physical threshold regarding scalability, averaging approximately 7 transactions per second (TPS).

These constraints are a direct consequence of prioritizing security and decentralization over throughput, a challenge often referred to as the *Scalability*

Trilemma [9]. To address these limitations without compromising the main layer's integrity, the Lightning Network was introduced. As a Layer 2 protocol built atop the Bitcoin blockchain, it enables off-chain processing of transaction volumes orders of magnitude higher than the base layer, effectively restoring Bitcoin's utility as a viable, high-frequency payment system.

This introductory chapter outlines the general operational mechanisms of the Bitcoin blockchain, which serves as the foundational layer for the Lightning Network examined in this work. A more detailed and comprehensive treatment of the concepts presented in this chapter can be found in the book *Mastering Bitcoin*. [4]

2.1 The Bitcoin blockchain

The underlying technology of Bitcoin is the blockchain, a type of *Distributed Ledger Technology* (DLT) that enables consensus within a *peer-to-peer* network.

The infrastructure comprises a distributed network of nodes that propagate transaction data and maintain a comprehensive chronological record of all past transactions. This ledger is organized into a sequence of blocks, immutably linked through cryptographic primitives. The inclusion of new transactions is governed by a distributed consensus mechanism, which in Bitcoin is implemented as *Proof-of-Work* (PoW). Furthermore, PoW regulates the issuance of new currency according to a controlled inflationary model that decays exponentially over time.

Network participants operate pseudonymously, managing the monetary unit (bitcoin) via digital wallets. These tools leverage public-key cryptography and digital signatures to ensure the security and ownership of assets.

The following sections provide a detailed technical analysis of the architectural components that enable these mechanisms.

2.1.1 Keys, addresses and wallets

Addresses are the primary identifiers used to interact with the Bitcoin network and, in particular, to receive funds.

Transaction security relies on asymmetric (public-key) cryptography: in this context, the cryptographic component is not intended to encrypt information, but rather to guarantee the authenticity, integrity, and non-repudiation of transactions.

Asymmetric cryptography is based on pairs of mathematically correlated keys: a private key and a public key.

The private key is known exclusively to the user and is typically stored in a software medium (file or database) known as a wallet; it is used to generate digital signatures through specific signature algorithms, thereby authorizing the spending of the associated funds.

The public key is deterministically derived from the private key and can be safely shared with third parties to enable the receipt of payments. By construction, the computational properties of the system make it impractical to derive the private key from the public key, an essential requirement so that disclosing the public key does not compromise the ability to authorize transfers.

By way of analogy, the private key can be compared to the PIN of a payment card, which is required to authorize operations and is kept secret by the holder; the public key can be compared to a banking identifier (for example, an IBAN), which can be used to receive transfers but is insufficient to dispose of the funds.

Pairs of private and public keys are stored in a software medium (or, more generally, in a dedicated device) known as a wallet, which facilitates their management, the secure backup of private keys, and the derivation of new key pairs.

In the following sections, these elements will be examined in greater detail.

Private keys

A private key is a randomly generated 256-bit sequence, whose representation may adopt different encoding formats.

In principle, an equivalent private key could also be obtained through a repeated random experiment (for example, by recording the outcomes of 256 coin tosses). In practice, however, generation relies on dedicated software and hardware components that draw on suitable sources of entropy. The use of non-cryptographically secure pseudorandom generators can significantly weaken security and entail a risk of fund compromise.

The size of the key space, equal to 2^{256}, makes *brute-force* attacks computationally impractical, since an adversary would have to explore an enormous number of possible values.

The private key constitutes the cryptographic proof required to generate the digital signatures that authorize spending the associated funds. Consequently, it must remain permanently secret: disclosing it to third parties is equivalent to granting full control over the managed assets.

The 256 bits can be expressed in several encodings, including 64-character hexadecimal string or the *WIF* (Wallet Import Format), which uses *Base58Check encoding* (Base58 with a checksum) to reduce reading ambiguities and enable error checking.

Public keys

The public key is disclosed when spending the funds associated with a wallet, because the transaction must include a cryptographic proof (a digital signature) demonstrating ownership of the right to spend with respect to the address in

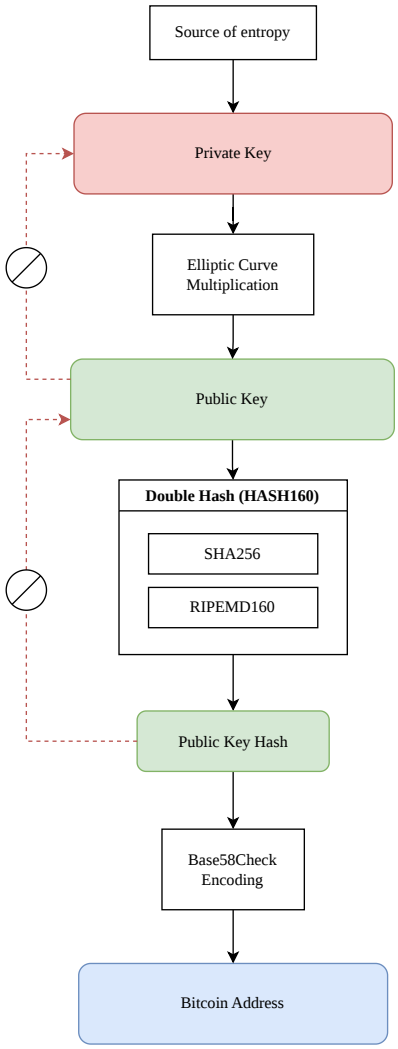


Figure 2.1: The diagram illustrates the process of generating a Bitcoin address from a source of entropy, highlighting one-way transformations, operations that can only be performed in the forward direction and are computationally impractical to reverse.

question. This proof can be publicly verified by any participant in the network, in particular by the nodes responsible for validating transactions and recording them in the distributed ledger.

Addresses

A Bitcoin address is an alphanumeric string that identifies a recipient within the payment system and can be shared publicly to receive transactions. From a cryptographic standpoint, the address does not coincide with the public key; rather, it is obtained through a derivation procedure starting from the public key. Specifically, the public key (typically in ECDSA/secp256k1 format) is processed through a chain of hash functions: first SHA-256 (Secure Hash Algorithm, 256-bit) and then RIPEMD-160 (RACE Integrity Primitives Evaluation Message Digest), producing a 160-bit (20-byte) identifier commonly referred to as the public key hash.

The resulting value is then encapsulated in the format defined by the addressing standard, e.g., Base58Check, which includes a version prefix and a checksum, and is encoded in Base58. This encoding reduces visual ambiguity by removing easily confusable characters (e.g., 0/O and I/l), thereby decreasing the likelihood of dictation or transcription errors. Because Bitcoin transactions are irreversible, sending funds to an address for which one does not possess the corresponding private key results in the permanent loss of the ability to spend those funds.

A hash function f [7] is defined over a domain X and maps to a codomain Y of finite cardinality, with $|X| = n$ and $|Y| = m$, typically with $m \ll n$. A hashing algorithm associates to each element $x \in X$, whose representation requires approximately $\log_2(n)$ bits, an image $y = f(x) \in Y$ (*fingerprint*) representable with approximately $\log_2(m)$ bits. Since the cardinalities of the two sets differ, the presence of collisions is unavoidable, i.e., there exist distinct pairs $x', x'' \in X$ such that $f(x') = f(x'')$.

A high-quality hash function aims to make the collision probability approximately $1/m$, thereby making the event negligible for large values of m, and to ensure a diffusion effect, whereby similar inputs produce significantly different fingerprints.

In the cryptographic setting, hash functions are modeled as *one-way* (non-invertible) functions and satisfy the following properties:

- given an element x, it is computationally efficient to compute $f(x)$
- given a value y, it is computationally infeasible to determine an x such that $f(x) = y$ (*preimage resistance*)
- it is computationally infeasible to find two distinct elements x' and x'' such that $f(x') = f(x'')$ (*collision resistance*)

The use of hash functions in address derivation also helps to limit exposure of the public key: the address can be disclosed without directly revealing the public key, which is typically made public only at the time of spending (specifically, in the unlocking script), together with the digital signature that constitutes the cryptographic proof of authorization. In a future *quantum-computing* scenario, algorithms such as Shor's [30] make it theoretically more efficient to invert certain mathematical problems underlying public-key cryptography (including the discrete logarithm problem on elliptic curves), potentially allowing the private key to be derived from the public key if sufficient quantum resources were available. By contrast, for the hash functions used (SHA-256 and RIPEMD-160) there are no known quantum algorithms capable of compromising their security in an analogous way; consequently, the address (i.e., the hash of the public key) is considered more robust than the public key alone in this context.

Because the public key becomes observable during the spending phase, a hypothetical attacker equipped with a large-scale quantum computer could have only a limited time window to attempt to derive the private key before the transaction is confirmed. On average, this window is on the order of the block production time (about 10 minutes), within which the attacker would need to generate and propagate a competing transaction to an address under their control, potentially offering higher fees to increase its inclusion priority.

A widely recommended measure to mitigate this class of risks (although currently unlikely) and, above all, to substantially increase the privacy of Bitcoin users is to never reuse the same address (address reuse). On the one hand, reuse may direct new funds to an address whose associated private key may have been compromised in the past; on the other hand, and with greater impact, it facilitates the public correlation of transactions, making it possible to infer the identity of users associated with specific addresses, estimate their balances, and monitor subsequent transactions. These activities fall within the scope of so-called chain analysis; in this domain, the company Chainalysis is among the leading operators and is frequently used as a reference in investigative contexts and for tracing on-chain fund flows. [31]

To prevent address reuse, it is necessary to generate, for each incoming transaction, a new pair of cryptographic keys (private and public), corresponding to a distinct address. This process is typically automated by wallets, software or physical devices designed for key generation and management, thereby reducing operational complexity for the user.

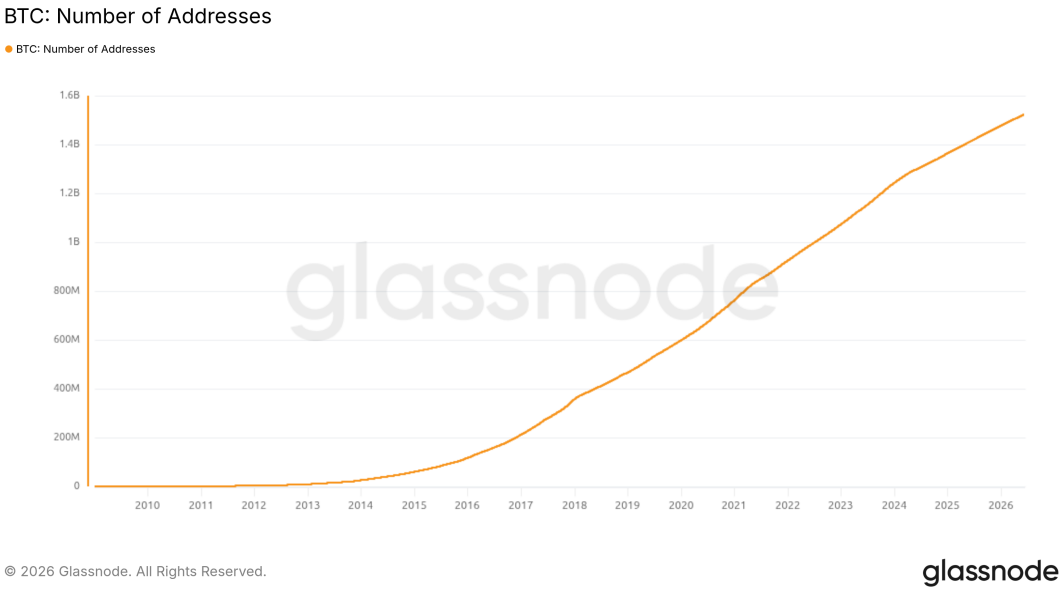


Figure 2.2: The chart illustrates the evolution over time of the total number of addresses created on the Bitcoin blockchain. This figure increases steadily because it is a cumulative measure.

Wallets

A Bitcoin wallet, in abstract terms, can be described as a data structure that enables the management of cryptographic key pairs (a public key and a private key). In operational terms, a wallet corresponds to software or a physical device that allows one to manage transactions on the Bitcoin network, including the derivation and management of addresses, the estimation of the balance associated with those addresses, and the generation of the digital signatures required to authorize the spending of funds.

Wallet implementations can be classified, as a first approximation, into two main categories:

- *non-deterministic* wallets: a now relatively uncommon solution in which private keys are generated independently and without correlation. In this model, the backup procedure is burdensome, as it requires retaining a potentially growing set of private keys generated over time.

- *deterministic* (*seeded*) wallets: the currently predominant approach, in which the set of private keys is deterministically derived from a single initial seed, generated from an initial source of entropy. This setting significantly simplifies backup and recovery, since storing the seed is sufficient to regenerate the

entire hierarchy of derived keys and addresses. To support usability, the BIP-39 standard defines a mnemonic encoding of the seed as a sequence of words drawn from predefined lists (available in languages other than English as well, including Italian, French, and Spanish). The mnemonic representation reduces the likelihood of transcription errors and incorporates a checksum mechanism intended to verify the coherence of the sequence.

If, instead, the wallet is considered as a physical and infrastructural artifact, it is common to distinguish between two main custody modalities:

- *hot storage*: wallets run on devices connected to the Internet, such as desktop applications like Electrum. Network connectivity increases the attack surface, making these solutions generally more exposed to cyber threats.

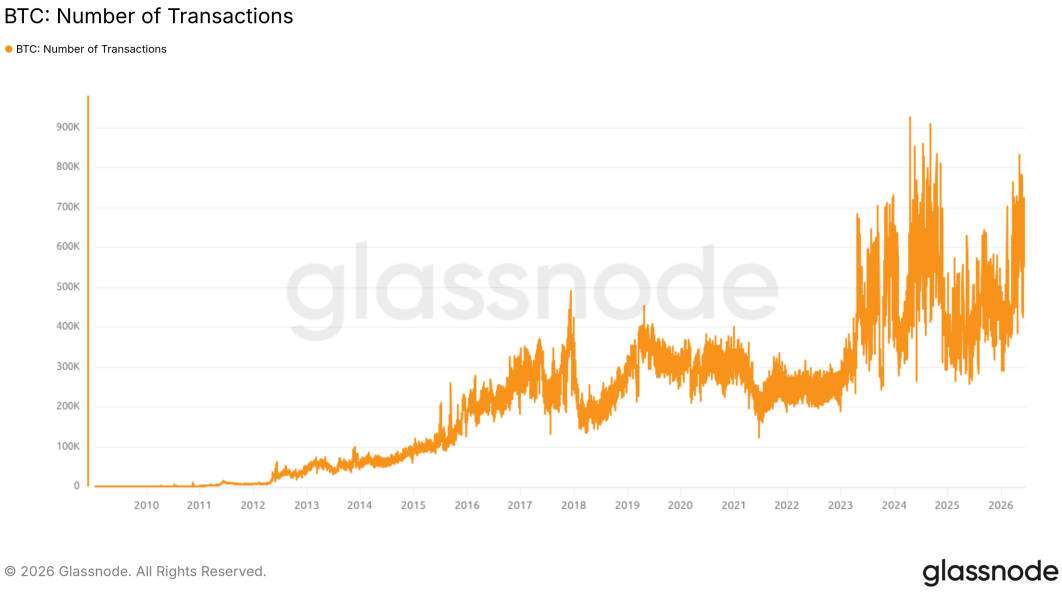
- *cold storage*: devices or systems not connected to the Internet, designed to minimize exposure of private keys. This category includes both computers configured for offline operation (e.g., with connectivity disabled or network interfaces removed) and dedicated hardware devices, typically compact in form, such as *Ledger*, *Trezor*, and *BitBox02*. In a stricter sense, these devices can be interpreted as signing devices, as they primarily perform transaction signing using the stored private keys, without directly broadcasting the transaction to the network. Propagation to the Bitcoin network typically occurs via Internet-connected software that receives the signed transaction and relays it to network nodes, while the private keys remain confined to the offline device.

2.1.2 Transactions

Bitcoin transactions are a core element of the protocol, as they enable the exchange of value between counterparties. Contrary to an intuitive interpretation, the distributed ledger (blockchain) does not maintain an explicit list of balances associated with individual addresses; instead, balances are determined through the unspent output model, based on tracking spendable transaction outputs.

When considering the structure of a Bitcoin transaction, two fundamental components can be distinguished: inputs and outputs. Outputs from transactions that have not yet been spent, referred to as UTXOs (*Unspent Transaction Outputs*), represent spendable units of value. The set of UTXOs associated with a given address, and controllable through possession of the corresponding private key, contributes to defining the effective balance available for spending.

New UTXOs are created within the context of a transaction, which consumes (spends) a set of input UTXOs and produces a set of output UTXOs. The difference between the sum of input amounts and the sum of output amounts constitutes



the transaction fee, paid to the miner, i.e., the entity that includes the transaction in a block during the mining process, discussed in more detail in subsequent sections.

This model makes it possible to perform multiple payments within the same transaction. A common case is the change transaction: because a UTXO is not divisible, transferring to a recipient only part of the value contained in a UTXO requires spending it in full and generating at least two outputs, one directed to the payment recipient and one to a change address controlled by the sender (typically a new address within the same wallet).

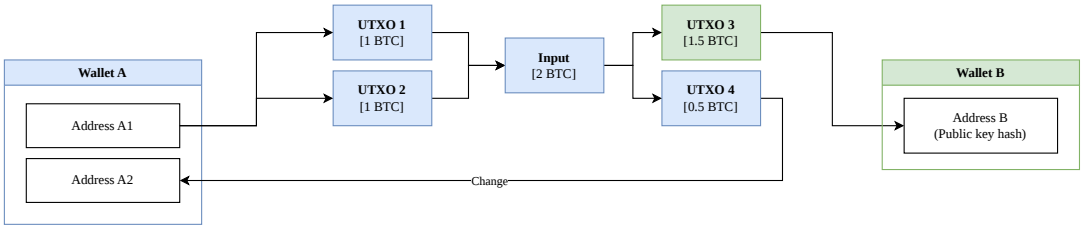


Figure 2.3: The diagram illustrates the operational scheme of a Bitcoin transaction. To transfer funds from wallet A to wallet B, the sender's wallet selects and aggregates multiple UTXOs as inputs in order to reach a total amount equal to or greater than the amount to be transferred. The transaction then generates one or more UTXOs as outputs: a portion is allocated to the intended recipients, while any surplus is returned to the sender via a change UTXO, sent to an address controlled by the sender's wallet. This address is often newly generated to enhance privacy and reduce the linkability between transactions.

Similarly, to make higher-value payments it is possible to aggregate multiple UTXOs as inputs to the same transaction. However, increasing the number of inputs and/or outputs increases the transaction size (measured in bytes) and, consequently, may lead to higher fees, since the transaction occupies a larger portion of the available block space, reducing the remaining capacity for other transactions that may be more profitable.



Script language

Unspent Transaction Outputs (UTXOs) represent units of value that have not yet been spent and can be spent exclusively by the party in possession of the corresponding private key. These outputs are constrained by a locking script that specifies the conditions required to authorize spending; in particular, spending is permitted only for an entity capable of providing a valid cryptographic proof, typically a digital signature generated using the associated private key.

The validation mechanism relies on Bitcoin's scripting language, known as *Script*, which is characterized by a stack-based execution model. Unlike Turing-complete languages, in which arbitrary computations can be expressed, Script is intentionally restricted to a limited set of operations. This design choice helps ensure predictable execution times and constrains the complexity of scripts verified on-chain, thereby reducing the risk of bugs and vulnerabilities that could compromise network operation or enable *denial-of-service* attacks. By comparison, Ethereum natively supports Turing-complete smart contracts, including iterative constructs such as loops, which can introduce potentially unpredictable execution times. However, Ethereum's fee model differs: whereas in Bitcoin the transaction cost is primarily a function of the space occupied in the block (and thus of the transaction and script size in bytes, rather than the amount transferred or the computational complexity), in Ethereum fees are tied to the computational cost of the operations executed (gas), and the block constraint is expressed in terms of an overall computational budget rather than byte size.

Script execution takes place via a LIFO (Last-In, First-Out) stack, on which elementary push and pop operations are performed on the most recently added elements.

In the typical case of transferring funds, the spending logic is divided into two components: an unlocking script, provided in the transaction that spends the UTXO, and a locking script, defined in the original output. The former serves to demonstrate authorization to spend and generally includes the digital signature produced with the private key associated with the address controlling the UTXO. The latter establishes the conditions that must be satisfied for outputs to be spendable as inputs in subsequent transactions, typically requiring the production of a valid signature attributable to the recipient.

The most common form of locking script is *Pay-to-Public-Key-Hash* (P2PKH), since the recipient's address is represented by the hash of the public key; historically, this has been the predominant type in Bitcoin transactions. Despite the language constraints, Bitcoin allows the implementation of more articulated spending conditions, which can be understood as elementary forms of smart contracts. A more flexible and widely adopted approach is *Pay-to-Script-Hash* (P2SH), which makes spending conditional on satisfying specific requirements, such as time locks

and multisignature schemes, while deferring to the time of spending the presentation of the script (the redeem script) whose hash had been originally committed in the output.

Stack operators

In Bitcoin's scripting language, scripts are composed of constants (numbers, addresses, keys, or arbitrary data) and operators. During execution, these elements interact with a stack data structure, onto which operators can push or from which they can pop one or more items. Operators are identified by standardized codes, known as *opcodes*.

Among the most relevant for describing transactions and contractual constructs in the context of the Lightning Network are:

- `CHECKSIG`: verifies the validity of a digital signature provided as input against the corresponding public key included in the script. It is commonly used in locking scripts to specify and enforce the spending conditions of outgoing UTXOs.
- `CHECKMULTISIG`: enables verification of a multisignature (*m-of-n*), specifying a set of public keys and a minimum threshold of valid signatures required to authorize spending. Funding transactions used to open payment channels in the Lightning Network are typically based on multisignature schemes.
- `CHECKLOCKTIMEVERIFY`: introduces a time constraint that prevents an UTXO from being spent before a specified time instant or block height is reached. The parameter can be expressed either as a Unix timestamp or as a block height.

Digital signatures

The fundamental mechanism for attesting ownership of funds in a blockchain system, as the ability to authorize their expenditure, lies in possession of the private key, which is required to produce valid signatures on transactions associated with the corresponding public key and address. This property is implemented through digital signature schemes, which make it possible to demonstrate, in a verifiable manner, the authorization of the private-key holder without revealing the key itself.

A digital signature must preserve, in the electronic context, properties equivalent to those of a handwritten signature affixed to a document. In particular, it must guarantee:

- authenticity and resistance to forgery, since only the party in possession of the corresponding private key can generate a valid digital signature;
- non-reusability, because the signature is cryptographically bound to the document on which it was created and cannot be transferred or applied to different documents;
- content integrity, because the signed document must not be modifiable without the tampering being detectable;
- non-repudiation, as generating the signature is possible exclusively by means of the private key held by the signer.

The main operations in the context of digital signatures are generation (signing) and verification. In the signing phase, the sender computes the hash of the message m and generates the signature s by applying a signing function F with their private key k.

$$s = F(\text{hash}(m), k)$$

In the verification phase, the algorithm takes as input the signature s, the sender's public key K, and the hash of the signed message, returning a boolean value (true) if the signature is valid.

The digital signature has essential properties for its use in blockchain contexts. In particular, its generation is exclusively bound to possession of the private key, which enables the production of a valid signature that can be verified against the corresponding public key. Verification, in fact, can be performed by any party using the signature and the public key, thereby ensuring the non-repudiation of the signing act. Moreover, once applied, the signature cannot be altered without invalidating it, thus ensuring the integrity of the signed message.

In the Bitcoin protocol, the digital signature scheme employed is ECDSA (Elliptic Curve Digital Signature Algorithm). Signature verification is performed at the scripting-language level via specific stack operators, including, `OP_CHECKSIGVERIFY` and `OP_CHECKMULTISIGVERIFY`. These opcodes take the signature and the public key as inputs and evaluate their cryptographic validity, thereby determining the outcome of the authentication required by the script.

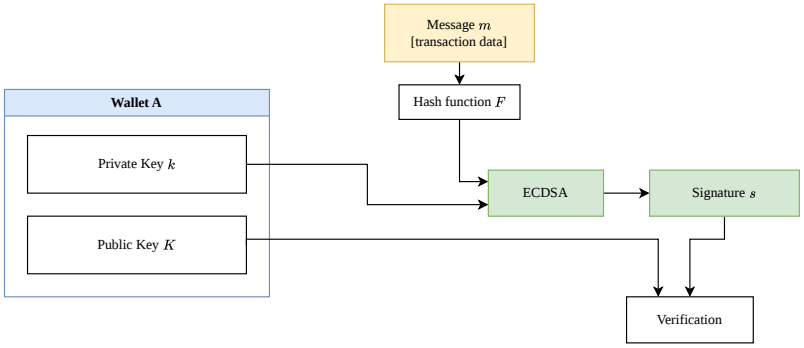


Figure 2.4: The diagram illustrates a digital signature scheme used in the Bitcoin protocol, in which the cryptographic signature of the message containing the transaction data is generated using the private key, kept secret and held exclusively by the owner of the funds; the validity of the signature can instead be verified by any party using the corresponding public key.

2.1.3 Blocks and mining

In the Bitcoin protocol, transactions propagated among nodes of the *peer-to-peer* network are aggregated into blocks, i.e., data structures linked sequentially and cryptographically bound by hash functions; from this concatenation derives the term blockchain, namely a "chain of blocks." Each block includes a header containing information and metadata, including the timestamp and a reference to the hash of the previous block. Once a block is included in the chain, retroactive modification becomes effectively impracticable, because any alteration would require recomputing the cryptographic links for that block and for all subsequent blocks.

Synchronization of the blockchain state among network nodes (potentially on the order of thousands) is achieved through a decentralized consensus mechanism. This consensus is an emergent property arising from the interaction of a large number of independent nodes, in the absence of any central authority responsible for coordination.

Each node receives, verifies (validates), and propagates new transactions awaiting inclusion in a block. A subset of nodes also performs mining, aimed at creating new blocks. Miners select transactions not yet included (typically maintained in a local storage structure known as the mempool), often prioritizing those associated with higher fees, and group them into a candidate block. The SHA-256 hash function is then repeatedly applied to the candidate block, including the reference to the previous block's hash; at each iteration, appropriate header fields are modified (in particular, an incremental counter called the nonce) in order to obtain a digest below a threshold value (target) imposed by the protocol. The target is adjusted

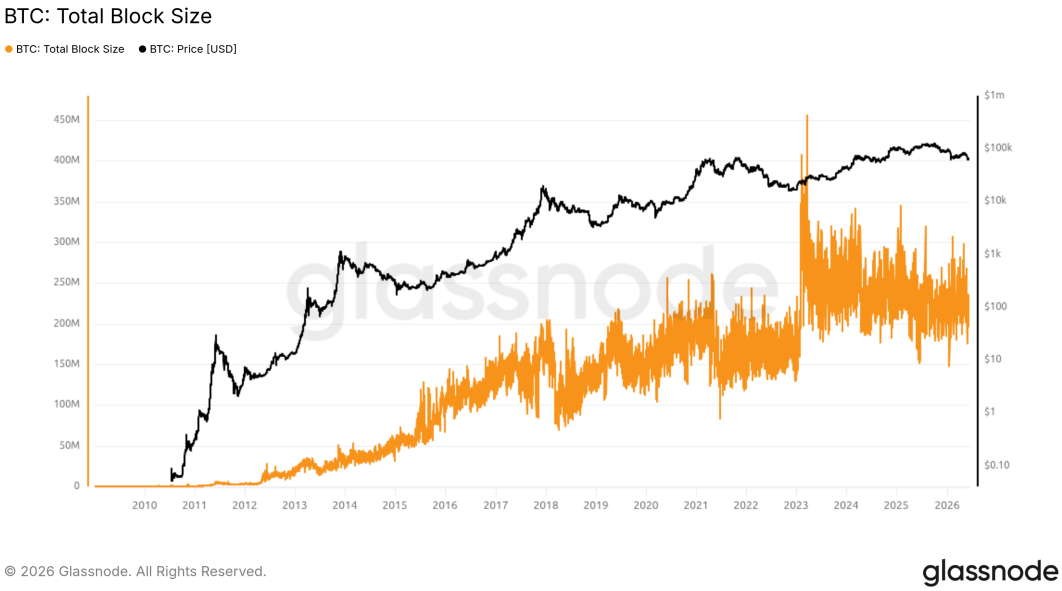


Figure 2.5: The chart illustrates the evolution over time of block sizes in the Bitcoin blockchain. It shows that, during periods characterized by more intensive network usage, block size tends to increase and is closely associated with the volume of transactions included, as reported in Fig. 2.1.2. The reduction in the space available within blocks leads to a rapid increase in transaction fees, consistent with a price-formation mechanism driven by the interaction between supply and demand, as discussed in the following sections and highlighted in Fig. 2.8.

dynamically based on the average production times of previous blocks, with the aim of maintaining an average generation time of about 10 minutes; difficulty is updated every 2016 blocks, corresponding on average to about 14 days, based on the total duration of the most recent 2016-block interval.

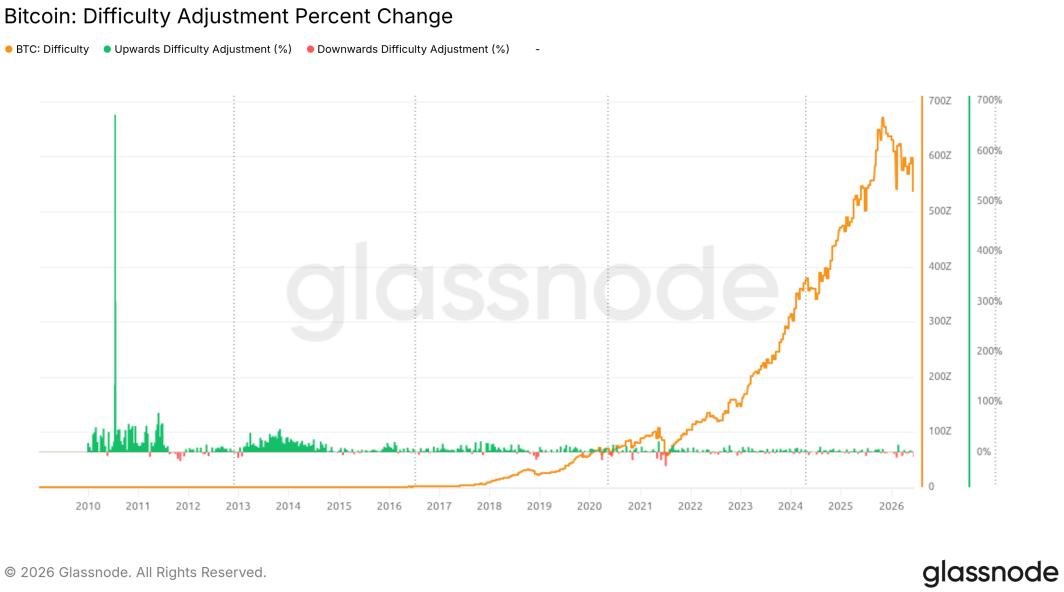


Figure 2.6: The chart highlights the overall trend and the relative variations in the difficulty of the Bitcoin mining process. Although it exhibits a generally upward trajectory, indicative of sustained interest in mining activity, difficulty has repeatedly experienced sharp decreases. These contractions are frequently associated with abrupt price declines or with economic, political, and regulatory events that have reduced mining profitability and led many operators to shut down their computing infrastructure. As a result, the network's overall hashrate decreased significantly, as shown in Fig. 2.7.

Competition among miners to produce the next block entails the use of substantial computational resources, in order to increase the probability of finding a valid solution. This use involves both dedicated hardware and energy consumption: for efficient SHA-256 computation, specialized circuits (ASICs) are widely used, more efficient than general-purpose CPUs or GPUs, and capable of maximizing the hashrate (number of hashes computed per second) for a given amount of electrical power consumed; additional consumption is attributable to heat dissipation systems.

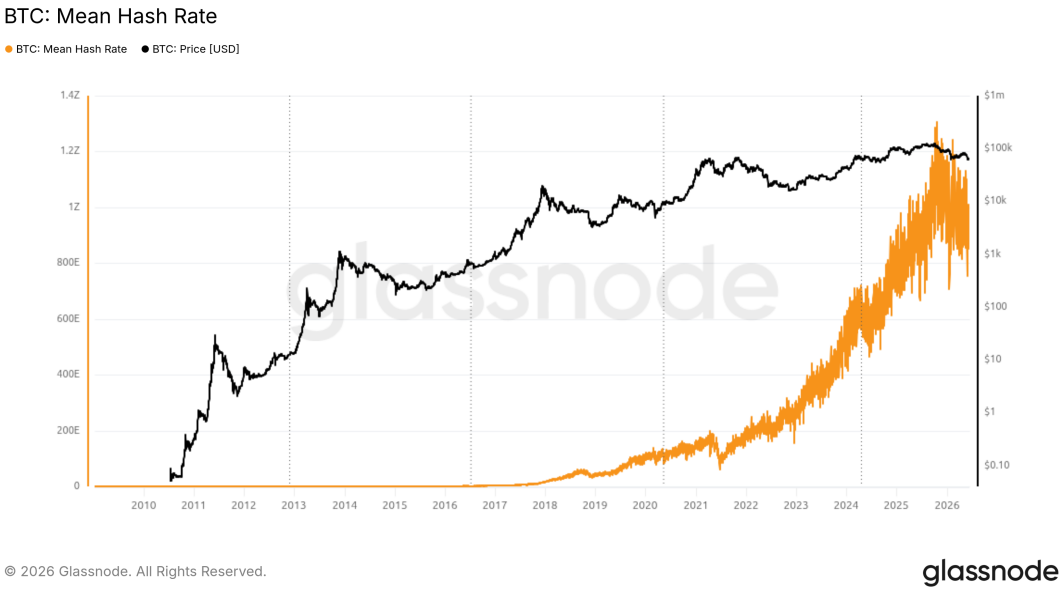


Figure 2.7: The chart illustrates the temporal evolution of the network's mean hashrate, defined as the sum of the computational power made available by active miners. Since no global mechanism exists to measure this quantity directly, the reported hashrate is an estimate derived from the network difficulty and the average block generation time.

Mining is economically incentivized. First, the miner who produces a valid block collects the fees associated with the included transactions, which motivates the priority given to transactions with higher fees. Second, the block contains a special transaction, called *coinbase transaction*, which has no inputs and includes an output (UTXO) assigned to the miner, through which the monetary issuance prescribed by the protocol is allocated. The total amount received by the miner for a block is commonly referred to as the block reward and includes both the newly issued subsidy and the sum of transaction fees.

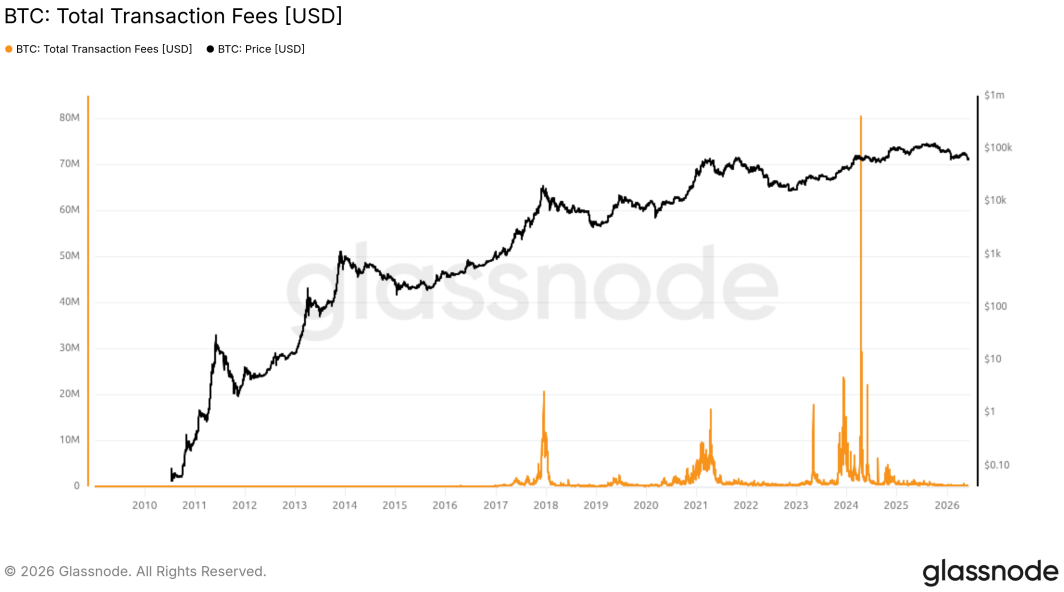


Figure 2.8: The chart illustrates the historical evolution of the total fees associated with transactions and received by miners. A clear association emerges between periods characterized by high network usage intensity and phases of rapid growth in the asset's price. In such contexts, the increase in trades executed by investors raises the demand for transactions to be included in blocks, reducing available space and generating congestion. This results in intensified competition among users to obtain confirmation within shorter timeframes, which in turn leads to an increase in the average fee per transaction.

The coinbase transaction is the mechanism by which new monetary units are created in Bitcoin (programmed inflation), as it is the only way new bitcoins are generated. To limit inflation over time, the protocol provides for a periodic reduction in the newly issued component: the coinbase amount is halved every 210,000 blocks, corresponding to approximately four years. In the earliest mined blocks, the subsidy was 50 BTC; subsequent halvings progressively reduced it to 3.125 BTC in 2026. Continuing along this trajectory, around 2140 issuance will become negligible, because the subsidy will fall below the minimum Bitcoin fraction unit (satoshi). In this scenario, miners' economic incentives will depend primarily on transaction fees.

$$R = \frac{50 \cdot 10^8}{2^{\lfloor \frac{B}{210000} \rfloor}} \tag{2.1}$$

The equation describes the inflation-adjustment mechanism, which determines

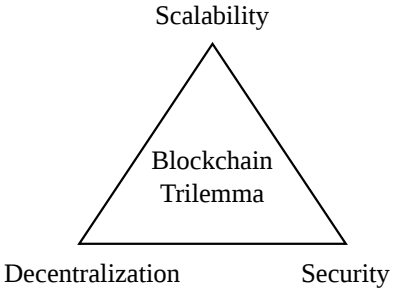
the issuance rate of the new currency R for each *coinbase transaction* as a function of the *block height* B, the index of the most recently mined block.

This scheme of programmed inflation, progressively decreasing until it reaches zero, is a distinctive feature of Bitcoin and has significant monetary implications. In particular, it helps position Bitcoin as an asset more akin to gold than to a modern fiat currency, whose supply can be expanded through discretionary policies set by governments and central banks. Unlike gold, whose availability, while finite, can vary with demand through intensified mining activity, Bitcoin's supply is rigidly determined by the protocol's mathematical rules and is therefore essentially unchangeable, with significant consequences for its economic and monetary profile. [3]

2.2 The Blockchain Trilemma

The term Blockchain Trilemma was coined in 2017 by Vitalik Buterin [9], cofounder of Ethereum, the foremost Turing-complete blockchain for decentralized computation.

The trilemma posits that a blockchain architecture cannot simultaneously achieve high levels of decentralization, security, and scalability.



This constraint has been empirically observed in various alternative protocols designed to prioritize high throughput (TPS), which often resulted in increased centralization by restricting network participation to high-performance nodes.

Recent work [20] provides a mathematical foundation for this trilemma, defining its core properties as follows:

- Decentralization: The ability of the distributed system to function when individual nodes have limited computational resources c.
- Scalability: The network's capacity to process a total number of transactions n such that $n > c$.

- Security: The probability of resisting an adversarial attack from entities controlling resources up to n.

Presently, Bitcoin's maximum block size is 1 MB, accommodating roughly 4,000 transactions. Given a 10-minute average block interval, the network supports an average throughput of 7 transactions per second (TPS). This performance is incompatible with global payment processors like VISA or Mastercard, which can handle peak loads of up to 50,000 TPS.

For years, the scalability debate has centered on increasing the block size or decreasing the block time. However, larger blocks impose stricter hardware requirements regarding bandwidth and storage, leading to node consolidation and reduced decentralization. [2] Conversely, shorter block intervals increase the frequency of spontaneous forks, thereby undermining network security.

Consequently, it has been demonstrated that on-chain scaling alone cannot achieve the performance metrics required for a global monetary system. [16]

To address these limitations, scalability solutions are required to enhance Bitcoin's transaction throughput to a level suitable for a global medium of exchange, without undermining the system's core security or decentralization.

Notably, this scalability is not achieved by modifying the underlying consensus rules or the base-layer protocol. Instead, it is realized through a higher-level layer (Layer 2) engineered on top of the existing primary protocol.

At present, the leading candidate for this paradigm is the Lightning Network, a prominent example of Payment Channel Networks (PCNs) that functions as an overlay to the Bitcoin blockchain.

These protocols are built upon the concept of payment channels between nodes, facilitating off-chain transactions. By operating outside the main consensus layer, these networks can process a high volume of operations while inheriting the robust security guarantees of the underlying blockchain. This is achieved through cryptographic constructs that ensure transaction integrity and provide a trustless mechanism for dispute resolution on the main ledger.

2.3 Scalability solutions

To enhance the scalability of blockchain systems, numerous solutions have been proposed, generally falling into two broad categories. The first includes Layer 1 interventions, aimed at improving performance by modifying the base protocol, for example by changing the distributed consensus rules, the block size, or the block production interval (block time). Because such changes are in most cases not backward compatible, they require hard forks and may lead to a split in the network's consensus; for this reason, the adoption of Layer 1 solutions usually faces resistance from node operators and miners.

A second line of intervention is represented by Layer 2 solutions, which seek to increase scalability without altering the underlying protocol. The approach is to avoid propagating every single payment on the main blockchain, characterized by high costs and latency, and instead to perform transfers off-chain, using the main chain as a dispute-resolution mechanism and as a security anchor.

A requirement commonly considered essential for a protocol to be classified as Layer 2 is the possibility of unilateral exit [12]. In other words, each participant must be able to recover their bitcoin on the main blockchain at any time, within timeframes compatible with handling potential disputes.

Accordingly, several Layer 2 protocols have been developed both to enable additional functionality beyond Bitcoin's base protocol, such as the tokenization of fungible and non-fungible assets, and to improve performance properties, including faster payments, shorter confirmation times, and lower fees, also in support of micropayment scenarios.

Classification

The main types of scalability solutions for Bitcoin can be grouped into six primary design models: [29]

- Sidechains: These are independent blockchains that operate in parallel with Bitcoin's main network (Layer 1) and are connected to it through a "two-way peg" mechanism. This system makes it possible to transfer assets to a secondary chain to benefit from higher speed or lower fees, and then move them back to the mainchain. Examples include solutions based on federated consensus, merged mining, or Proof-of-Transfer (PoX) models.

- State Channels: These allow participants to carry out many off-chain transactions by exchanging signed messages, while recording on the blockchain only the opening and the final closure of the channel. The Lightning Network is the best-known example, and it is particularly well suited to frequent micropayments thanks to the near-instant settlement of transactions.

- Rollups: This technology moves computation off-chain and submits only aggregated data or proofs to the main network in order to increase throughput. They are typically divided into:

  - ZK-Rollups: They use zero-knowledge proofs (ZKPs) to cryptographically attest to the validity of every transaction in the batch.
  - Optimistic Rollups: They assume transactions are valid by default, but include a window during which participants can challenge them by submitting "fraud proofs."

- Inscriptions: These leverage the Ordinals protocol to embed arbitrary data (such as metadata, images, or scripts) directly into Bitcoin transaction fields (specifically the "witness" field). Although initially used for NFTs, they also enable an extension of the network's programmability.
- BitVM: This is an advanced solution that enables the execution of complex (Turing-complete) smart contracts on Bitcoin without modifying the base protocol. It works by translating computational logic into off-chain binary circuits, using the blockchain only for final verification or for dispute resolution through a challenge–response protocol.
- Client-side Verification: In this model, transaction validation does not occur globally on the blockchain; instead, it is handled privately by the individual clients involved. The Bitcoin network is used exclusively as an immutable ledger for cryptographic commitments tied to transaction outputs (UTXOs), ensuring data integrity without burdening the network.

In the following paragraphs, the main scalability protocols currently considered the most promising and most widely adopted are presented.

Liquid Network

Liquid [22] is configured as a federated-consensus sidechain (Federated Sidechain), designed to provide faster and more private transactions than Bitcoin's main network. Rather than relying on Proof-of-Work, the network is operated by a consortium of block signers (*blocksigners*) and guardians (*watchmen*) who validate operations and preserve the system's integrity. Asset transfers occur through a two-way peg mechanism, in which Bitcoin is locked on the mainchain and issued on the sidechain as Liquid Bitcoin (L-BTC) at a 1:1 ratio.

The platform stands out for its implementation of Confidential Transactions, a technology that encrypts transferred amounts and asset types, ensuring that only the parties involved can access the transaction details. In addition, Liquid drastically reduces waiting times by fixing block production at one minute, making it well suited for rapid settlement between exchanges and financial institutions. From a security standpoint, the protocol employs multi-signature authentication and periodic state commitments to the mainchain to mitigate risks associated with the federation's centralization.

Lightning Network

The Lightning Network is one of the most mature and widely adopted second-layer (Layer 2) scalability solutions in the Bitcoin ecosystem, and it is based on the state-channel paradigm. The protocol architecture is designed to mitigate the capacity

and latency constraints of the base blockchain by moving most payment operations off the main chain. This enables micropayments with near-instant confirmation times and transaction costs that are significantly lower than on-chain processing.

Its operation relies on opening payment channels between two counterparties, in which only the opening and closing transactions are published and finalized on the blockchain, while intermediate state updates occur off-chain through the direct exchange of cryptographically binding messages. The security and fairness of the protocol are ensured by advanced contractual mechanisms: *Revocable Sequence Maturity* Contracts (RSMCs) enable dispute management and discourage the use of outdated states, while *Hashed Timelock Contracts* (HTLCs) allow conditional and routable payments, enabling atomic transfers along multi-hop paths. These primitives constitute the core technical components that ensure correctness, economic finality, and network resilience.

Ark Protocol

The Ark protocol [19] represents a significant evolution in the field of state channels, designed specifically to overcome the structural limitations of the Lightning Network. Unlike traditional channels, Ark removes the need to manage inbound liquidity, allowing users to receive payments without complex configuration or preallocation of funds. Another crucial advantage is non-interactivity: recipients do not need to be online or actively participate in signing in order to successfully receive a transaction.

From a technical perspective, Ark operates through an innovative "lifting" mechanism that converts on-chain UTXOs into virtual UTXOs (vUTXOs) without requiring trusted third parties. This process enables operation in a high-performance Layer 2 environment while maintaining a secure link to the main blockchain via a redemption function, which guarantees that funds can be withdrawn back to Layer 1 at any time. Security assessments indicate that Ark provides high scalability and functionality, while maintaining moderate complexity and a multisignature-based security model.

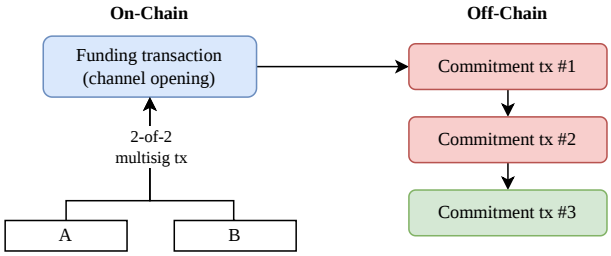
Chapter 3

Lightning Network

The Lightning Network represents the most significant implementation of a payment channel network and, at present, is one of the most widely adopted scalability protocols for Bitcoin.

This chapter covers the foundational concepts governing the operation of the Lightning Network. A more detailed and comprehensive treatment of these principles can be found in the book *Mastering the Lightning Network*. [5]

The operating principle of payment channels is to record on the Bitcoin blockchain (intrinsically slow and costly, with fees typically proportional to transaction size in bytes) only the minimum necessary information. Transactions are therefore carried out primarily off-chain through repeated updates of the channel state between participating nodes, enabling a large number of payments to be executed with reduced latency and low marginal costs. This architecture also makes high-frequency micropayments feasible, for example in consumption-based subscription models or pay-per-use schemes.



During the lifetime of the channel, the parties exchange digitally signed transactions that reflect the latest agreed-upon state, without immediately publishing them to the main blockchain, keeping them locally until the channel is closed. The security of the funds is ensured by cryptographic mechanisms activated both at channel opening and during each state update, which prevent a party from

unilaterally finalizing an obsolete or fraudulent state. Consequently, participants can establish payment channels even with unknown counterparties, significantly reducing the risk of misappropriation of funds.

The first theoretical formalization of payment channels was proposed by Poon and Dryja in 2016 [27]. In the following sections, the design principles will be described systematically, along with the procedures for opening, operational management, and closing a payment channel.

3.1 Payment Channels

Nodes participating in the Lightning Network are publicly identifiable through a unique identifier composed of the node's public key (encoded in hexadecimal format), the IP address, and the listening port exposed on the network. The commonly adopted notation is `<pub_key>@<node_ip>:<port>`, where the default port for the Lightning protocol is `9735`.

The announcement and propagation of reachability and topology information take place via the *Gossip protocol*.

Opening a payment channel between two nodes requires executing a protocol procedure designed to ensure that the channel is created without exposing funds to unnecessary risks. This procedure consists of a sequence of messages exchanged in a deterministic order: `open_channel`, `accept_channel`, `funding_created`, `funding_signed`, `funding_locked`.

The `open_channel` message expresses the intention to establish a payment channel and carries a set of metadata concerning operational parameters and shared expectations, which are necessary to bilaterally negotiate the channel configuration, including, for example, the channel capacity and the sender's public key. In response, the counterparty node sends the `accept_channel` message, in which it confirms or proposes changes to the configuration parameters, including its own public key and the minimum number of confirmations (in terms of mined blocks) required for the funding transaction to be considered valid and, consequently, for the channel to be regarded as operational.

Following acceptance, the initiating node proceeds to construct the funding transaction, i.e., the on-chain transaction that locks funds within the channel. The funds are locked to a 2-of-2 multisignature address, generated from the public keys exchanged in the previous phases, via a script in Bitcoin's scripting language. The standard script is of the form:

```
2 <pub_key_1> <pub_key_2> 2 CHECKMULTISIG
```

The public keys are ordered lexicographically, so as to ensure that both parties

derive the same script and, consequently, the same hash. The resulting output is a P2WSH (Pay to Witness Script Hash) output.

Subsequently, the sender can generate the funding transaction (`funding_created`) by signing a transaction to the newly created multisignature address and transferring the amount defined in the previous phase. In the absence of further safeguards, however, the sender cannot yet broadcast this transaction to the blockchain: the funds would in fact be locked in an output spendable only through the counterparty's joint signature. In a trustless setting, the counterparty could refuse to sign (for instance, by demanding compensation to unlock the funds) or become unavailable; in both cases, the lack of the second signature would make the funds effectively unrecoverable.

To mitigate this risk, the sender must have a mechanism that enables a unilateral exit if the negotiation fails. Therefore, in addition to the funding transaction, a safeguard transaction (commitment transaction) is constructed, specifying the conditions for restoring the previous state, the full return of the deposited value to the sender. Because the commitment transaction spends the multisignature output, it must be signed by both parties. The sender prepares it first, as it must include as an input the identifier (transaction id) derived from the hash of the funding transaction.

At this stage, the security of the protocol depends on the stability of the funding transaction's identifier. Transaction malleability (the ability to alter the transaction id by varying certain elements of the digital signature) would have undermined the validity of the commitment transaction and, consequently, the security of payment channels. The introduction of SegWit resolved this vulnerability by separating signature data from the transaction body, thereby making the transaction id effectively immutable. [34] It follows that the id computed by the party proposing the channel opening matches the one that will be observed once the transaction is broadcast, preserving the validity of the commitment transaction.

The sender then sends the commitment transaction to the counterparty to obtain the countersignature and receive it back duly signed. Only after both signatures have been applied can the funding transaction be safely broadcast: in the event of misconduct or unavailability on the part of the counterparty, the commitment transaction, already validly signed, can be propagated to return the channel to its initial state.

3.1.1 Sending payments

In the channel's initial state, the entire liquidity endowment is allocated to the sender's side, i.e., the party that opened and funded the payment channel. The execution of subsequent payments is enabled by the generation of new commitment transactions, which update the channel state and simultaneously invalidate all

previous states.

Each state update consists in constructing a pair of new commitment transactions, exchanged and countersigned by the channel parties, but not immediately broadcast nor recorded on the main blockchain. These transactions are designed through cryptographic mechanisms and economic incentives that make it disadvantageous to publish an obsolete state on-chain relative to the most recent one. Consequently, any attempted fraud based on broadcasting a prior commitment transaction results in an economic penalty imposed on the dishonest party.

Since a commitment transaction, once signed with valid digital signatures, cannot be technically revoked, protection against the publication and enforceability of obsolete states relies on an economic disincentive borne by the dishonest party. This guarantee is achieved by generating two mirrored and asymmetric commitment transactions, which regulate the spendability conditions of the funds between the parties to the payment channel in different ways. In particular, the Bitcoin script of the commitment transaction introduces a time constraint (timelock) on the funds allocated to the holder of that transaction: these funds can be spent only after an interval determined by the to_self_delay parameter, expressed in number of blocks and negotiated when the channel is opened, generally proportional to the channel's capacity. By contrast, the funds assigned to the counterparty are immediately spendable and not subject to time constraints.

Consequently, if a party attempts to broadcast a prior commitment transaction to the network, they will be unable to access the funds allocated to them for the duration of the timelock, thereby granting the counterparty a sufficient time window to detect the fraudulent behavior and respond, for instance by publishing a penalty transaction.

The penalty mechanism provides that the output paying the alleged attacker, in addition to being spendable only after the timelock by the corresponding address, can also be spent immediately by the counterparty provided that the counterparty possesses the corresponding revocation key. The revocation key is revealed to the counterparty following the state update, typically after the new commitment transactions are signed, thus making the publication of an outdated state economically disadvantageous.

Although payment channels can, in principle, remain open for an indefinite period, either party may initiate closure at any time, thereby determining the final settlement on the blockchain. Closure can occur in two distinct ways.

Cooperative closure is the preferred scenario, in which both nodes agree to terminate the channel. In this context, one party submits a closing proposal, which is negotiated and subsequently accepted by the counterparty. On-chain settlement is carried out via a final settlement transaction conceptually analogous to a commitment transaction, with the difference that no time constraint (timelock) is

imposed on the release of the funds; consequently, both parties can immediately receive their respective shares.

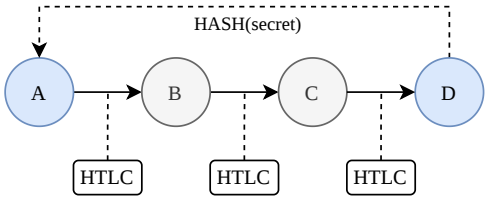
Non-cooperative closure instead arises when it is not possible to reach an agreement. In such cases, one party may use the revocation key to construct a punishment transaction, thereby obtaining the entire channel balance. This transaction allows the spending of both the party's own share, already encumbered in the 2-of-2 multisignature script, and the counterparty's funds, which become immediately spendable by the party executing the revocation. By contrast, the counterparty remains subject to a time constraint (timelock) before being able to dispose of the funds allocated to them.

3.1.2 Routing payments

One of the most relevant features of the Lightning Network is its ability to route payments through a sequence of channels, enabling transactions even between nodes that are not directly connected. This functionality relies on smart contracts implemented using Bitcoin's scripting language, which make it possible to perform multi-hop payments in a trustless manner, thereby reducing counterparty risk.

The fundamental mechanism is provided by HTLCs (Hash Time-Locked Contracts). An HTLC constrains the execution of a payment on knowledge of a secret, called the payment preimage, typically a pseudorandomly generated value. From the preimage, a hash is computed using SHA-256; presenting the correct preimage satisfies the hashlock condition and thus unlocks the funds specified by the script.

In the context of multi-hop payments, HTLCs are chained across intermediaries so that the release of funds occurs in a coordinated way along the entire route: revealing the preimage at one hop triggers, in a cascading manner, the unlocking of the preceding contracts. The time constraint (timelock) additionally introduces a deadline by which the transaction must complete; if finalization does not occur, funds are automatically returned to their respective owners. This ensures payment atomicity, preventing intermediate states caused by partial or incomplete payments.



Each intermediary node retains a fraction of the amount as a fee, providing the economic incentive to participate in routing. The protocol typically specifies

a base fee (a fixed charge) and a fee proportional to the routed amount, as will be discussed in the next chapter.

3.2 The Gossip protocol

The Lightning Network can be modeled as a dynamic graph, characterized by a time-varying topology. This evolution is driven by the opening and closing of channels, the addition of new nodes, and updates to the fee policies of routing nodes. Each node must maintain a local reconstruction of this graph to effectively perform payment routing.

It is noteworthy that funding transactions, which consist of a transaction to a Pay-To-Witness-Script-Hash (P2WSH) address, are publicly recorded on the blockchain. However, the specific nature of the script (a 2-of-2 multisignature) is only disclosed when the transaction output is spent. Consequently, nodes require an independent mechanism to derive and update the channel graph.

To achieve this, the Lightning Network implements a Gossip protocol, a standard approach in peer-to-peer (P2P) networks. This protocol allows nodes to construct and synchronize the channel graph while maintaining connections with only a few peers (typically 3 or 4). The fundamental mechanism involves forwarding any newly received information, such as a channel announcement, to all connected peers. To prevent redundancy and mitigate network spam, a node broadcasts a specific piece of information only upon its initial receipt.

3.2.1 Procedure

When a new node joins the Lightning Network, it integrates into the ecosystem by following three distinct operational phases:

1. Peer Bootstrapping: The node must first identify and establish connections with a set of active peers already present in the network. P2P bootstrapping is a well-documented challenge, requiring robust mechanisms to ensure both efficiency and resilience.

2. Channel Graph Download and Validation: After connecting to its initial peers, the node begins downloading the data required to construct the network topology. During this phase, the node receives channel announcements and must validate this information against the blockchain to ensure the local state is both accurate and authentic.

3. Channel Maintenance and Synchronization: The node must continuously process incoming Gossip messages to maintain an up-to-date view of the

network. Ongoing synchronization is vital for payment routing efficiency, as it ensures the node possesses the latest information regarding channel liquidity and routing policies.

3.2.2 Messages types

To maintain an updated state of the network graph, nodes must continuously exchange messages with their peers following the initial synchronization. Specifically, there are three primary message types used to propagate information regarding newly added nodes, the opening of new channels, and modifications to routing policies. Collectively, these messages form the Gossip Protocol, which ensures that each node maintains a consistent local view of the network topology.

- `node_announcement`: message broadcast when a new node joins the network and intends to announce its presence publicly. The node provides a unique identifier (consisting of its public key), connection details such as IPv4, IPv6, or Tor addresses, and protocol-level features. The latter is represented by a bitvector of feature flags, which informs peers about the specific functionalities the node supports or requires.

- `channel_announcement`: this message informs the network of the creation of a new payment channel. Each announcement must correspond to a valid on-chain transaction on the blockchain; nodes verify this transaction before propagating the message to prevent the creation of unverified channels, which could lead to Denial of Service (DoS) attacks during payment routing. The transaction to be validated consists of a multisig output controlled by the two endpoints of the channel. Along with the cryptographic proof of ownership, the message includes a bitvector of supported features, specifying the technical capabilities of the channel.

- `channel_update`: this message is used to update the routing policies of a payment channel, such as the base and proportional fees. Each channel update includes a *timestamp*, which allows network nodes to synchronize with the most recent channel state, regardless of the order in which messages are received. This information is crucial for passive channel rebalancing and will be analyzed and discussed in greater detail in a subsequent section.

It is important to note that nodes are not required to announce themselves or their channels. They may choose to do so if they intend to participate in routing activities, where a reconstruction of the network graph, even if incomplete, enables other peers to perform pathfinding for payments.

Alternatively, two or more nodes may choose to open and manage their payment channels privately (*unannounced* or *private channels*), without the need to publicly broadcast state updates via the Gossip Protocol.

Consequently, the Lightning Network graph analyzed in this thesis should not be considered exhaustive; rather, it represents only the subset of information explicitly and publicly shared by network participants.

Chapter 4

Data collection and network analysis

4.1 Data collection

4.1.1 Introduction

In a public, permissionless blockchain such as Bitcoin, every transaction is recorded on a public distributed ledger, ensuring that each full node maintains a complete copy of the entire transaction history.

In contrast to a blockchain, the Lightning Network, or more generally, a Payment Channel Network (PCN), is designed to handle a high volume of small value payments. Consequently, individual payment details are not recorded globally by all nodes. As described in previous sections, PCNs are built upon cryptographic protocols anchored to the Bitcoin blockchain, yet they eliminate the need to register every transaction on-chain. These transactions remain private, involving only the sender, the receiver, and any intermediate routing nodes.

In a PCN, the concept of topology is therefore fundamental; it is the structural foundation that enables payment routing and determines the overall resilience and efficiency of the system.

The analysis and optimization of the Lightning Network proposed in this thesis are based on a network snapshot taken on March 30, 2026. This section details the methodology employed to retrieve the data, ensuring the process is both reproducible and verifiable.

4.1.2 Installing a Lightning Node

The most direct and native method to retrieve the Lightning Network graph is by running a dedicated Lightning node, thereby actively participating in the peer-to-

peer network.

There exists multiple implementations of the Lightning Network protocol:

- Lightning Network Daemon (LND): `Go` implementation of the LN
  https://github.com/lightningnetwork/lnd
- Core Lightning (CLN): `C` implementation of the LN
  https://github.com/elementsproject/lightning
- Eclair: `Scala` implementation of the LN
  https://github.com/acinq/eclair

In this project, an LND (Lightning Network Daemon) node was deployed; this section details the installation procedure and the methodology used for data acquisition.

Stystem requirements

Running a Lightning node requires modest hardware; for instance, a Raspberry Pi 5 with 8GB of RAM and a 128GB microSD card is sufficient for basic operations.

By running LND in Neutrino mode, a light client protocol that connects to a remote Bitcoin full node via RPC (Remote Procedure Call) to verify transactions, it is possible to set up a functional and synchronized Lightning node within minutes. In this mode, the network graph can be retrieved, but information regarding channel capacities will be missing. This limitation exists because Neutrino mode does not perform full on-chain transaction verification.

Consequently, connecting to a Bitcoin full node is mandatory if channel capacity data (essential for this analysis) is required. Initializing a Bitcoin full node involves the Initial Block Download (IBD), which requires downloading and verifying approximately 700GB of transaction data. This process demands high-performance hardware and a stable, high-speed internet connection.

While a Raspberry Pi 5 can still be used, it is strongly recommended to use a high-performance SSD instead of a microSD card, as the latter is prone to rapid wear due to intensive I/O operations and would significantly bottleneck the IBD process. Furthermore, verifying digital signatures for all transactions is CPU-intensive and could take several days on such hardware.

For this project, a Bitcoin full node was deployed on a VPS with the following specifications:

- 6 vCPUs
- 18GB RAM

- 1TB SSD
- 600 Mbit/s connection

In general, the critical requirements are storage, at least 1TB capacity with sufficient speed for rapid index operations (essential for verifying channel funding transactions), and a high-bandwidth connection to speed up the IBD process.

Download

To install the node, the software can either be built from source code retrieved from the official GitHub repository and compiled locally, or deployed using precompiled binaries:

```
wget https://github.com/lightningnetwork/lnd/releases/download
/v0.20.1-beta/lnd-linux-amd64-v0.20.1-beta.tar.gz
```

Precompiled binaries for each operating system and architecture are available in the official repository:

```
https://github.com/lightningnetwork/lnd/releases
```

Installation

To install the binaries, the downloaded compressed archive is extracted using the following command:

```
tar -xvf lnd-linux-amd64-v0.20.1-beta.tar.gz
```

Subsequently, the executable files must be installed into a directory within the system PATH, commonly `/usr/local/bin`. The following command ensures the correct ownership and execution permissions are set:

```
sudo install -m 0755 -o root -g root -t
/usr/local/bin lnd-linux-amd64-v0.20.1-beta/*
```

Once this step is completed, the lnd daemon and the lncli (Lightning Network Command Line Interface) utility will be globally accessible via the terminal.

Configuration

LND configuration is defined via the `lnd.conf` file. Create the directory and open the file:

```
mkdir -p ~/.lnd && nano ~/.lnd/lnd.conf
```

The newly created configuration file should have the following structure:

```
[Application Options]
alias=node
color=#000000
debuglevel=info

[Bitcoin]
bitcoin.active=1
bitcoin.mainnet=1
bitcoin.node=bitcoind

[Bitcoind]
bitcoind.rpcuser=*******
bitcoind.rpcpass=*******
bitcoind.zmqpubrawblock=tcp://127.0.0.1:28332
bitcoind.zmqpubrawtx=tcp://127.0.0.1:28333
```

The key configuration parameters are detailed below:

- Node Metadata (alias and color): Optional public attributes associated with the node's identity, used to facilitate node identification within network visualization tools.
- Logging Level (debuglevel): Determines the verbosity of the system logs. The info level is recommended for effectively monitoring the node's synchronization progress.
- Bitcoin Backend Configuration: By designating Bitcoin Core (bitcoind) as the backend, the Lightning node can independently verify on-chain transactions.

The communication interface between the Lightning node and the Bitcoin full node relies on two specific protocols:

- RPC (Remote Procedure Call): A bidirectional communication channel that allows the Lightning node to query blockchain data and broadcast on-chain transactions.

- ZMQ (ZeroMQ): A high-speed, low-latency asynchronous messaging protocol utilizing a pub/sub (publisher/subscriber) pattern. It is employed to notify the Lightning node of real-time events, such as the mining of new blocks or on-chain updates to channel states.

Setup

To finalize the Lightning node configuration, the lnd daemon must be initialized by executing the command:

```
lnd
```

Upon startup, the daemon will signal that the associated wallet must be either created or unlocked. In a separate terminal session, the user can interface with the daemon using the following lncli commands:

- `lncli create`: Used for initial setups. This command facilitates the generation of a new wallet, displaying the corresponding mnemonic seed phrase for recovery purposes, or the restoration of an existing wallet through supported recovery methods.
- `lncli unlock`: Used for subsequent startups. This command grants access to the previously configured wallet by prompting the user for the encryption password.

Syncronization

Upon startup, the node establishes connections with Lightning peers and initiates the synchronization of the network topology via the gossip protocol. This process, which typically spans several hours depending on network latency and peer selection, is essential for constructing a local view of the active payment channels and nodes.

The synchronization status can be monitored using the following command:

```
lncli getinfo
```

The alignment with the network graph is finalized once the `synced_to_graph` field returns `true`. To accelerate graph convergence, it is often beneficial to manually connect to highly-connected nodes (e.g., major network operators).

Example connection to the ACINQ node:

```
lncli connect
03864ef025fde8fb587d989186ce6a4a186895ee44a926bfc370e2c366597a3f8f
@3.33.236.230:9735
```

In this context, the identifier preceding the `@` symbol is the node's public key (pubkey), while the subsequent IP address and port specify the network endpoint.

Data collection

Once synchronization is complete, the entire public graph topology can be exported and stored locally in JSON format.

This data extraction is performed via the following command:

```
lncli describegraph > ln.json
```

The generated file provides a comprehensive overview of all publicly announced nodes and payment channels.

The structure of this dataset, along with its specific technical parameters (such as channel capacities, routing policies, and node identifiers) will be examined in depth in the following chapters.

4.1.3 Running a Bitcoin full node

As previously established, the operational integrity of a Lightning node necessitates a connection to a Bitcoin full node. This software acts as a cryptographic validator, independently verifying transaction legitimacy and asset ownership without relying on external intermediaries.

Download and Installation

Users may either compile the source code or download pre-compiled binaries for their specific architecture:

```
wget https://bitcoin.org/bin/bitcoin-core-28.1/
bitcoin-28.1-x86_64-linux-gnu.tar.gz
```

The complete list of available platforms:

```
https://bitcoin.org/bin/bitcoin-core-28.1/
```

Following extraction, the binaries are moved to the system PATH to ensure global availability:

```
tar -xvf bitcoin-28.1-x86_64-linux-gnu.tar.gz

sudo install -m 0755 -o root -g root -t
/usr/local/bin bitcoin-28.1/bin/*
```

Configuration

The node's behavior is defined via the `bitcoin.conf` file, which must be configured prior to initialization:

```
mkdir ~/.bitcoin && nano ~/.bitcoin/bitcoin.conf
```

A standard configuration for Lightning compatibility includes:

```
daemon=1
server=1

txindex=1

rpcuser=*******
rpcpassword=*******

zmqpubrawblock=tcp://127.0.0.1:28332
zmqpubrawtx=tcp://127.0.0.1:28333
```

The `txindex=1` directive enables a comprehensive transaction index. While this significantly increases disk space requirements (by tens of gigabytes), it is highly recommended as the Lightning node requires frequent lookups of historical on-chain data.

Execution and Initial Block Download (IBD)

The daemon is launched using the bitcoind command. Upon execution, the node initiates the *Initial Block Download* (IBD), a process wherein the software downloads and validates every block sequentially, starting from the *Genesis Block*.

```
bitcoind
```

The duration of the IBD is subject to several hardware constraints:

- Network Bandwidth: Required for the transfer of the entire blockchain dataset (exceeding 600 GB).
- Computational Throughput (CPU): Necessary for the intensive verification of ECDSA digital signatures.
- Disk I/O: A critical factor due to the high frequency of read/write operations on the UTXO set.

Progress can be tracked via the command-line interface:

```
bitcoin-cli -getinfo
```

4.1.4 Data structure

The dataset generated by the `lncli describegraph` command represents the network topology in JSON format. This file encapsulates the graph structure by providing a detailed characterization of nodes (network participants) and edges (bidirectional payment channels).

Nodes

Nodes function as autonomous agents within the network, capable of establishing connections and routing payments. Each node is uniquely identified by a public key associated with a Bitcoin wallet and is characterized by the following parameters:

- `last_update`: A timestamp indicating the most recent state update.
- `pub_key`: The unique cryptographic identifier of the node.
- `alias` and `color`: Optional metadata used for node identification in network explorers.
- `addresses`: Network endpoints (IPv4, IPv6, or Tor) used for peer-to-peer communication.
- `features`: A bit-vector signaling the protocol enhancements supported or required by the node.

Channels

Channels are on-chain smart contracts that lock a specific amount of liquidity between two peers.

A channel is defined by the following attributes:

- `channel_id`: A unique numerical identifier derived from the block height and transaction index of the funding transaction.
- `chan_point`: The specific outpoint (transaction ID and output index) on the Bitcoin blockchain that anchors the channel.
- `node1_pub` / `node2_pub`: The public keys of the two channel participants.
- `capacity`: The total liquidity, measured in satoshis, committed to the channel.
- `node1_policy` / `node2_policy`: routing policies of the nodes.

Nodes acting as routing nodes broadcast specific policies to govern how payments are relayed through their channels:

- `fee_base_msat`: A fixed routing fee charged per transaction, denominated in millisatoshis.
- `fee_rate_milli_msat`: A proportional fee based on the transaction value.
- `time_lock_delta`: The minimum time buffer (expressed in blocks) required for HTLC resolution.
- `min_htlc` / `max_htlc_msat`: The allowable range for transaction amounts passing through the channel.
- `disabled`: A boolean flag indicating whether the channel is currently unavailable for routing.

4.2 Graph Analysis

4.2.1 Mathematical Model

The Lightning Network can be modeled as a graph, a mathematical object consisting of a set of nodes and edges that connect pairs of nodes.

Formally, the network can be represented as an undirected weighted multigraph; however, the presence of multiple channels between the same pair of nodes can be handled by aggregating them into a single edge with an appropriate weight, thereby obtaining an undirected weighted graph G.

$$G\,(V, E, \mathrm{cap})$$

The graph G has a set of $n = |V|$ vertices (peers / nodes), contains $m = |E|$ edges (payment channels) of $V \times V$ with a weight (capacity) associated via the function $\mathrm{cap} : E \to N$.

We call $c_e = cap(e)$ the single channel capacity and $C = \sum_{e \in E} c_e$ the total capacity of the network.

Consider a mathematical description of a payment channel defined by the edge $e = (u, v) \in E$, where the quantities e_u and e_v, with $e_u, e_v \in \{0, \ldots, \mathrm{cap}(e)\}$, denote the liquidity held, respectively, by nodes u and v within the channel.

Let L_G denote the set of all possible states of the system, also known as the polytope of wealth distributions:

$$L_G = \{\lambda : E \times V \to \{0, \ldots, C\} | \lambda(e, u) + \lambda(e, v) = \mathrm{cap}(e)\}$$

A polytope [24] is a geometric object in a multidimensional space defined by linear constraints, obtained as the intersection of half-spaces (which enforce that no one has a negative balance) with a hyperplane (which enforces that the total sum of coins is constant).

The number of possible network states is therefore given by the volume of L_G, which can be computed using the following formula:

$$\mathrm{vol}(L_G) = |L_G| = \prod_{i=1}^{m} (c_i + 1)$$

The number of feasible states is maximized when the network has m channels of equal capacity ($\frac{C}{m}$).

$$\mathrm{vol}(L_G) = \left(\frac{C}{m} + 1\right)^m$$

L_G has a fundamental geometric relationship with the polytope W_G (which represents users' total-wealth distributions). There is a natural projection that maps each liquidity state in L_G to a corresponding wealth distribution in W_G.

Different liquidity states in L_G may correspond to the same wealth distribution in W_G. Moving from one state to another can occur through circulations (or circular rebalancings), which change the placement of liquidity across channels without altering the participants' total wealth.

4.2.2 Metrics and statistics

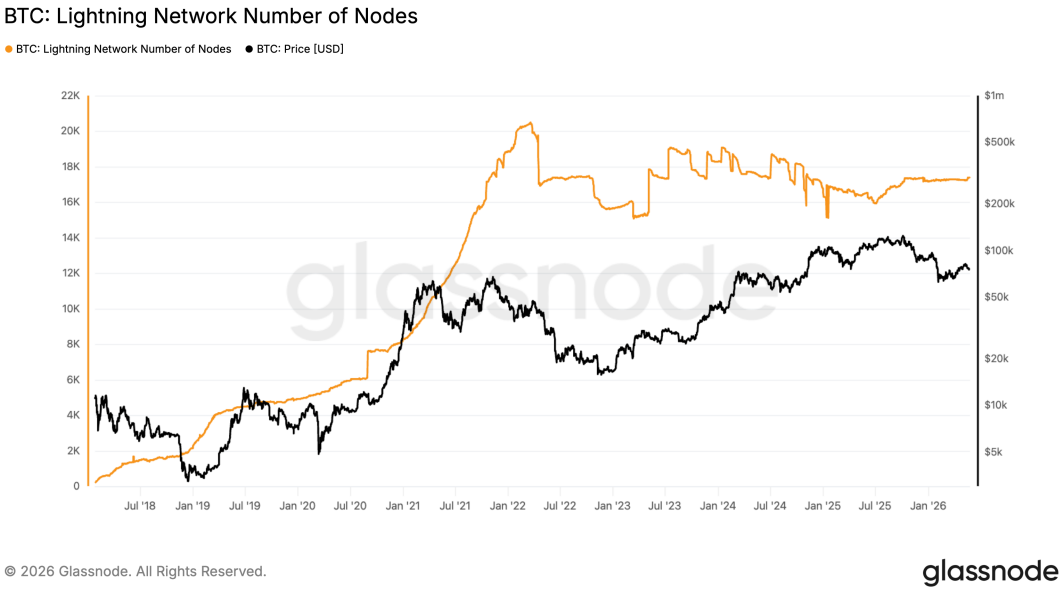
Through a statistical and descriptive analysis [11] of the Lightning Network graph, it is possible to calculate or estimate global parameters of interest that help characterize its structure and main topological properties.

The data used refer to a snapshot of the network state captured on 30 March 2026.

n	Total nodes	$1.3050 \cdot 10^4$
m	Total channels	$4.0350 \cdot 10^4$
d	Density	$4.7390 \cdot 10^{-4}$
L	Network capacity	$3.0589713 \cdot 10^3$ BTC
D	Diameter	13
k_{avg}	Average node degree	6.1839
k^w_{avg}	Average weighted degree	$4.6881 \cdot 10^7$ sat
C	Average clustering coefficient	0.1014
r	Assortativity coefficient	-0.2252
γ	Power law exponent (degree)	-2.0231
γ_b	Power law exponent (betweenness)	-0.8368

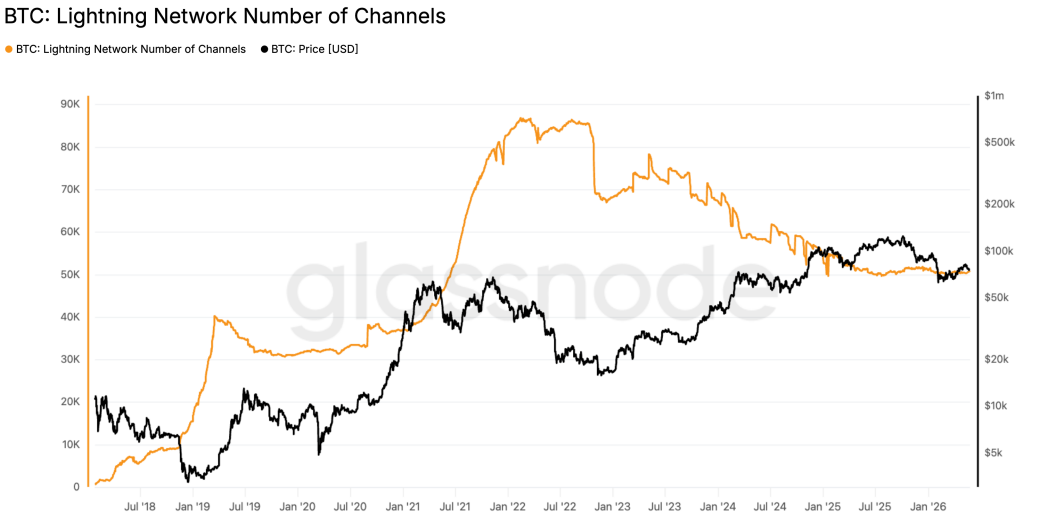
Network order and size

The total number of nodes and channels in a graph is referred to, respectively, as the graph order and the graph size.



© 2026 Glassnode. All Rights Reserved.

glassnode



© 2026 Glassnode. All Rights Reserved.

glassnode

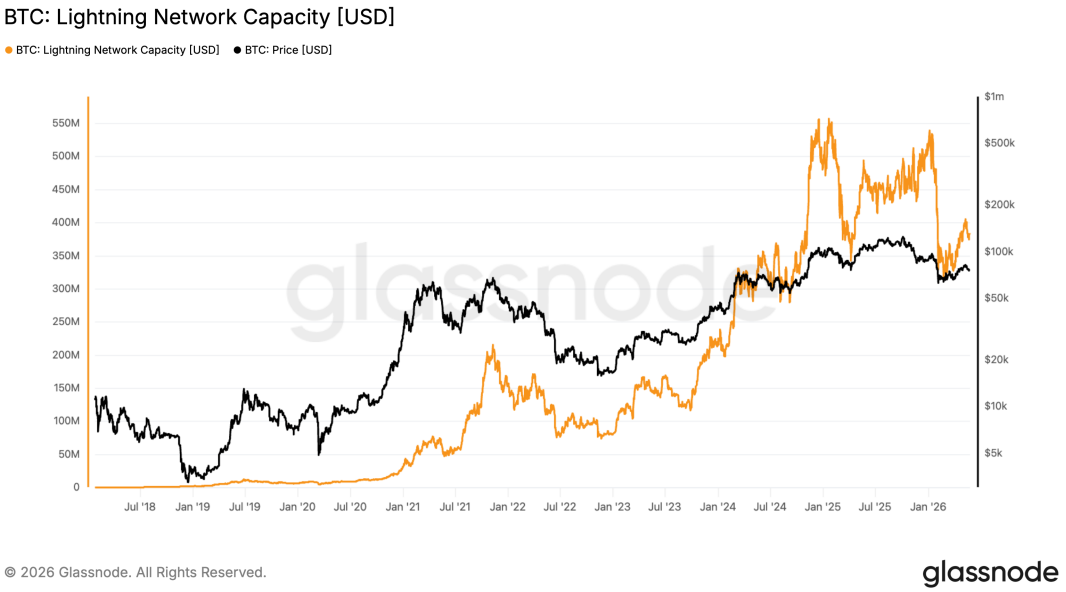
Channel density can be defined as the ratio between the number of observed edges and the maximum number of possible edges that could connect all pairs of nodes.

Network capacity

The total capacity of the network corresponds to the overall amount of liquidity allocated in payment channels.

$$L = \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} a_{ij}^{w}$$

It represents a metric commonly used to assess the evolution of the Lightning Network over time: as the number of nodes increases and new channels are opened, as well as following an increase in liquidity in channels that already exist, this parameter tends to grow progressively. Consequently, total capacity constitutes an indicator of the aggregate capital deployed within the network.



© 2026 Glassnode. All Rights Reserved.

Degree

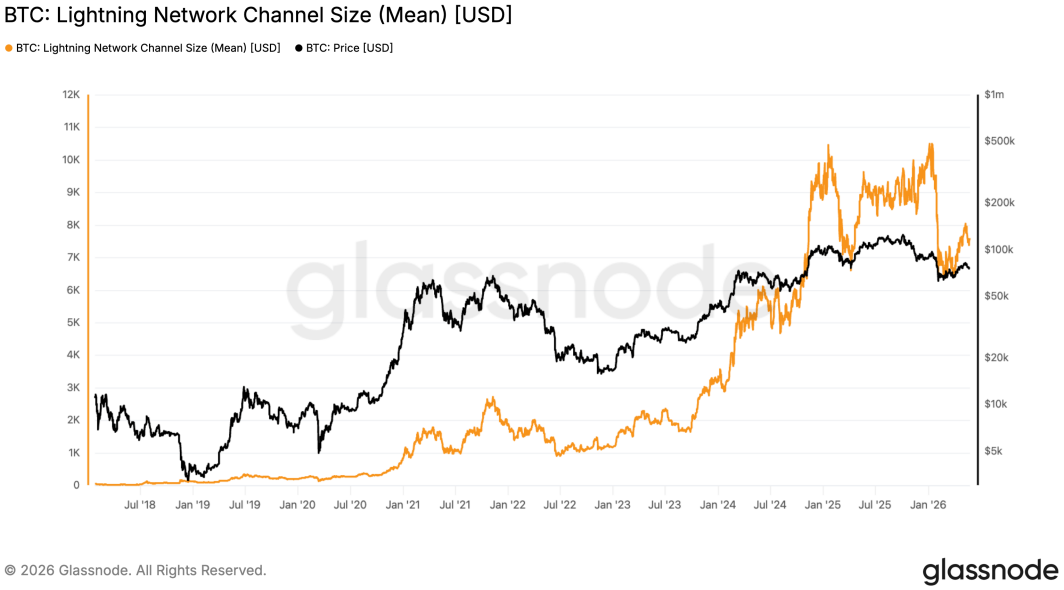
The average degree is computed as the mean number of channels associated with each node.

$$k_{\text{avg}} = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} a_{ij}$$

The observed value indicates that, on average, each node in the network is connected to more than 6 peers.

By average weighted degree, we mean the average liquidity allocated in the channels associated with each node.

$$k_{\text{avg}}^{w} = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1, j \neq i}^{n} a_{ij}^{w}$$



© 2026 Glassnode. All Rights Reserved.

Since channel balance, the allocation of liquidity between the two counterparts of a payment channel, is generally not publicly observable (it is known only to the participating nodes and can, at most, be inferred probabilistically by adjacent nodes when payment routing is enabled), it is not possible to reliably distinguish between average weighted *in-degree* and average weighted *out-degree*.

Network diameter

The diameter of a graph is defined as the maximum, taken over all pairs of vertices in the graph, of the distance between the two vertices considered.

The distance between two vertices is defined as the length of a shortest path connecting them, i.e., the minimum number of edges that must be traversed to go from one vertex to the other.

$$d_{ij} = \min \{ \sum_{k,l \in \mathcal{P}_{ij}} a_{kl} \}$$

where $\mathcal{P}_{ij}$ is the path connecting nodes i and j.

Custering coefficient

The clustering coefficient of a vertex quantifies the density of connections within its local neighborhood, and is defined as the ratio between the number of pairs of vertices adjacent to the vertex that are actually connected to each other and the maximum number of possible links among such neighbors. Within the adjacency-matrix-based formalism, it is possible to express the clustering coefficient of a node:

$$C_i = \frac{2}{k_i \left(k_i - 1\right)} \sum_{j,k} a_{ij} a_{ik} a_{jk}$$

Assortativity

The assortativity coefficient r is a particular case of Pearson's correlation coefficient and quantifies the correlation between the degrees of connected nodes.

Its computation is based on a correlation function that is appropriately normalized with respect to the variance.

$$\langle k_1 k_2 \rangle - \langle k_1 \rangle \langle k_2 \rangle = \sum_{k_1,k_2} k_1 k_2 \left(P(k_1, k_2) - P(k_1) P(k_2) \right)$$

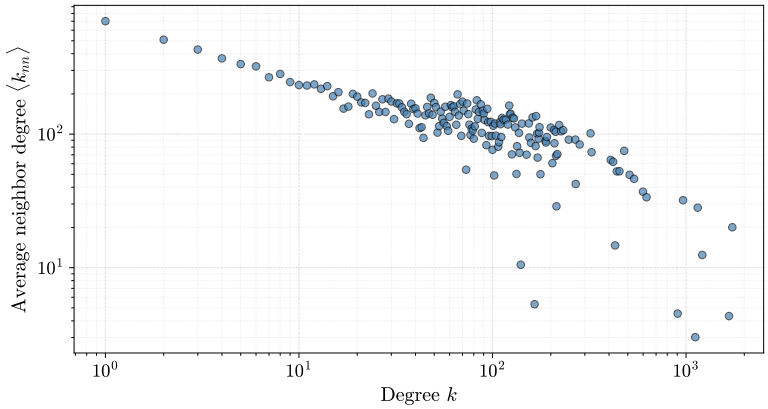
where $P(k)$ denotes the probability distribution of node degrees: when selecting a node uniformly at random, the probability that it has degree k equals $P(k)$. If, instead, one selects an edge of the graph uniformly at random, the probability that this edge is incident to a node of degree k is proportional to $kP(k)$, since higher-degree nodes have a larger number of incident edges and, consequently, a higher probability of being reached by drawing an edge.

$$\sigma^2 = \sum_k k^2 P(k) - \left(\sum_k k P(k) \right)^2$$

$$r = \frac{1}{\sigma^2} \sum_{k_1,k_2} k_1 k_2 \left(P(k_1, k_2) - P(k_1) P(k_2) \right)$$

The computed r value is negative; consequently, the Lightning Network exhibits a *disassortative* structure, consistent with what has been observed in other technological networks such as the Internet and the World Wide Web. [10]

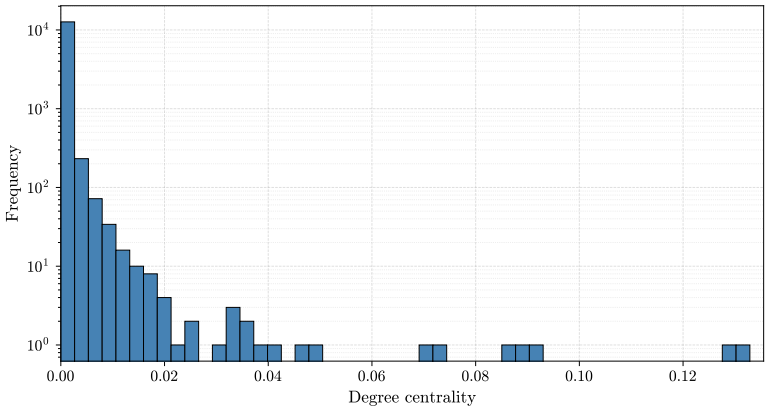
A complementary and more readily interpretable approach to highlighting the network's disassortative nature is to analyze the profile of the neighbors' average degree as a function of node degree.



From the plot, a decreasing trend emerges: as a node's degree increases, the average degree of its neighbors decreases, a pattern consistent with a disassortative structure.

Degree distribution

By representing the frequency of node degrees in the network as a histogram, one observes that the degree distribution is consistent with the typical topology of *scale-free* networks, [10] characterized by a *fat-tail* distribution: most nodes have a low degree, whereas a limited fraction of nodes exhibits very high degrees, with a slowly decaying curve.



In particular, the degree distribution function $P(k)$ can be approximated by a power law.

$$P(k) \propto k^{-\gamma}$$

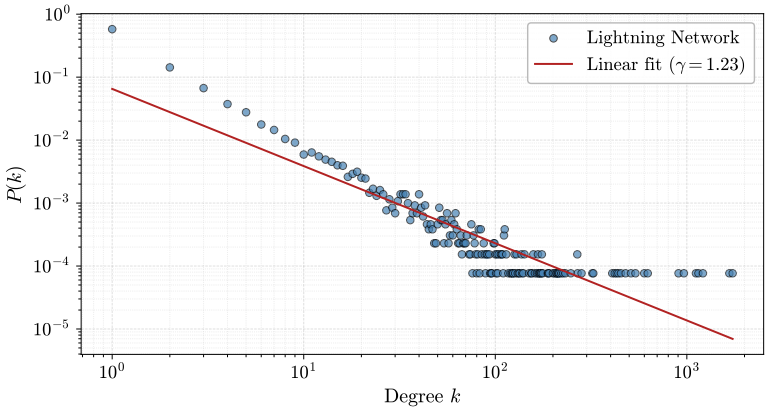
This evidence suggests that, in the Lightning Network, there are a few nodes with a very large number of channels, which can be interpreted as network hubs capable of exerting a significant influence on the connectivity and overall dynamics of the system.

As discussed in the following sections, this centrality may constitute a potential factor of vulnerability for the network. A malicious actor intending to carry out an attack, for instance a *denial-of-service* (DoS) attack, could concentrate efforts on the most highly connected nodes, thereby maximizing the impact on overall connectivity and, consequently, on system performance.

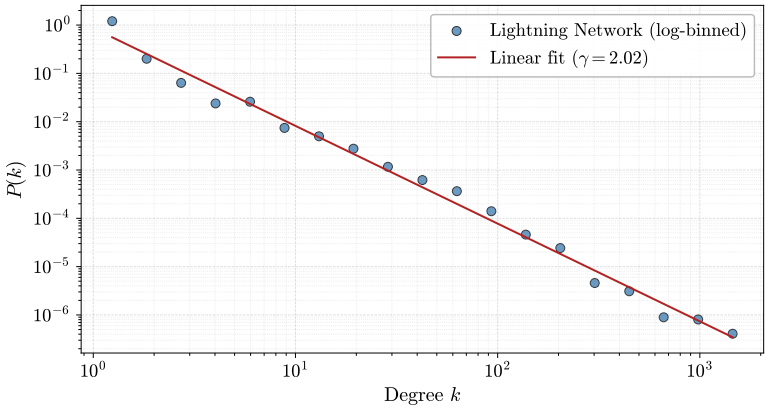
A similar dynamic is instead significantly less plausible in a genuinely distributed network such as Bitcoin, where the degree of decentralization, together with the robustness of the consensus mechanism and the way transactions are recorded, makes large-scale DDoS attacks extremely costly and, in practice, difficult to implement.

We proceed to estimate the exponent of the power-law distribution using a log-log representation.

$$P(k) = k^{-\gamma} \rightarrow \log P(k) = -\gamma \log k$$



The tail of the distribution is often highly noisy and may introduce substantial bias in the parameter estimate. To mitigate the impact of statistical fluctuations, we adopt a binning procedure in which the values along the *x-axis* are grouped into intervals (bins) and the observations within each interval are averaged. To further enhance the robustness of the analysis, we use bins with increasing width following an exponential progression, so as to maintain a comparable number of samples in each bin.



In a double-logarithmic scale plot, estimating the exponent γ reduces to determining the slope of the line that approximates the experimental data. To this end, a linear interpolation is performed using the least-squares method (a first-degree polynomial fit), which provides an estimate of the slope of the log–log relationship.

$$\begin{pmatrix} q \\ m \end{pmatrix} = \arg\min_{m,q} \sum_{i=1}^{n} \left(y_i - (mx_i + q)\right)^2$$

The exponent γ of the power-law distribution corresponds to the negative of

the slope coefficient m estimated via linear regression, according to the relation $\gamma = m$.

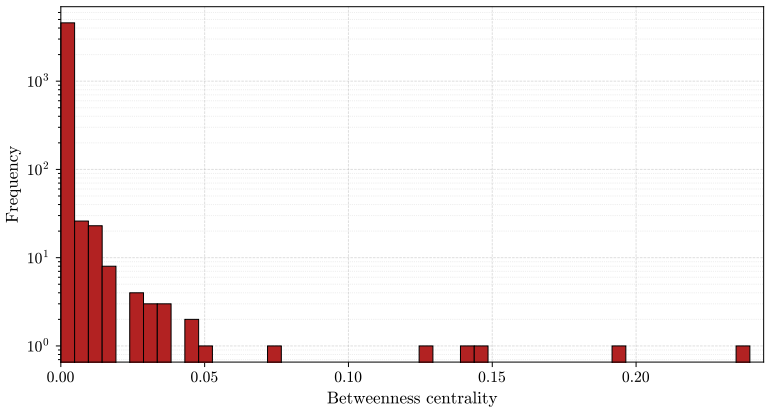
In the case considered, $\gamma = 2.0231$, a value consistent with the hypothesis of scale-free behavior typical of complex systems, for which γ generally lies between 2 and 3.

Similarly, it is possible to analyze the distribution of nodes' *betweenness centrality*. Betweenness is a *non-local* measure that quantifies a node's importance in terms of brokerage, namely the frequency with which a node lies on shortest paths between pairs of nodes and, intuitively, how many times it is traversed during graph navigation.

$$b(i) = \sum_{j,l=1,n} \frac{\mathcal{D}_{jl}(i)}{\mathcal{D}_{jl}}$$

where $i \neq j \neq l$, $\mathcal{D}_{jl}$ denotes the number of paths that start at j and go to l, and $\mathcal{D}_{jl}(i)$ denotes the subset of those paths that pass through node i.

Since, in large-scale graphs, computing betweenness centrality can be computationally expensive, its estimation can be accelerated by considering a subset of nodes rather than the entire population. This strategy significantly reduces the number of required operations and makes it possible to obtain an approximate measure that is generally close to the actual value.



Analogously to the distribution of degree centrality, betweenness centrality also exhibits a distribution consistent with a power-law trend; the exponent of the power law γ_b can be estimated via linear regression on data transformed to a logarithmic scale.

Betweenness centrality is generally correlated with degree centrality (fig. 4.2.2): as degree increases, betweenness also tends to increase, indicating nodes that are potentially more influential in the network structure. In the case considered, the

Pearson correlation coefficient [6] between degree and betweenness centrality across nodes is 0.8506, a value that indicates a strong positive correlation between the two measures.

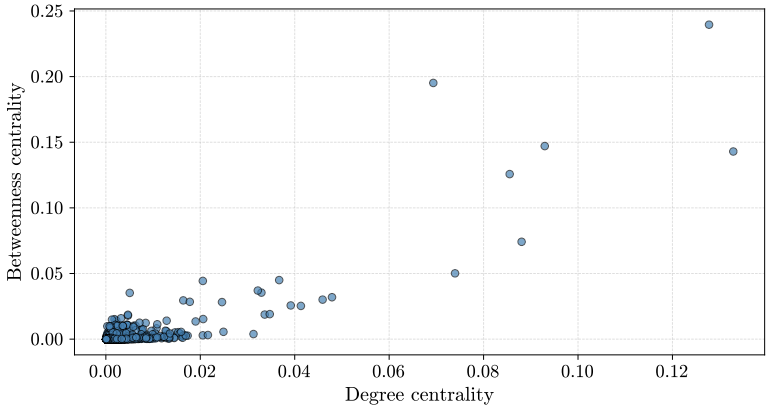


Figure 4.1: Representing betweenness centrality and degree centrality through a scatter plot highlights a strong positive correlation between the two indicators, suggesting that the most highly connected nodes also tend to occupy more prominent positions along the network's intermediary paths.

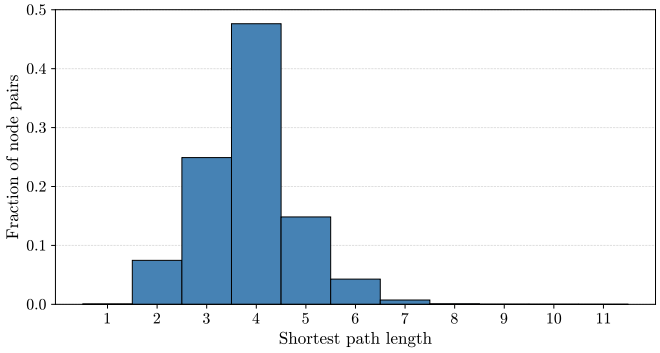
In the context of the Lightning Network, betweenness centrality is particularly relevant because it proportionally represents the potential volume of payments that can flow through specific nodes; this translates into the need to allocate more liquidity to channels and into the possibility of obtaining higher revenues through the collection of routing fees.

Distances distribution

A further relevant aspect concerns the distribution of geodesic distances between pairs of nodes in the graph. In particular, it can be observed that this distribution exhibits a pronounced peak around the value 4, a pattern that is typical of many complex networks. In numerous network models, the average distance between nodes grows in an approximately logarithmic manner with the total number of nodes N. [33]

$$d_{\text{avg}} \sim \log(N)$$

This phenomenon, known as the *small-world effect*, describes a system's ability to maintain short paths between nodes despite the large size of the network, thanks to the presence of a relatively limited number of connections.



In the context of the Lightning Network, this property has significant operational relevance: the possibility of reaching a very large fraction of nodes through a limited number of hops suggests that payment routing can take place along short paths. Since routing costs tend to increase with path length, due to the accumulation of fees (particularly base fees) at each hop, longer paths would be more costly and could increase the likelihood of failures or delays. The typically short distances therefore help make routed payments, overall, faster and more economically efficient.

4.2.3 Graph visualization

For the graphical representation of the network, SFDP (Scalable Force-Directed Placement) [18] was adopted, a force-directed layout algorithm particularly suited to the visualization of large-scale graphs. The approach is based on a physical model in which the arrangement of nodes emerges from the equilibrium between repulsive and attractive forces, enabling complex structures to be handled efficiently while preserving an overall coherent visual rendering. In this model, nodes are treated as particles carrying the same electric charge and are therefore subject to mutual repulsive forces, whereas edges are modeled as elastic springs that exert an attractive force between connected nodes.

In addition, the visualization highlights the degree of each node, defined as the number of node channels, through a color map.

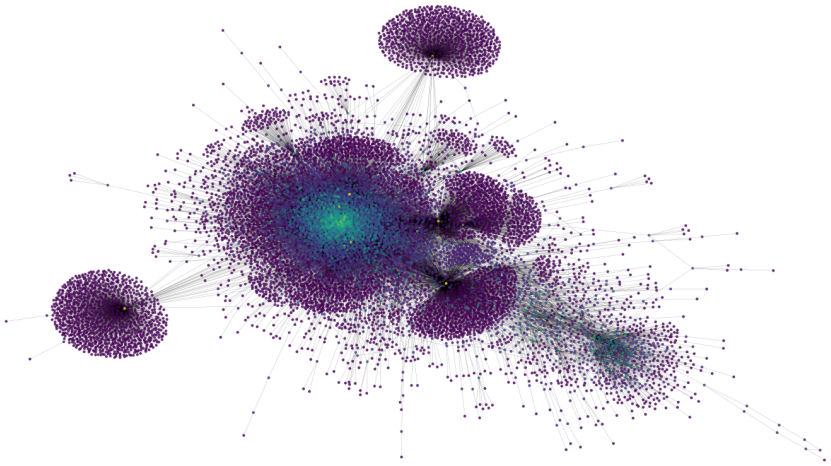


Figure 4.2: A graphical representation of a snapshot of the Lightning Network as of 5 March 2026, using an *SFDP* layout and a node degree based color map.

4.2.4 Clustering and communities

A further element that is typical of graph analysis is the presence of communities, understood as sets of nodes that exhibit a higher density of internal connections than that observed with the rest of the network.

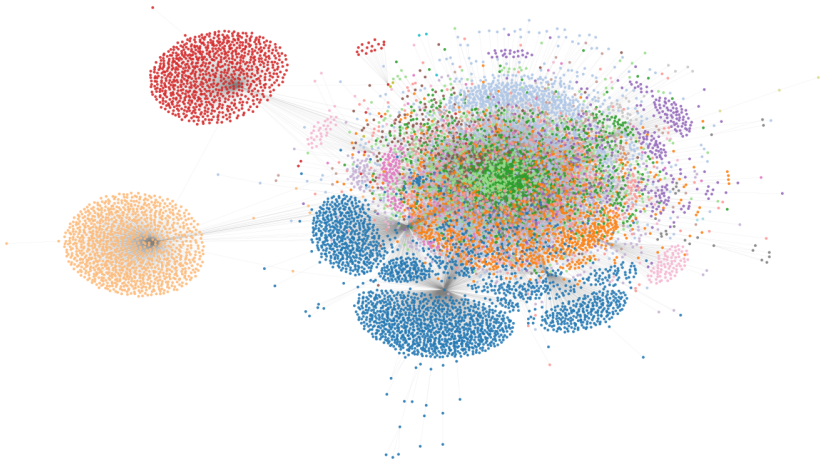
Community identification can be carried out according to two main families of approaches:

- *agglomerative (bottom-up) methods*, which progressively build groups starting from individual nodes or elementary clusters on the basis of shared characteristics and common connectivity patterns;
- *divisive (top-down) methods*, which recursively partition the graph by removing nodes or edges, until relatively isolated and coherent subnetworks (subgraphs) are obtained.

Girvan Newman algorithm

An example of a divisive approach to identifying communities in a network is the Girvan–Newman algorithm [15], which iteratively removes edges according to their betweenness. In particular, betweenness centrality quantifies, for a given edge or node, the fraction of shortest paths between pairs of vertices that pass through it;

it therefore provides a non-local measure of the intermediary role in the graph's global connectivity.



The recursive removal of edges with the highest betweenness tends to eliminate connections that act as "bridges" between densely connected subsets, progressively fragmenting the network into connected components that can be interpreted as communities.

The evolution of the splitting process can be represented by a *dendrogram*, i.e., a tree diagram that describes the hierarchical sequence of partitions and the groupings of nodes produced at the different levels of edge removal.

The computational complexity of the Girvan–Newman algorithm is high, namely $O(m^2 \cdot n)$, where m denotes the number of edges in the graph and n the number of nodes.

This estimate is obtained by analyzing the operations performed at each iteration: the algorithm recomputes edge betweenness centrality, an operation that requires $O(m \cdot n)$. In the worst case, this recomputation is repeated for a number of iterations proportional to m, corresponding to the progressive removal of edges.

Consequently, although the cost increases particularly sharply for dense graphs (large m, close to the theoretical limit), applying the method to the Lightning Network remains impractical: while the network is not strongly dense, its scale makes the approach computationally expensive. For this reason, the Louvain algorithm was adopted for community detection.

Louvain algorithm

The Louvain algorithm [8] is a heuristic method for community detection in graphs, based on maximizing the metric known as modularity (Q). Developed to overcome the computational limitations of traditional divisive methods, it adopts an agglomerative and hierarchical approach, operating through the cyclical repetition of two distinct phases until a local maximum of modularity is reached.

In the first phase, the algorithm initially assigns each node in the network to a distinct community. Then, for each node i, it evaluates the impact, expressed as a modularity gain (ΔQ) of moving the node from its current community to that of one of its neighboring nodes. The node is reassigned to the community that maximizes this increase, provided that the gain is strictly positive; otherwise, the node retains its original affiliation. This procedure is performed sequentially over all nodes and iterated cyclically until no further increase in overall modularity can be achieved.

The second phase consists of constructing a new coarse-grained (macroscopic) graph. The communities identified in the previous phase are collapsed into single "super-nodes." Edges within the same community are transformed into a self-loop on the corresponding super-node, with a weight equal to the sum of the weights of the original edges. Inter-community edges are instead aggregated into a single weighted edge connecting the corresponding super-nodes. Once this contracted network is generated, the algorithm repeats the first phase on it, iterating the entire cycle until the graph structure remains unchanged and modularity has stabilized.

From the standpoint of computational complexity, the Louvain algorithm does not have a strictly defined theoretical bound, since its runtime depends on the specific topology of the network. However, empirical evidence shows that the average complexity can be estimated as $O(n \cdot \log n)$, where n is the number of nodes, and it tends toward linear behavior $O(m)$ on sparse graphs (with m edges). This efficiency is ensured by the strictly local nature of the computations in the first phase and by the drastic reduction in graph dimensionality at each subsequent macro-iteration, making the algorithm applicable to large-scale complex networks.

Chapter 5

Simulations and optimization

By leveraging the Lightning Network graph topology, it is possible to run simulations aimed at evaluating network performance. This approach makes it possible to test and quantify the impact of different optimization strategies, intended both to increase the network's resilience to node removal and to improve the throughput of successfully completed payments.

In particular, the project considers two main directions of analysis and optimization.

- The first focuses on network topology and resilience improvement, by studying the arrangement of payment channels among nodes and identifying interventions that can reduce vulnerability to the shutdown of central nodes. In this context, the automatic or semi-automatic opening of new payment channels, selected strategically, can help strengthen overall connectivity and robustness.
- The second direction concerns increasing transaction throughput by optimizing liquidity in the channels managed by nodes, using liquidity rebalancing strategies.

5.1 Robustness analysis

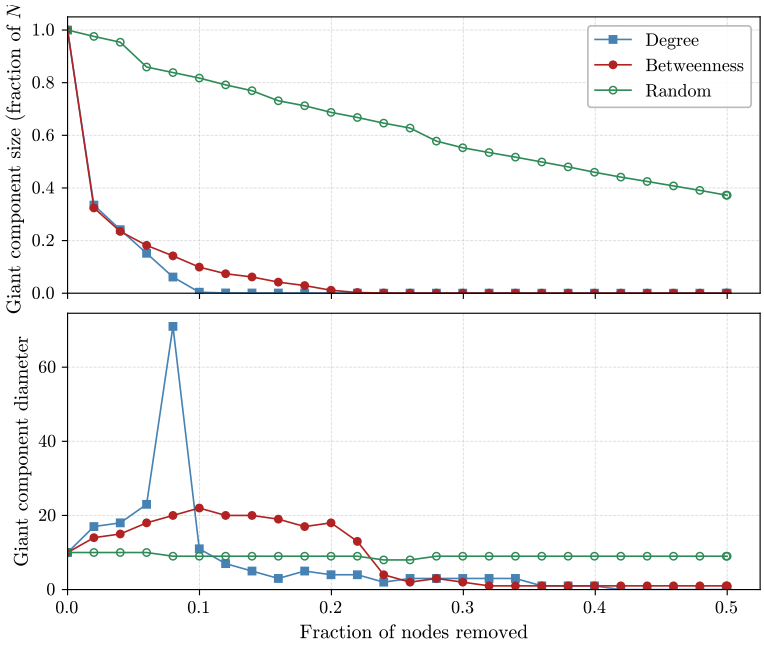
A particularly relevant line of analysis concerns the study of the network's resilience to attacks and the fault tolerance of nodes. This analysis can be traced back to two main scenarios: [1]

- random failures, in which a node is removed at random together with its associated channels;

- targeted attacks, in which the nodes selected as targets are those deemed most "influential" according to centrality metrics such as degree centrality or betweenness centrality.

In the case of random removals, high resilience is generally observed, with effects that are mostly localized around the affected nodes and with limited global repercussions. By contrast, a targeted attack against high-centrality nodes may, in the early stages, produce widespread performance degradation and, in some cases, lead to a partial disconnection of the network.

The main quantities to monitor are the network diameter D and the size of the largest connected component, as a function of the fraction f of nodes removed.



As highlighted by the experimental results, under random removals these metrics remain nearly unchanged for small values of f, and then increase approximately linearly as the share of removed nodes grows.

In the case of targeted attacks, the behavior is markedly different: for low percentages of removed nodes, a rapid reduction of the main component and an initial increase in the diameter are observed. The latter effect can be interpreted positively, as it suggests that even in the absence of highly influential routing nodes, the network may be able to find alternative paths to execute payments. However, this capability is typically associated with an increase in fees, due to a higher average number of hops required to route a payment.

It should be emphasized that this analysis does not take into account the capacity of the removed channels (liquidity): in large routing nodes, channels often

have capacities that are significantly higher than those associated with peripheral nodes. Consequently, the mere existence of alternative paths does not guarantee payment success.

As the fraction of removed nodes increases, simulating an organized, large-scale attack that systematically targets the largest nodes, a pronounced decrease in the diameter is finally observed after the initial increase. This phenomenon is attributable to the progressive fragmentation of the network, with the formation of subnetworks within which payments can still be executed, while global-level payments become difficult and characterized by a high probability of failure.

A substantial difference is also observed between attacks that select nodes based on degree centrality and those based on betweenness centrality: in the former case, a faster reduction in graph size and a more pronounced peak in diameter are recorded in the early stages, whereas in the latter case the effects of fragmentation are delayed and emerge at higher removal percentages.

5.1.1 Graph generation

An extension of interest is to repeat the same node-removal simulation on synthetic graphs that preserve the main properties observed in the analyzed graph, while exhibiting different topologies. To this end, one can rely on mathematical network-generation models capable of reproducing structural features typical of real-world graphs, such as scale invariance and power-law degree distributions.

In this context, the Barabási-Albert (BA) model is particularly appropriate for approximating the scale invariance of the Lightning Network graph. The BA model is based on mechanisms of network growth and preferential attachment, which are consistent with the dynamic nature of the LN, where new nodes progressively enter the network and open payment channels with a probability strongly influenced by centrality measures (and, more generally, by the topological importance) of the nodes already present. Operationally, new nodes tend to route their liquidity towards already influential nodes, maximizing reachability and thus the ability to send and receive payments to and from most nodes in the network. This dynamic helps explain the emergence of large hubs, i.e., high-degree nodes connected to many peripheral nodes with one or few connections.

Synthetic generation under the BA model can be described iteratively as follows. One starts from an initial set of unconnected nodes; at each step, a new node is added and connected to m_0 existing nodes. The probability that the new node connects to an existing node i is proportional to the degree k_i of that node.

The attachment probability $\Pi(k_i)$ is given by the following expression:

$$\Pi(k_i) = \frac{k_i}{\sum_{j=1}^{n} k_j}$$

The number of nodes n and edges m of the graph at iteration t is described by:

$$n = n_0 + t$$

$$m = \frac{1}{2}\sum_{j=1}^{n} k_j = m_0 \cdot t$$

This procedure leads to the formation of a scale-free graph, in which the degree distribution follows a power law.

$$\frac{\partial k_i}{\partial t} = A \cdot \Pi(k_i) = A \cdot \frac{k_i}{\sum_{j=1}^{n} k_j} = \frac{A k_i}{2 m_0 t}$$

Since $A = m_0$ and at initial time t_i the initial degree is $k_i(t) = m_0$

$$\frac{\partial k_i}{\partial t} = \frac{k_i}{2t} \rightarrow k_i(t) = m_0\sqrt{\frac{t}{t_i}}$$

To derive the exponent of the power law from this relation, it is sufficient to consider the probability that a vertex has degree less than k.

$$P(k_i < k) = P\left(t_i > \frac{m_0^2 t}{k^2}\right)$$

The distribution $P(t) = A$ is uniform because the nodes grow linearly with time (one at each iteration), so A can be obtained as the normalization constant of the uniform distribution.

$$A = \frac{1}{n} = \frac{1}{n_0 + t}$$

$$P\left(t_i > \frac{m_0^2 t}{k^2}\right) = 1 - P\left(t_i \leq \frac{m_0^2 t}{k^2}\right) = 1 - \frac{m_0^2 t}{k^2}\frac{1}{n_0 + t}$$

$$P(k) = \frac{\partial P(k_i > k)}{\partial k} = \frac{2 m_0^2 t}{k^3 (n_0 + t)}$$

From this, we can see that the distribution follows a power law with exponent 3.

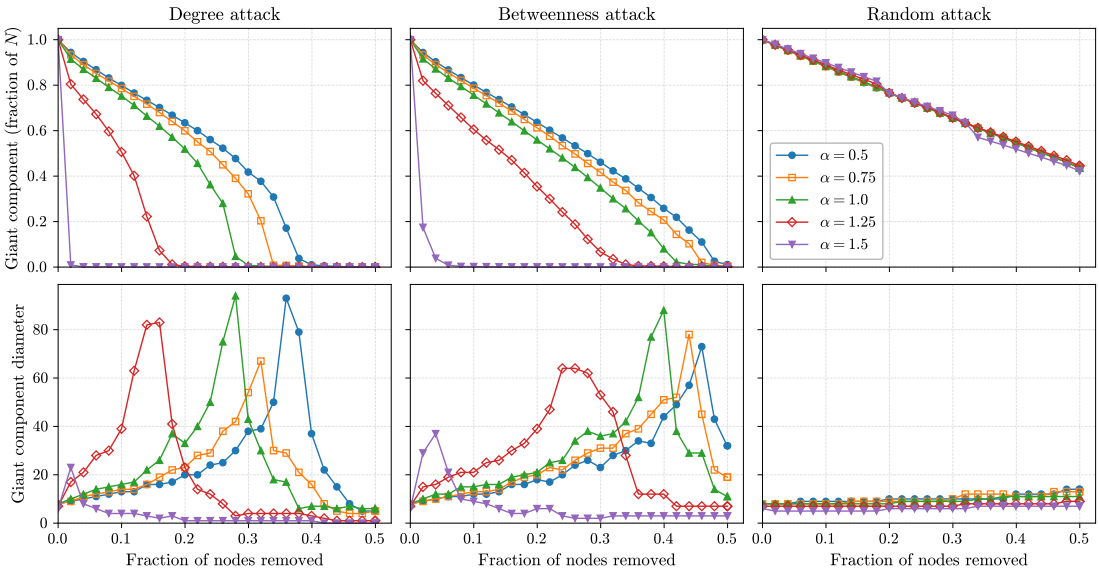
To more faithfully reproduce the behavior of the Lightning Network and to modulate the properties of the generated graphs, some extensions of the model are required.

A first modification consists in introducing a fitness parameter η_i, which alters the attachment probability and makes it possible to control the level of network centralization. This control enables the simulation of attacks on more or less centralized topologies and a comparative assessment of their resilience.

$$\Pi(k_i, \eta_i) = \frac{\eta_i k_i}{\sum_{j=1}^{n} \eta_j k_j}$$

Instead of acting on the probability associated with each node through the parameter η_i, the simulation modified the preferential attachment function, making it of an order different from the linear one. In particular, by introducing a dependence on node connectivity of the form k^α, it is possible to obtain a superlinear regime ($\alpha < 1$) or a sublinear one ($0 < \alpha < 1$). In the sublinear case, the preference for creating links toward nodes that are already highly connected is attenuated, leading to a reduction in the degree of network centralization.

A second modification involves varying the number of edges associated with each new node (m_0), so as to obtain a heterogeneous population of nodes: on the one hand, small nodes with a single payment channel, and on the other hand, larger nodes with multiple channels, consistent with what can be observed in the analysis of the original network. The proportion between nodes with a single payment channel and the other nodes can be adjusted via a parameter that controls the percentage fraction.

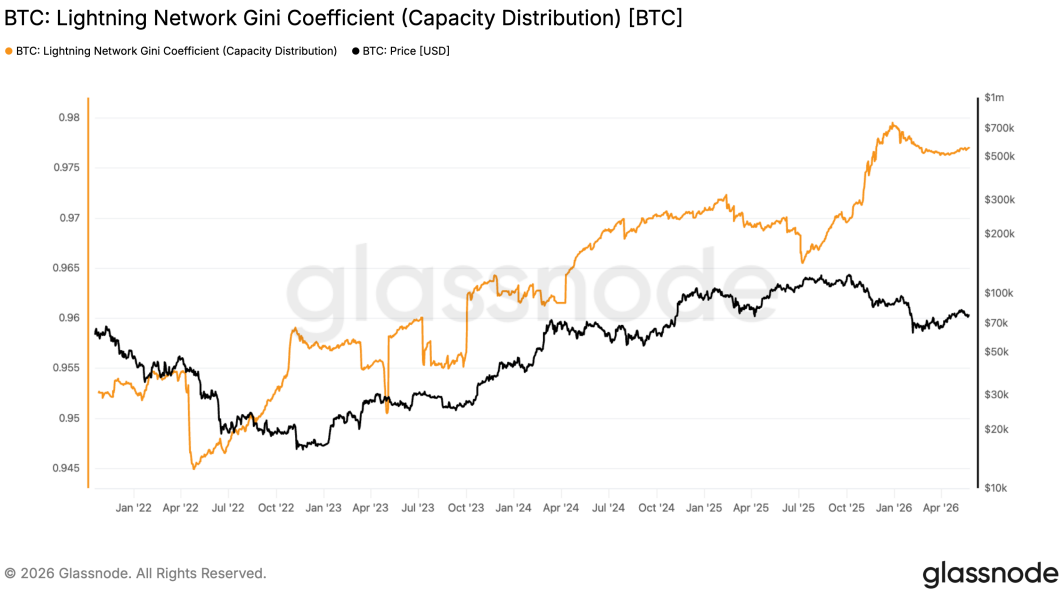


Simulation results indicate that, as the parameter α varies, and thus as the degree of network centralization changes, a delay emerges in the fragmentation process of the network's main connected component. This evidence provides empirical support for the claim that a more efficient configuration of payment channels can significantly increase the resilience of the Lightning Network against targeted attacks, while simultaneously preserving its performance even in the presence of failures or unavailability of nodes with high topological importance.

5.1.2 Robustness optimization

As a topology optimization strategy designed to enhance network resilience against centrality-based attacks, a method to be applied at the inception of a new payment channel between two nodes is proposed. Except for pragmatic operational constraints influencing peer selection, such as empirical traffic forecasts or specific technical considerations, nodes generally prefer to connect with high-degree counterparts to maximize the probability of routing payments globally. This behavioral pattern, however, precipitates an inevitable centralization of the network topology via a preferential attachment paradigm, wherein nodes possessing superior degree and liquidity continually accumulate new channels. This centralization trend is further exacerbated by automated liquidity management algorithms, such as *autopilot* [26], a Python-implemented recommendation engine for channel creation, which utilizes node degree or structurally similar metrics as heuristics, even though customizable filters can be deployed to restrain such phenomena.

Confirming the inherent trend of network topology toward centralization, a new snapshot taken on September 12, 2026, recorded metrics comparable to the March 30 snapshot in terms of node and channel counts, subject only to minor variations. However, as illustrated in the graph 5.1.2, the overarching temporal trend demonstrates increasing centralization in liquidity distribution, favoring the emergence of major hubs that control the vast majority of network channels.



© 2026 Glassnode. All Rights Reserved.

glassnode

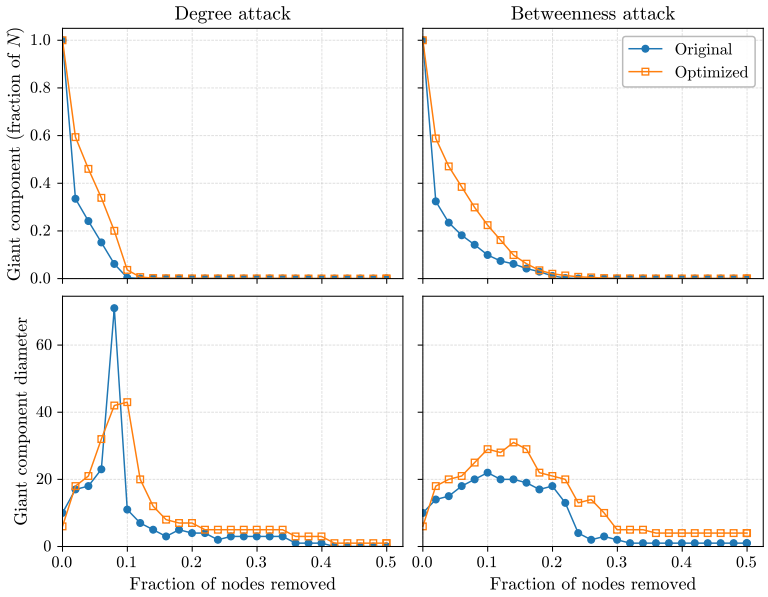
Specifically, the graph depicts the evolution of the Gini coefficient as applied

to channel distribution. The Gini coefficient is a standard economic metric used to measure distribution inequality, ranging from 0 to 1; values close to 0 indicate a homogeneous distribution, whereas values approaching 1 (such as those observed in the graph's most recent period) indicate high inequality.

The steadily upward trajectory of the Gini coefficient underscores the network's spontaneous drive toward centralization to optimize payment efficiency for newly joining nodes.

The methodology investigated via simulation in this work mitigates these structural vulnerabilities by optimizing channel selection to maximize the algebraic connectivity of the graph, which serves as a global measure of network cohesiveness. A graph exhibiting low algebraic connectivity is typically characterized by distinct communities linked exclusively by large hubs, rendering these central nodes critical bridges between isolated subgraphs; consequently, the failure or removal of such a hub triggers network fragmentation. Conversely, a network with elevated algebraic connectivity demonstrates superior resilience against centrality-targeted attacks, as alternative pathways ensure robust interconnectivity without relying heavily on high-betweenness nodes.

Operating on a node-by-node basis to identify the ideal target for a new channel, the proposed approach utilizes the Fiedler vector to select the node with the maximum spectral distance from the initiating node. Mathematically, the computation initiates from the graph Laplacian matrix, defined as $L = D - A$, where D is the degree matrix and A is the graph adjacency matrix. The second smallest eigenvalue λ_2 of the Laplacian matrix denotes the algebraic connectivity of the network, and its corresponding eigenvector is designated as the Fiedler vector. To identify the optimal peer j for an initiating node i, the algorithm selects the node that maximizes the squared difference of their Fiedler vector components, expressed as $(v_i - v_j)^2$. Conceptually, this selects nodes positioned within topologically distant communities; bridging these nodes effectively maximizes the global connectivity and structural robustness of the entire network.



Subsequently, a comparative robustness analysis was conducted to evaluate the network response under targeted centrality attacks, specifically leveraging degree and betweenness centrality metrics. This evaluation contrasted the structural integrity of the empirical Lightning Network topology against the optimized graph topology, which was generated by allowing each node to establish one additional channel. To mitigate the computational overhead associated with calculating the Fiedler vector, and to restrict the target pool to nodes demonstrating a baseline level of connectivity, a k-core decomposition technique was applied. This preprocessing step isolates a core subgraph comprising exclusively nodes with a degree of at least k, thereby optimizing algorithmic efficiency while filtering out peripheral nodes.

As illustrated by the simulation results in Fig. 5.1.2, the proposed optimization framework yields a substantial enhancement in network resilience against centrality-targeted disruptions. This improvement is particularly pronounced during the initial fractions of node removal, which represents the most plausible and critical threat scenario in real-world networks. Regarding the relative size of the giant component, the optimization effectively halves the structural degradation inflicted by the attack during the early stages of the simulation. Furthermore, tracking the diameter of the giant component reveals a systematic delay in topological fragmentation, demonstrating that the optimized network maintains global reachability and cohesive routing pathways significantly longer before undergoing complete disconnection.

5.2 Throughput optimization

5.2.1 Introduction

The most common error during the payment routing process does not stem from the lack of available paths, since the network is generally well connected, but rather from liquidity imbalances in channels. In particular, when liquidity is concentrated mainly in one direction of a channel, executing a payment may become impossible due to insufficient capacity in the required direction.

To mitigate this issue, nodes can adopt channel rebalancing procedures, carried out either in a *just-in-time* manner or on a periodic basis, with the aim of increasing the number of successfully routed payments. An increase in the volume of forwarded transactions translates, for routing nodes, into higher fee collection and thus greater revenues associated with node operation and the liquidity they provide. However, rebalancing operations can entail non-negligible costs, which must be taken into account when assessing overall profitability. From an economic perspective, the rebalancing costs of the various channels should not exceed the additional fees that can potentially be obtained, in order to keep rebalancing activity sustainable and incentive-compatible.

The following section presents the main rebalancing strategies analyzed and simulated in this work.

5.2.2 Rebalancing strategies

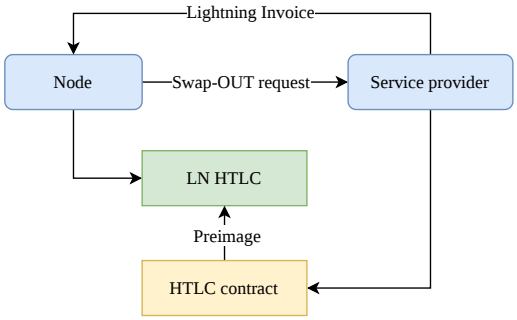
Submarine swap

A submarine swap enables the exchange of assets between the on-chain and off-chain domains, allowing transfers between the Bitcoin blockchain and the Lightning Network without exposure to counterparty risk. This operation typically requires the involvement of a third party that provides the service (e.g., a liquidity provider), compensated through a fee or spread. Conceptually, the submarine swap constitutes a particular case of an atomic swap: the outcome is binary, in the sense that the procedure is either completed in full or fails, without allowing either party to appropriate the other's funds.

The mechanism relies on HTLC (Hashed Time-Locked Contract) contracts, the same cryptographic primitives used in Lightning Network payment channels. In a typical instance, the recipient of a Lightning payment generates a random 32-byte secret, called *preimage*, and computes its hash using a one-way cryptographic function. Given the one-wayness property, it is computationally straightforward to verify that a hash value corresponds to a specific preimage only when the preimage is known; in the absence of such information, deriving the preimage from the hash

is considered computationally infeasible.

The hash of the preimage is embedded in the script of the on-chain contract that governs the release of funds. The same preimage is also used to construct the Lightning invoice, whose operation is natively supported by the protocol. Upon payment of the invoice, the payer obtains the preimage required to redeem the funds locked on-chain and must exercise this right within the time limit imposed by the HTLC.



This architecture provides security properties for both parties, preventing any single counterparty from unilaterally seizing the funds. Compared to circular rebalancing, a submarine swap entails an on-chain transaction, potentially associated with higher fees than a rebalancing performed exclusively on Lightning. However, the technique is more flexible, as it allows channel liquidity to be adjusted both upward and downward; moreover, it can be used to transfer funds from a Lightning channel to on-chain storage (e.g., cold storage), making it useful in scenarios where inbound payments are received frequently, such as in e-commerce activities.

In the simulation, the use of submarine swaps as a rebalancing strategy was modeled, and the resulting reduction in the share of failed payments was quantified. It is, however, appropriate to make explicit some methodological considerations. In this analysis, for simplicity, the constraint of conserving the capital allocated to each Lightning node was not enforced strictly, since a submarine swap entails a reallocation of liquidity between the on-chain and off-chain domains. Nevertheless, when nodes have a large number of channels, the total amount withdrawn and the amount reintroduced into the Lightning Network tend to offset each other on average, due to the statistical aggregation of operations.

Alternatively, liquidity conservation can be imposed explicitly by rebalancing pairs of channels: liquidity is withdrawn from one channel and the same liquidity is used to increase the available capacity of another channel within the node. This approach, while satisfying the constraint, introduces two cost factors: increased operational latency, due to the waiting times required for blockchain block confirmations, and higher fees, typically about twice those of rebalancing a single

channel, since the operation requires two distinct transfers.

With regard to estimating fees, it is advisable to prepare a forecasting model that integrates: Lightning Network fees, on-chain transaction costs, and the mark-up applied by the service provider delivering the service. Considering the scenario in which funds are transferred from Lightning to on-chain for the purpose of rebalancing inbound liquidity, the Lightning node performing the submarine swap must forward a Lightning payment to the provider's node. The cost of this payment depends on routing conditions and on the fee structure of the traversed channels (fixed and proportional components), and therefore is a function of the transferred amount and the number of hops. On-chain costs include the fees required for block inclusion of the transaction associated with the HTLC smart contract and vary with the level of blockchain congestion. Finally, service-provider fees are generally expressed as a percentage of the amount being converted and are paid in addition to the operational fees borne by the provider to execute on-chain or off-chain payments, depending on the direction of the flow (inbound or outbound).

When a node performs routing activities professionally, or more generally for profit, the operator has an incentive to recover the costs incurred for channel rebalancing through the commissions (fees) generated by subsequent routed payments. To this end, based on a statistical analysis of the traffic traversing the channel, the operator can consistently determine the pricing parameters (specifically the base fee and the fee rate) so as to estimate the investment payback horizon and configure the fee policy accordingly. Moreover, the operator may consider an estimate of the number of payments that, in probabilistic terms, could be unlocked through the rebalancing operation, in order to determine whether to perform rebalancing and, if so, with what optimal frequency.

The following describes the strategy adopted in the simulation for channel rebalancing via submarine swaps. A baseline approach consists of rebalancing channels that exceed a given imbalance threshold in order to restore a symmetric liquidity distribution (50% on each side). Although this heuristic can be broadly effective, it does not account for the statistical asymmetry of payment flows, which in many scenarios makes it optimal to maintain a larger amount of liquidity on a specific side of the channel. To address this limitation, an approach based on a statistical analysis of the payments routed through the channel was implemented, with the goal of estimating the optimal liquidity allocation percentage.

As an illustrative example, consider a node operating as a merchant and therefore predominantly receiving payments. In this configuration, the channel quickly becomes depleted on the receiving side, becoming imbalanced and reducing the capacity to accept further payments (while still retaining the ability to send payments). In this case, it is beneficial to shift liquidity to the opposite side of the channel and convert part of the proceeds into on-chain funds. This can be

achieved via a submarine swap by sending a Lightning payment to the service provider through the channel to be rebalanced, thereby moving liquidity.

The optimal amount P to rebalance is estimated from the observed proportion between incoming and outgoing payments through the channel.

$$P = (r_{out} - f_{out}) \cdot c$$

where $r_{out} = \frac{c_{out}}{c_{out}+c_{in}}$ denotes the ratio between the available outbound liquidity and the channel's total liquidity; $c = c_{out} + c_{in}$ indicates the channel's total liquidity; and $f_{out} = \frac{\lambda_{out}}{\lambda_{out}+\lambda_{in}}$ is the fraction of outbound payments considered in the analysis relative to the total number of payments. The analysis of the payment flow within the channel is conducted using the parameter λ, which can be interpreted as the parameter of a Poisson process modeling the demand for payments through the channel [17]. In particular, λ can be estimated as the product of the number of transactions per unit time and the average payment amount. In the simulation setting, instead, λ is assumed in a simplified form as the sum of the payments analyzed.

Substituting into the previous formula, we obtain:

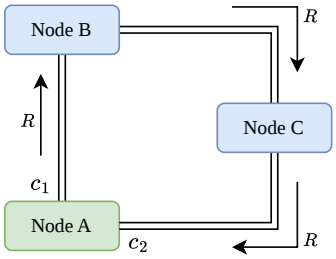
$$P = c \cdot \left(\frac{c_{out}}{c} - \frac{\lambda_{out}}{\lambda} \right) = c_{out} - \frac{\lambda_{out} \cdot c}{\lambda}$$

To prevent rebalancing operations of negligible magnitude, an activation threshold has been introduced, defined as a fraction of the channel's total capacity; below this value, rebalancing is not performed.

Moreover, the simulation approach can be extended to a multi-period dynamic, enabling the optimization to be carried out recurrently over time. In this formulation, payments are aggregated into time epochs, and the traffic observed in the previous epoch (or an average over multiple epochs) is used to periodically update the channel rebalancing decision.

Circular rebalancing

Circular rebalancing is one of the most widely used channel rebalancing techniques and is supported by most Lightning clients. The procedure involves sending a payment from a node to itself along a cyclic route, with the goal of reallocating available capacity by moving liquidity from a channel with high outbound capacity to a channel whose outbound capacity is low or close to depletion. The approach is generally time-efficient, since Lightning Network payments are completed with very low latency, and it entails limited operational costs because it does not require on-chain transactions.



In the simulation, circular rebalancing is employed following a methodology analogous to that adopted in the submarine swap approach, based on analyzing the flow of payments traversing the channels and applying a proportional rebalancing of available liquidity.

Unlike submarine swaps, this technique can be applied to routing nodes that have at least two channels, as it requires an incoming channel and an outgoing one. Performance is superior when it is possible to coordinate the rebalancing of multiple channels: for instance, in the case of two channels that are imbalanced respectively on the inbound side and on the outbound side, a single circular payment makes it possible to mitigate both imbalances simultaneously.

From a cost perspective, circular rebalancing is significantly cheaper than submarine swaps, since it does not require the involvement of a third-party service provider subject to fees and, most importantly, it does not entail on-chain costs, which are typically higher than those associated with a Lightning payment.

In the simulation setting considered, circular rebalancing is performed proactively, enabling periodic re-equilibration of a node's channels. However, thanks to its operational simplicity and rapid execution, the procedure can also be adopted with a *just-in-time* approach: upon receiving a payment to be routed that cannot be completed due to a channel imbalance, the routing node immediately performs a circular rebalance and subsequently proceeds with routing the payment.

Channel splicing

An alternative approach to modifying the capacity of a payment channel is channel splicing, a technique that makes it possible to increase (splice-in) or decrease (splice-out) the channel's liquidity through a single on-chain transaction. In the absence of splicing, changing the channel capacity would typically require two separate transactions: one to close the existing channel and another to open a new one. In addition, splicing allows the sending and receiving of payments to remain operational during the update, without waiting for the on-chain finalization of the commitment transaction; this yields benefits in terms of latency and overall costs.

The basic principle is to spend the output of the transaction that represents the channel state and, at the same time, create a new channel with increased or decreased capacity. In the case of splice-out, a portion of the funds is allocated to an on-chain address of the recipient, thereby reducing the channel capacity. In the case of splice-in, a transaction is instead constructed and signed that injects additional capital into the channel, increasing its capacity beyond the funds already present.

Passive rebalancing

Passive rebalancing is generally a less effective strategy than other approaches, but it allows channels to be rebalanced organically, without resorting to active interventions. This technique is based on adjusting the routing fees associated with channels; fee changes are propagated through the network via the gossip protocol, with the aim of influencing payment path selection.

In particular, routing algorithms tend to route payments through channels with lower fees in order to minimize the overall transaction cost. Consequently, if the goal is to encourage liquidity to flow in a given direction, it is sufficient to reduce fees in the desired direction and increase them in the opposite direction. Over time, the expected redistribution of payments can help rebalance the channel automatically.

The main advantage of this method lies in the absence of direct costs: the only burden is the potential loss of fee revenue in the direction where fees are reduced. This opportunity cost depends on traffic volume, but it is generally one or more orders of magnitude lower than the cost of an on-chain transaction, which is commonly used in other rebalancing techniques. The main limitation is that the process is not immediate; therefore, passive rebalancing is poorly suited to scenarios of severe imbalance that require timely intervention to restore the channel's payment capacity.

The simulation aimed at quantifying the efficiency of passive rebalancing was carried out by defining the initial channel balance parameter. In this context, a value of 0.5 represents an equilibrium condition (50% inbound liquidity and 50% outbound liquidity), whereas lower values indicate progressively higher levels of imbalance.

Subsequently, the base fee and fee rate parameters, which determine routing fees, were set. These fees were defined as a function of the channel's imbalance, with the goal of incentivizing the natural flow of payments in the desired direction and, consequently, promoting rebalancing.

Next, random payments were generated between randomly selected nodes within the network, analogously to the procedure described in the following chapter; the payments were then propagated and executed by applying the updated fee condi-

tions.

Finally, the mean imbalance ratio of the channels involved was computed in order to assess the improvement achieved relative to the initial mean condition.

5.2.3 Generating payments data

To study the network's topological properties and robustness, it is possible to collect real data on its structure, as described in the previous chapters. By contrast, payment data are not publicly available. Moreover, due to the onion routing used by the routing protocol, information about the sender and recipient remains anonymous, making an analysis based on real transactions impracticable. Consequently, it is necessary to synthetically generate a transaction dataset in order to study the network's response.

A natural approach is to adopt a Monte Carlo simulation, generating random payments by selecting a source node, a destination node, and an amount. To represent a workload with an average payment volume, it is appropriate to control the distribution of the generated amounts. In particular, the payment generation module uses a Gamma probability distribution, which makes it possible to set a desired mean value for transaction amounts. The Gamma distribution is described by the following probability density function, defined for $x < 0$.

$$\mathrm{PDF}(x; k, \theta) = \frac{x^{k-1} e^{-\frac{x}{\theta}}}{\theta^k \Gamma(k)}$$

In this context, k denotes the shape parameter and, in the simulation considered, it takes the value $k = 2$. For $k < 1$ the density decreases monotonically (exponential-like behavior), whereas for $k > 1$ it exhibits a maximum and a long right tail that asymptotically tends to zero. The parameter θ is the scale parameter, which determines the stretching of the distribution along the x-axis and, consequently, the location of the peak; in the adopted model it is set consistently with the specified mean amount. Indeed, the expected value of a Gamma random variable equals the product of the shape and scale parameters.

$$E[X] = k \cdot \theta$$

Although the Monte Carlo approach is useful for characterizing the system's general behavior, it does not fully reproduce actual Lightning Network traffic, since some channels tend to be used predominantly in a single direction. This occurs, for example, in the case of e-commerce operators that receive many inbound payments, or Lightning Service Providers that favor outbound transactions. To simulate a dataset that is more representative of a real-world scenario, pairs of nodes (sender and recipient) were generated. A fraction of these pairs, defined through

a configurable parameter, was then used to produce more than one transaction. This choice makes it possible to control the likelihood of observing recurring payments between the same counterparties and, consequently, the repeated use of the same payment channels. In this way, the emergence of unbalanced channel configurations is emphasized, making the imbalance problem more pronounced and realistic.

5.2.4 Simulation results

The simulation was conducted using real data drawn from a snapshot of the Lightning Network graph and a sequence of synthetic payments generated according to the methodology described earlier; however, the approach is general and also allows, depending on experimental needs, the use of alternative graphs.

To minimize failures attributable to the lack of available paths between specific pairs of nodes, the analysis was restricted to the largest connected component, excluding isolated nodes or those belonging to smaller components. Consequently, the observed failures are primarily attributable to channel imbalance, which is the specific focus of this study.

The experimental campaign includes 10^5 payment attempts with an average amount of $2 \cdot 10^3$ sat and starts from an initial configuration characterized by a parameterizable and variable degree of channel imbalance. The output metric considered is the fraction of failed payments relative to the total.

The following charts compare the simulation results as a function of variations in the input parameters, in particular the initial channel balance ratio and the fraction of recurring payments. This configuration makes it possible to reproduce a biased, rather than fully random, traffic pattern, thereby making the simulated scenario more consistent with realistic operating conditions.

The results considered refer to the percentage of unsuccessful payments out of the total number of generated payments, assessed both in the non-optimized scenario and in the scenario following channel rebalancing via submarine swaps.

In the first chart, the evolution of this metric is reported as a function of the initial channel-imbalance condition; in the second chart, the same analysis is instead carried out as a function of the fraction of recurring payments.

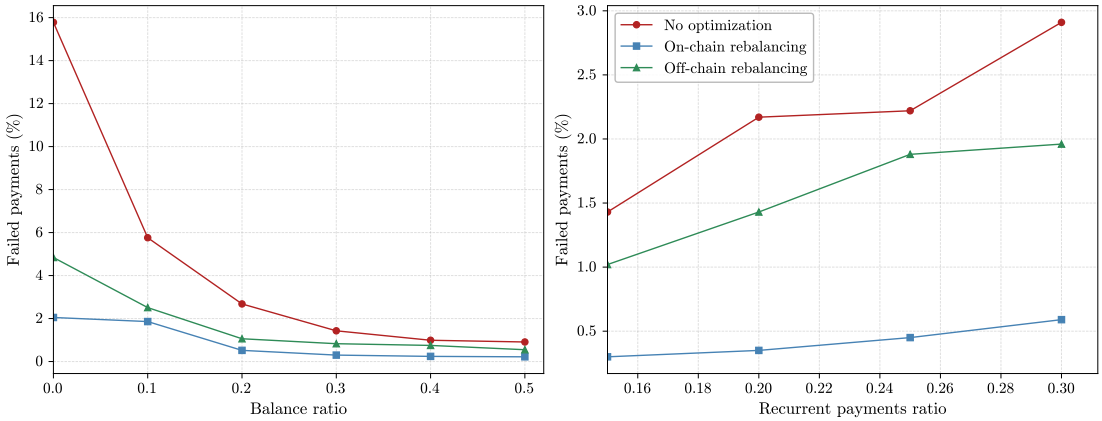


Figure 5.1: Comparison of payment failure rates between the unoptimized baseline and the on-chain (Submarine Swap) and off-chain (circular rebalancing) optimized scenario across varying initial channel balance ratios and recurring payment fractions.

The simulation results indicate an overall improvement in network throughput, quantified as a reduction in the percentage of failed payments due to insufficient liquidity in the channels. The performance gain is more pronounced under initial conditions characterized by greater channel imbalance. Moreover, consistently with expectations, the payment failure rate increases as traffic bias grows. The optimized scenario maintains a performance advantage over the baseline case across all the initial configurations analyzed.

Importantly, rebalancing techniques yield performance gains even when channels are initialized in a perfectly balanced state (i.e., a 50/50 channel liquidity distribution). This occurs because optimization algorithms reallocate channel liquidity to reflect the empirical traffic conditions observed across individual channels. By capturing the statistical behavior of routing requests, these techniques dynamically tailor channel balances to maximize overall transaction throughput.

The second technique examined in the simulation is passive rebalancing, assessed by comparing the channels' initial balance parameter with the value measured after optimizing routing fees and executing the generated payments.

The simulation was conducted by generating 10^5 random payments with an average amount of $5 \cdot 10^4$ sat. This average amount was set to a higher value than in the previous simulation in order to make the effects of rebalancing more clearly observable, including in channels with greater capacity.

It is important to note that the performance of passive rebalancing, achieved via dynamic adjustments to channel routing fees, is highly dependent on the volume and characteristics of the transaction flow routed across the network. In this

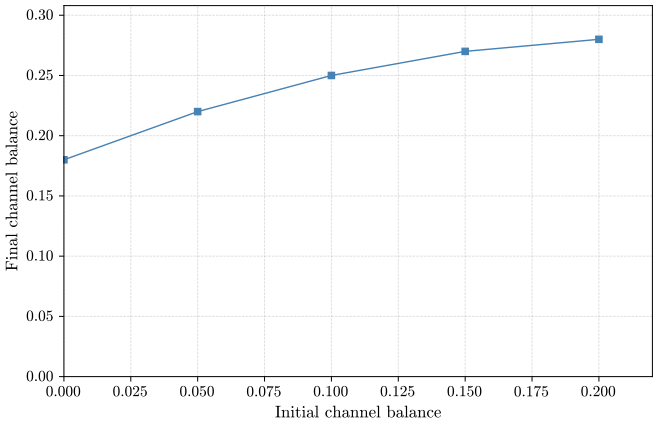


Figure 5.2: Simulation results of passive rebalancing achieved through dynamic node routing fee adjustments designed to incentivize channel rebalancing.

simulation, the operational objective was to restore channels to a symmetric liquidity state (i.e., a 50/50 liquidity distribution on each side of the channel). While this target equilibrium was not achieved across all initial conditions, the strategy yielded substantial improvements in balance distribution, particularly in scenarios with severe initial imbalances.

These outcomes can be optimized further by adapting target liquidity ratios to match the expected traffic profile of each specific channel rather than enforcing a uniform static benchmark. To achieve this, passive rebalancing can be combined with proactive rebalancing techniques, as discussed in the following section.

With regard to rebalancing via circular payments, adopting an approach analogous to that used in submarine swaps, distinct from involving pairs of channels belonging to the same node (one outbound and one inbound), did not yield statistically significant results; under some configurations of the initial parameters, performance even deteriorated. This evidence is plausibly attributable to the fact that rebalancing high-capacity channels can induce an imbalance in the channels involved in the circular payment, leading to a post-optimization final state that, although appearing more balanced in overall terms, is not consistent with the specific traffic pattern for which the rebalancing was performed, thereby reducing transactional throughput.

In light of these considerations, to obtain a rebalancing mechanism that is truly efficient, immediate, and low-cost, such as that offered by circular rebalancing, it would be optimal to introduce a form of coordination among nodes, aimed at executing more appropriate and globally optimal rebalancing payments.

For these reasons, introducing an additional type of signal, exchangeable among nodes via the gossip protocol, aimed at indicating the existence of channels to be

prioritized for routing circular rebalancing operations. If this information is not disseminated globally through gossip, an alternative is to transmit it selectively only to adjacent nodes, or to the main routing nodes that implement this optimization.

Having an explicit indication of preferred channels for routing circular rebalancing payments makes it possible to exploit as efficiently as possible the largest feasible number of circular payments. In particular, a single rebalancing payment could simultaneously contribute to rebalancing multiple channels, in a manner consistent with the liquidity needs communicated by the nodes.

However, this approach must be supported by appropriate economic incentives. Selecting specific paths for rebalancing payments tends, on average, to produce longer routes and, consequently, a higher overall cost for the node that initiates the operation. Reducing routing fees, especially the base fee of the channels involved, can therefore provide an economic incentive to choose the proposed path.

This strategy has already been discussed in the literature and is commonly referred to as coordinated or collaborative rebalancing. [25]

A simulation based on applying this method was also conducted to assess its effectiveness. The results show performance approaching the efficiency of on-chain rebalancing. The chart 5.1 reports the simulation outcomes, compared with the unoptimized baseline and the on-chain rebalancing.

5.3 Related work

In the literature, the Lightning Network has been extensively analyzed, modeled, and optimized through simulation frameworks. Specifically, simulations have been employed to evaluate and refine routing algorithms [28] [35], assess performance gains introduced by Multi-Path Payments (MPP) [23], examine network resilience against various attack vectors, and analyze both active and passive rebalancing strategies. [13] [25]

Unlike existing implementations such as the CLoTH simulator [14], which aim to faithfully replicate payment channel mechanics down to HTLC contract execution, the simulation framework developed in this thesis adopts a high-level modeling approach for routing, payment flows, and rebalancing techniques, prioritizing strategic analysis over protocol-grade fidelity.

Unlike prior simulations in the literature [13] that evaluate rebalancing strategies using uniform or purely random synthetic payment data, the optimization conducted in this study incorporates a biased payment traffic profile that more accurately reflects real-world network traffic. This approach demonstrates the feasibility of implementing effective rebalancing strategies even under non-homogeneous traffic conditions.

Regarding performance evaluation, unlike the circular rebalancing simulation conducted in [25], where the success rate is defined as the fraction of executable 1-sat payments across all possible node pairs, the simulation presented in this work models the routing of variable payment amounts that align more closely with realistic use cases.

Chapter 6

Routing node economics

Simulation results presented previously demonstrate that various channel rebalancing techniques significantly enhance the overall payment success rate across the network. However, node operators, particularly routing node operators, act as independent economic agents whose primary objective is the profit maximization of their routing activities. Consequently, a cost-incurring rebalancing operation is economically viable only if the incremental routing revenue derived from newly facilitated payments exceeds the associated rebalancing expenses.

Although overall network throughput (the number of successfully routed payments) and operator revenue (collected routing fees) are generally positively correlated, channel-specific traffic dynamics strongly dictate the performance of individual techniques. Depending on the volume and asymmetry of traffic on specific channels, certain rebalancing strategies prove to be economically and operationally superior to others.

To address these trade-offs, this chapter presents a comprehensive cost-benefit analysis for each rebalancing strategy and outlines the specific scenarios in which each is preferentially applicable, placing particular emphasis on the economic balance and financial performance of routing node operators.

6.1 Taxonomy of channel rebalancing costs

Channel rebalancing strategies incur specific operational costs that frequently appear in combination, depending on the chosen execution technique. These financial burdens can be categorized into four primary cost components.

6.1.1 On-Chain Transaction Fees

Representing the most significant cost component, on-chain fees arise from recording transactions directly onto the Bitcoin base-layer blockchain. These fees are proportional to the transaction's data size, dictated by the underlying smart contract script complexity and the number of inputs and outputs (e.g., aggregating multiple Unspent Transaction Outputs, or UTXOs, increases the data footprint due to multiple required signatures).

Following the Segregated Witness (SegWit) upgrade, transaction weight is measured in virtual bytes (vBytes), which applies a 75% weight discount to signature witness data.

Expressed in sats/vByte, fee rates fluctuate based on mempool congestion, as miners prioritize higher-bidding transactions.

While the Taproot upgrade (BIP 340–342) introduces key efficiency gains through *Schnorr signature aggregation*, which reduces the virtual size (vBytes) of complex multi-signature transactions, this analysis deliberately adopts a conservative estimation model. Because legacy SegWit scripts remain the predominant standard, all subsequent cost calculations rely exclusively on standard Native SegWit transactions to establish a reliable upper-bound baseline for operational expenses.

Although not strictly mandated by the protocol, on-chain transaction fees are critical for ensuring timely inclusion in upcoming blockchain blocks. The fee selection mechanism operates as a dynamic auction where transactions compete for limited block space, giving miners economic incentives to prioritize transactions offering the highest fee density per unit of block space occupied.

Transaction fee rates C_r are expressed in satoshis per virtual byte (sat/vByte) and fluctuate dynamically according to overall network congestion and demand for block space. Historically, baseline fees range from 1 sat/vByte during periods of minimal activity to 20–30 sat/vByte under moderate congestion, with extraordinary spikes exceeding several hundred sat/vByte during peak network saturation.

$$C_{BTC}[\text{sat}] = W_{tx}[\text{vByte}] \cdot C_r[\text{sat/vByte}]$$

For Lightning Network operations, precise fee estimation is particularly vital, as many on-chain transactions are constrained by timelocked smart contracts that require settlement within strict execution windows.

To account for these operational requirements, this study adopts a baseline fee rate of 10 sats/vByte for its analytical calculations. This parameter represents a medium-to-high priority processing speed under low to moderate congestion regimes. However, practical deployment must account for the reality that operational costs can increase substantially during high-congested peak periods.

6.1.2 Off-Chain Lightning Routing Fees

Incurred during multi-hop off-chain payment execution (such as circular rebalancing), these fees depend on the transferred principal amount and the length of the payment path (number of intermediary routing hops). While variable across different network paths, off-chain routing fees are generally significantly lower than base-layer on-chain execution costs.

To evaluate generic off-chain transaction costs across the network topology, the baseline path length is derived from the structural graph analysis presented in Figure 4.2.2. Given that the mode of the node distance distribution is 4 nodes, a representative payment traverses $h = 3$ intermediate routing hops. Routing node fee policies comprise two configurable components: a fixed base fee (C_b, in msat) and a proportional fee rate (C_r, in parts per million, ppm). The routing fee for a transferred principal P (in sats) is modeled as:

$$C_{LN}[\text{sat}] = h \cdot \left[\frac{C_b[\text{sat}]}{10^3} + \left(P \cdot \frac{C_r[\text{sat}]}{10^6} \right) \right]$$

An analysis of the Lightning Network snapshot data reveals a prevailing environment of relatively low routing fees across the network. Specifically, the network-wide 75th percentile metrics yield a base fee of 10^3 msat (1 sat) and a proportional fee rate of 500 ppm (parts per million). These minimal fee levels are largely indicative of the network's current expansion phase, in which a substantial number of non-professional node operators deploy low or zero-fee policies primarily to encourage adoption rather than to cover operational expenditures or optimize capital efficiency. Conversely, a professionally operated routing node requires a higher fee schedule to compensate for liquidity lockup, routine channel rebalancing, and infrastructure maintenance. A prominent real-world benchmark is the routing node managed by Block, Inc., which processes significant transaction volumes for its enterprise applications. By implementing higher routing fees aligned with commercial demand, Block reported an annualized yield of 9.7% on its allocated channel liquidity [32], proving that active, professional node management can be both economically sustainable and highly profitable.

Consequently, to reflect the operational parameters of a professional and financially viable node, this study adopts the following adjusted fee parameters for all subsequent calculations.

Fee type	**Amount**
C_b	10 sats
C_r	500 ppm (0.05%)

Applying these estimated parameters to the total routing fee model for an average transaction of 10^5 sat yields a mean Lightning payment cost of 180 sat.

6.1.3 Third-Party Service Provider Fees

Arising from interaction with swap services (e.g., Submarine Swaps), these fees are charged by entities facilitating cross-layer liquidity movements. Service providers (such as Boltz or Lightning Loop) pass the underlying on-chain execution costs to the user while levying an additional Service Fee, a percentage of the rebalanced volume, typically ranging between 0.05% and 0.5%.

6.1.4 Opportunity Costs

Associated primarily with passive rebalancing techniques, where node operators drastically decrease or completely eliminate routing fees on imbalanced channels to incentivize organic, directed traffic flows. Although not an upfront expenditure, this represents an implicit financial cost in the form of foregone routing revenue during the adjustment period.

6.2 Operational cost breakdown by transaction type

6.2.1 Channel opening: the funding transaction

The foundational operation of the Lightning Network is channel establishment, which is executed via an on-chain funding transaction. This operation locks funds into a 2-of-2 multisignature smart contract address (typically implemented as a Pay-to-Witness-Script-Hash, P2WSH) on the Bitcoin base layer.

The data footprint of a baseline funding transaction is broken down as follows:

Table 6.1: Funding transaction size breakdown

Component	**Size (vB)**	**Notes**
Header	10.5	Transaction Overhead
Transaction Input	68	Assuming a single P2WPKH UTXO input (scales if multiple UTXOs are aggregated)
Channel Multisig Output	42	2-of-2 P2WSH
Change Output	31	P2WPKH (applicable when the input UTXO is not fully consumed)
Total Virtual Size	≈ 153	

6.2.2 Channel closure mechanisms: cooperative vs. unilateral

Although channel closures are generally avoided during nominal node operations, given that channels are designed for long-term persistence, they become necessary during strategic liquidity reallocations. For instance, an operator may close an underperforming channel to redeploy capital toward higher-demand routes or open more profitable peer connections.

The financial cost of closing a channel depends strictly on the mechanism required.

Cooperative close (mutual close)

Under optimal conditions, both channel counterparties agree on the final balance distribution and jointly sign a standard closing transaction. This transaction settles the balance back to both parties via two outputs:

Table 6.2: Channel closing transaction size breakdown (cooperative)

Component	**Size (vB)**	**Notes**
Header	10.5	Transaction Overhead
2-of-2 Multisig Input	104.5	Includes witness data for both signatures
Settlement Outputs	62	Two P2WPKH outputs at 31 vB each
Total Virtual Size	**177**	

Unilateral close (force close)

If a peer becomes unresponsive or acts maliciously, the initiating node must unilaterally terminate the channel. This procedure requires a two-step transaction sequence to safely redeem the locked funds:

As a result, a force-close scenario incurs a data footprint roughly 2.4 times larger than a cooperative close, imposing a severe fee penalty alongside temporal liquidity lockup on the initiating node operator.

6.2.3 Submarine Swaps

The operational expenditure of a Submarine Swap (such as a Loop Out or Loop In procedure) consists of four primary cost components: an on-chain Hash Time-Locked Contract (HTLC) funding transaction, an on-chain sweep transaction, off-chain Lightning routing fees, and a third-party service provider fee.

Table 6.3: Channel closing transaction size breakdown (unilateral)

Transaction Type	**Size (vB)**	**Notes**
Commitment Transaction	≈ 255	Broadcasts the latest state on-chain. Includes header (10.5vB), 2-of-2 multisig input (104.5vB), immediate `to_remote` output, and timelocked `to_local` output.
Sweep Transaction	≈ 177	Executed after relative timelock (`to_self_delay`) expires. Includes header (10.5vB), P2WSH timelock witness input (135vB), and P2WPKH output (31vB).
Total Virtual Size	≈ 432	

Major service providers (e.g., Boltz, Lightning Loop) levy a percentage-based service fee to compensate for liquidity provision and operational overhead, while passing through or bundling the underlying base-layer transaction costs incurred on their end.

Funding transaction

Table 6.4: Submarine swap funding transaction size breakdown

Component	**Size (vB)**	**Notes**
Header	10.5	Transaction Overhead
Input	68	1 P2WPKH UTXO
HTLC Output	43	P2WSH
Change Output	31	P2WPKH (if present)
Total Virtual Size	≈ 153	

Sweep transaction

Lightning payment and service fee

Submarine Swaps fundamentally involve the settlement of a Lightning Network invoice to counterbalance the corresponding on-chain capital transfer, incorporating both the principal amount and the service provider's overhead.

Assuming a baseline route length of four hops between the local node and the service provider's Lightning node, off-chain routing fees can be estimated on the

Table 6.5: Submarine swap sweep transaction size breakdown

Component	**Size (vB)**	**Notes**
Header	10.5	Transaction Overhead
Witness Input	135	HTLC P2WSH
Destination Output	31	P2WPKH
Total Virtual Size	≈ 180	

order of a few hundred satoshis. However, these routing costs scale with payment size due to proportional fee components. For substantially larger transactions, network liquidity constraints may necessitate longer routing paths to guarantee successful payment delivery, thereby incurring additional hop-by-hop routing costs.

Despite these potential scaling factors, off-chain routing costs remain considerably lower than the required on-chain transaction fees. In practice, the service provider typically bundles these on-chain execution costs alongside their service fee (generally around 0.5% of the swapped volume) directly into the settled Lightning invoice.

6.2.4 Channel Splicing

Channel splicing enables the dynamic resizing of Lightning Network channel capacity without necessitating a channel closure. Unlike Submarine Swaps, which rely on atomic dual-transaction mechanisms, splicing requires a single on-chain transaction that spends the existing 2-of-2 multi-signature Unspent Transaction Output (UTXO) and instantiates a updated multi-signature state.

Splice-Out Operations

The transaction spends the active channel output to create two new outputs: the newly sized channel multi-signature UTXO and an on-chain destination wallet output. The total transaction virtual size is approximately 190 vBytes, structured as:

Table 6.6: Splice-Out transaction size breakdown

Component	**Size (vB)**	**Notes**
Header	10.5	Transaction Overhead
Channel Input Spend	≈ 105	2-of-2 P2WSH
New Channel Output Creation	≈ 43	P2WSH
Destination Wallet Output	≈ 31	P2WPKH
Total Virtual Size	≈ 190	

Splice-In Operations

The transaction consumes the existing channel UTXO along with an additional on-chain wallet UTXO to fund the expanded channel capacity, typically producing a new channel output and an on-chain change output. The virtual size ranges between 250 and 260 vBytes, structured as:

Table 6.7: Splice-In transaction size breakdown

Component	**Size (vB)**	**Notes**
Header	10.5	Transaction Overhead
Channel Input Spend	≈ 105	2-of-2 P2WSH
Funding Wallet Input Spend	≈ 68	On-chain P2WPKH
New Channel Output Creation	≈ 43	P2WSH
Wallet Change Output	≈ 31	P2WPKH
Total Virtual Size	≈ 260	

6.2.5 Circular rebalancing

Circular rebalancing represents the most straightforward, rapid, and cost-efficient mechanism for adjusting capital distribution across channel pairs within a routing node. Mechanically, it executes a self-directed, circular Lightning payment originating from one local channel and terminating at another channel managed by the same node, thereby allowing internal liquidity reallocation through a single off-chain transaction.

The associated cost structure aligns with the off-chain routing fee dynamics described previously. However, overall transaction fees can scale under two primary conditions: when transferring large payment amounts, where proportional fee rates dominate, or when routing across extended paths with numerous intermediary hops, which causes fixed base fees to accumulate.

Additionally, within cooperative rebalancing frameworks, routing nodes are incentivized to signal specific channels experiencing liquidity imbalances. Other network participants can then preferentially incorporate these flagged channels into their own circular routing paths. This cooperative dynamic is often reinforced by node operators temporarily reducing routing fees along unbalanced channels, potentially allowing well-connected nodes to execute circular rebalances at near-zero cost.

Finally, a target rebalancing volume can be decomposed into multiple smaller circular payments distributed across various channels. While this granular method enables multi-channel liquidity redistribution, it incurs routing fees for each individual payment executed.

6.3 Breakeven analysis

For a rebalancing transaction to be economically viable for a routing node, the subsequent payment flows enabled along the rebalanced channel must generate sufficient routing fee revenue to offset the execution costs of the rebalancing operation itself.

This economic efficiency is evaluated by comparing the node's expected average routing fee earnings against the operational cost (in satoshis) of the selected rebalancing mechanism. Consequently, the financial feasibility of a given strategy is highly sensitive to the volume and fee profile of the traffic passing through the channel.

The table below summarizes the execution costs (in satoshis) for each rebalancing approach, along with the minimum number N of routed payments required to achieve breakeven. These values are derived using an on-chain fee rate of 10 sats/vByte, a baseline path length of three intermediate hops for an average Lightning payment, and a rebalancing transaction volume of 10^5 satoshis, consistent with the operational parameters and assumptions established earlier in this chapter.

Rebalancing operation	**Type**	**Cost [sats]**	**N**
Channel opening	On-chain	1530	25.5
Circular rebalancing	Off-chain	180	3
Submarine swap	On-chain + Off-chain	3330	55.5
Channel splicing	On-chain	4500	75

By incorporating the average traffic rate traversing a node's channel (such as the transaction arrival rate parameter λ introduced in Chapter 5.2.2), it becomes possible to determine the most economically sustainable rebalancing strategy for the node. This evaluation ensures the financial viability of periodic rebalancing operations executed whenever local channel liquidity is depleted.

Chapter 7

Conclusion

In conclusion, this work built upon an introductory overview of Bitcoin blockchain scalability requirements and the technical foundations of the Lightning Network to conduct an empirical topological analysis based on a real-world network snapshot. To ensure complete reproducibility of the analytical and simulation frameworks, the data acquisition methodology and network state extraction procedures were documented in detail. The complex systems analysis revealed that key network metrics conformed to power-law distributions, exhibiting a natural propensity toward structural centralization driven by routing efficiency and management convenience.

While centralization enhanced payment routing efficiency, it also introduced severe structural vulnerabilities. The analysis highlighted that high network concentration exposed critical hub nodes as single points of failure, meaning that under node outages or targeted attacks, the network payment routing capacity was heavily impaired. Simulations on the empirical network topology demonstrated that resilience could be significantly improved by implementing an automated channel selection algorithm optimized to maximize overall network connectivity. By targeting nodes based on connectivity metrics rather than naively establishing channels with already dominant hubs, which was the standard behavior in manual setups and traditional auto-management software, the structural robustness of the network was substantially enhanced.

The second part of the study evaluated channel rebalancing strategies across varying routed payment traffic profiles and initial channel imbalance states, comparing active on-chain techniques such as submarine swaps and channel splicing, active off-chain methods like circular payment rebalancing, and passive rebalancing through dynamic routing fee adjustments. On-chain active rebalancing proved to be the most direct and effective proactive mechanism, reducing payment failure rates caused by liquidity depletion by 87 percent. In contrast, circular off-chain rebalancing in a greedy configuration, where nodes executed rebalancing indepen-

dently, yielded minimal or even negative performance impacts, as uncoordinated transactions frequently disrupted neighboring balanced channels. Conversely, cooperative circular rebalancing, wherein routing nodes signaled preferred rebalancing directions to neighbors and applied fee discounts, achieved a 70 percent performance improvement under severe initial channel imbalances while maintaining low operational costs, even in the presence of non-homogeneous payment traffic. Finally, passive rebalancing successfully restored liquidity balance from diverse initial conditions without external capital allocation, albeit requiring significantly longer convergence times compared to proactive methods.

Lastly, a detailed cost analysis evaluated the financial viability of each strategy by calculating the breakeven threshold, defined as the minimum number of routed payments required to recover rebalancing execution costs. Although versatile, on-chain rebalancing incurred substantially higher costs and execution latency. The synthesis of these results demonstrated that integrating cooperative circular off-chain rebalancing with passive dynamic fee adjustments represented the most sustainable, economical, and efficient operational framework for routing nodes over long-term execution horizons.

Bibliography

[1] Réka Albert, Hawoong Jeong, and Albert-László Barabási. Error and attack tolerance of complex networks. *nature*, 406(6794):378–382, 2000.

[2] Turki Ali Alghamdi, Rabiya Khalid, and Nadeem Javaid. A survey of blockchain based systems: Scalability issues and solutions, applications and future challenges. *IEEE Access*, 12:79626–79651, 2024.

[3] Saifedean Ammous. *The bitcoin standard: the decentralized alternative to central banking*. Wiley Hoboken, NJ, 2018.

[4] Andreas M Antonopoulos. *Mastering bitcoin*. O'Reilly Media., 2019.

[5] Andreas M Antonopoulos, Olaoluwa Osuntokun, and René Pickhardt. *Mastering the lightning network*. " O'Reilly Media, Inc.", 2021.

[6] Jacob Benesty, Jingdong Chen, Yiteng Huang, and Israel Cohen. Pearson correlation coefficient. In *Noise reduction in speech processing*, pages 1–4. Springer, 2009.

[7] Anna Bernasconi, Fabrizio Luccio, and Paolo Ferragina. *Elementi di crittografia*. Pisa University Press, 2015.

[8] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of statistical mechanics: theory and experiment*, 2008(10):P10008, 2008.

[9] Vitalik Buterin. Sharding. https://vitalik.eth.limo/general/2017/12/31/sharding_faq.html, Dec 31 2017.

[10] Guido Caldarelli. *Scale-free networks: complex webs in nature and technology*. Oxford University Press, 2007.

[11] Guido Caldarelli and Alessandro Chessa. *Data science and complex networks: real case studies with Python*. Oxford University Press, 2016.

[12] John Carvalho. Credible exit and the law of conservation of blockspace. Delving Bitcoin / https://blockspace.science/paper, May 2026. Accessed: 2026-06-15.

[13] Marco Conoscenti, Antonio Vetro, and Juan Carlos De Martin. Hubs, rebalancing and service providers in the lightning network. *Ieee Access*, 7:132828–132840, 2019.

[14] Marco Conoscenti, Antonio Vetrò, and Juan Carlos De Martin. Cloth: A lightning network simulator. *SoftwareX*, 15:100717, 2021.

[15] Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.

[16] Johannes Göbel and Anthony E Krzesinski. Increased block size and bitcoin blockchain dynamics. In *2017 27th International Telecommunication Networks and Applications Conference (ITNAC)*, pages 1–6. IEEE, 2017.

[17] Paolo Guasoni, Gur Huberman, and Clara Shikhelman. Lightning network economics: Channels. *Management Science*, 70(6):3827–3840, 2024.

[18] Yifan Hu. Efficient, high-quality force-directed graph drawing. *Math J*, 10:37, 2005.

[19] Pim Keer, Ioannis Alexopoulos, Matteo Maffei, Marco Argentieri, Andrew Camilleri, and Zeta Avarikioti. Ark: Offchain transaction batching in bitcoin. *arXiv preprint arXiv:2605.20952*, 2026.

[20] Taishi Nakai, Akira Sakurai, Shiori Hironaka, and Kazuyuki Shudo. A formulation of the trilemma in proof of work blockchain. *IEEE Access*, 12:80559–80578, 2024.

[21] Satoshi Nakamoto et al. Bitcoin: A peer-to-peer electronic cash system, 2008.

[22] Jonas Nick, Andrew Poelstra, and Gregory Sanders. Liquid: A bitcoin sidechain. *Liquid white paper. URL https://blockstream. com/assets/downloads/pdf/liquid-whitepaper. pdf*, 2020.

[23] Dmytro Piatkivskyi and Mariusz Nowostawski. Split payments in payment networks. In *International Workshop on Data Privacy Management*, pages 67–75. Springer, 2018.

[24] Rene Pickhardt. A mathematical theory of payment channel networks. *arXiv preprint arXiv:2601.04835*, 2026.

[25] Rene Pickhardt and Mariusz Nowostawski. Imbalance measure and proactive channel rebalancing algorithm for the lightning network. In *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–5. IEEE, 2020.

[26] René Pickhardt. Lightning netowork autopilot.

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

[28] Pavel Prihodko, Slava Zhigulin, Mykola Sahno, Aleksei Ostrovskiy, and Olaoluwa Osuntokun. Flare: An approach to routing in lightning network. *White Paper*, 144(1):1–40, 2016.

[29] Minfeng Qi, Qin Wang, Zhipeng Wang, Manvir Schneider, Tianqing Zhu, Shiping Chen, William Knottenbelt, and Thomas Hardjono. Sok: Bitcoin layer two (l2). *ACM Computing Surveys*, 58(3):1–37, 2025.

[30] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[31] Dinesh P Srivasthav, Lakshmi Padmaja Maddali, and R Vigneswaran. Study of blockchain forensics and analytics tools. In *2021 3rd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*, pages 39–40. IEEE, 2021.

[32] Miles Suter. Block's lightning network routing node yields. Bitcoin Conference 2025, 2025.

[33] Gábor Szabó, Mikko Alava, and János Kertész. Shortest paths and load scaling in scale-free trees. *Physical Review E*, 66(2):026101, 2002.

[34] Pieter Wuille, Eric Lombrozo, and Johnson Lau. BIP 141: Segregated Witness (Consensus layer). https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki, December 2015.

[35] Ruozhou Yu, Guoliang Xue, Vishnu Teja Kilari, Dejun Yang, and Jian Tang. Coinexpress: A fast payment routing mechanism in blockchain-based payment channel networks. In *2018 27th international conference on computer communication and networks (ICCCN)*, pages 1–9. IEEE, 2018.