Abstracting Bitcoin Lightning Network Complexity with Ultraviolet

***Abstract*—The Lightning Network (LN) emerged in recent years as the most promising solution for scaling the Bitcoin network on a second layer. While the protocol specifications reached a relative maturity level, research efforts on higher-level challenges such as topology discovery, routing, and adversarial scenarios are still in their infancy. On the one hand, traditional network modelling approaches ignore the peculiarity of LN, that is, being based on a time-flow dictated by Bitcoin layer events. On the other, deploying thousands of actual nodes would require massive computational resources and, especially from an academic perspective, non-trivial interventions on complex production-stage code to test more experimental scenarios.**

With this work, we introduce the concept of Timechain-level model, along with an open-source implementation (Ultraviolet), to abstract the complexity of the Lightning Network while still providing a vision of base layer events and protocol internals. After describing how each element of the model is mapped into the LN architectural stack, we show a case study to demonstrate its usage in investigating large-scale scenarios for research, development and educational purposes. Finally, we present a detailed comparison to properly contextualize our contribution to the current state of the art of LN modelling, highlighting the advancements introduced by a Timechain-level and future directions of research it opens.

***Index Terms*—blockchain, timechain, bitcoin, model, simulation**

I. Introduction

Over the past 15 years, Bitcoin evolved from an obscure peer-to-peer network to an established worldwide decentralized reality that implements the first form of Internet-native digital scarcity [1], [2]. As the number of nodes and users continues growing, some design principles, while essential to preserve decentralization, have become an evident limit to its scalability. Currently, nodes are able to run even on low-resource hardware, validating blocks up to 4MB every ten minutes on average, thus allowing a decentralized distribution of nodes in embedded, single-board hardware, and even mobile/IoT environments [3]. Nevertheless, even considering a conservative scenario of 7 billion entities doing 2 transactions/day, it would require 24 GB-sized blocks, that is 3.5 TB/day (1.27 PB/year) [4]. Such large space and bandwidth requirements would result in a complete loss of decentralization since only a very few nodes running on big data centres would be able to check the blocks' validity, nullifying the whole point of using a redundant chain of data blocks linked by hashes: if a trusted centralized authority is needed anyway, using a hosted database from a well-known company would be a far better choice in terms of scalability and performance.

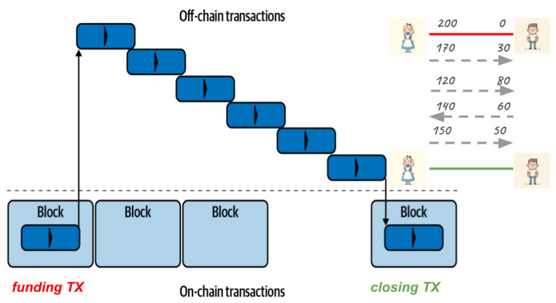

Fig. 1. Offchain transactions between two parties: only channel funding and closing events are broadcasted at the base layer

Scaling on a second layer, using the Bitcoin timechain[1] events as primitives, emerged in recent years as the most promising scaling solution to maintain base layer decentralization, and it is at the basis of the Lightning Network [5] The Lightning Network (LN) operates on top of the Bitcoin network, allowing for near-instant transactions between nodes while still providing the same properties of immutability of the base layer. In Fig. 1 is shown the fundamental concept of LN, *i.e.*, the *lightning channel*: Two nodes open a channel with each other with a base layer event, the *funding transaction*. The event locks some amount of digital assets to a multi-signature, that is, a cryptographic problem that would require private keys from parties to be unlocked. Once the channel opening is confirmed on the timechain, the two nodes can perform unlimited transactions by exchanging digitally signed transactions without having to broadcast them to the network. Referring again to Fig. 1, starting from an initial balance, each off-chain transaction updates its current distribution across the two sides of the channel, and only when/if the channel is closed the final status will be broadcasted on the Bitcoin timechain.

Currently, the LN consists of a global-scale network of tens of thousands of nodes, constantly growing in number, transaction number and number of channels. The Lightning Network has the potential to drastically improve the scalability of the Bitcoin network, opening its application to a heterogeneous new set of applications and services, from real-world assets tokenization to non-human AI agent nodes exchanging resources in pervasive Internet-of-Things environments. During the first years since the LN proof-of-concept in 2015, most of the development and research efforts have been focused

[1]To avoid any confusion with the general topic of blockchains, we will use this term, specifically indicating the Bitcoin base layer

on the major traits of the BOLT protocol [6], a collection of specifications that define every aspect of the Lighting Network elements. Only in the very recent years, as LN protocol is becoming more and more stable, interest is gradually moving toward more speculative research-oriented topics, and it is not surprising to find how exploration efforts of essential aspects such as topology graph discovery, routing, pathfinding strategies are still at their infancy.

Nevertheless, several factors still limit the current viability of many relevant research topics, especially when considering an academic-oriented perspective. Firstly, deploying a realistic scenario of thousands of actual nodes would require a massive amount of computational and economical resources (*e.g.*, funding nodes and maintaining hardware). But even more importantly, those efforts would result in a tricky testing environment where each experimental scenario would require non-trivial interventions on complex production-stage code. So, while this approach can still be useful, *e.g.*for industry and developers, to finalize specific services, it is much less viable for observing and evaluating the impact of more speculative/research-oriented global-scale aspects such as pathfinding/routing strategies, exotic graphs topologies, changes at the protocol level, adversarial scenarios, and so on. A second aspect, as we will extensively discuss in detail in Section IV, is related to the inadequacy of traditional network models (*e.g.*, graph-based, flow analysis, event-driven simulators). While those are certainly more flexible when compared to real network deployment, they all suffer from a major limitation: they ignore the key element that differentiates LN from any other network, that is, being based on a special time flow dictated by the events occurring in the realm of the Bitcoin base layer.

In this work, we introduce the concept of *timechain-level modelling*, which represents, to the best of our knowledge, the first model approach to the Lightning Network to be positioned at this level of abstraction. The key idea is avoiding the traditional levels typically adopted in network simulation, placing the abstraction model right at the level that matches the peculiarity of the Lightning Network, that is, the Bitcoin timechain. Indeed, all the time-related events in the Lightning Network are not related to some "real-world" actual time but to the events crystallized as transaction events in the Bitcoin network. The major contribution we aim to provide can be summarized as follows:

- Timechain-level: it enables testing scenarios that would be difficult to artificially inject into a network of Bitcoin nodes running a baselayer timechain, *e.g.*, congested blocks, unconfirmed transactions, or alternative structures for channel funding transactions.
- No actual node deployment: customization of several node/protocol internal mechanisms while hiding cryptographic primitives, internal byte encoding formats, and everything else that is not part of the design space.
- Global scale control: complete control on nodes/channels and timechain events, with no need of certificates/credentials to perform actions.

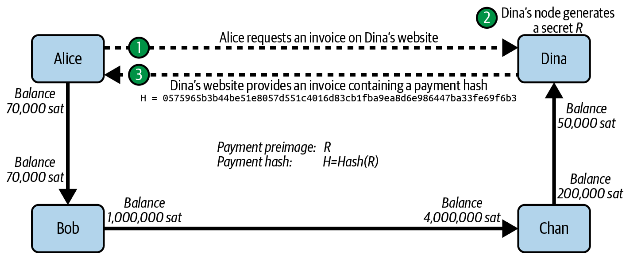
2 Dina's node generates a secret R
Alice
1 Alice requests an invoice on Dina's website
Dina
3 Dina's website provides an invoice containing a payment hash
H = 0575965b3b44be51e8057d551c4016d83cb1fba9ea8d6e986447ba33fe69f6b3
Balance 70,000 sat
Balance 70,000 sat
Balance 50,000 sat
Payment preimage: R
Payment hash: H=Hash(R)
Balance 200,000 sat
Bob
Balance 1,000,000 sat
Balance 4,000,000 sat
Chan

Fig. 2. Initial state: Alice wants to send $50,000$ sats to Dina, but there is no channel between them.

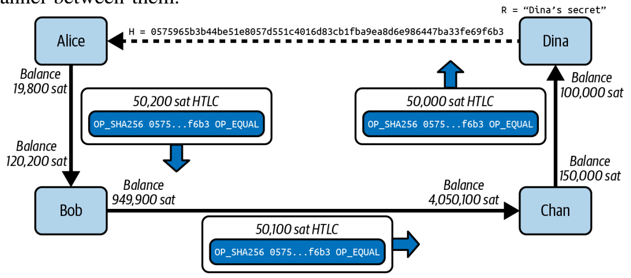

Fig. 3. Alice creates a packet with multiple HTLC containing channel updates that are locked to the hash H. When Dina receives her HTLC, reveals the secret R that produced H, so that all the channel updates will be performed atomically.

All the above have been the design guidelines to develop the project Ultraviolet (Unifying Lightning Topology Routing Abstractions VIsible On LEvel Timechain), aimed at providing an experimental platform for the Bitcoin Lightning Network, along with a modular and extensible abstraction model for hiding the complexity of the underlying elements.

II. Modeling the Lightning Network

In this section, we will start delving into the internal mechanics of the LN. This is a necessary step to understand why, in the context of the LN, many traditional network problems (*e.g.*, routing/pathfinding, resource allocation, and topology discovery) introduce new challenges and require completely different approaches. The main takeaway is that all these new challenges are strictly tied to the underlying timechain block flow, which cannot be simply dismissed or replaced with references to external world time references. Please notice that, for the sake of clarity, we will adopt the term *satoshi* or sat, which indicates the atomic unit of the digital asset represented, instead of *bitcoin* (or BTC), which is equal to 10^8 satoshis and could create confusion with the term *Bitcoin*, that refers to the protocol executed on each node.

A. Hash Time Locked Contracts

The creation and routing of a *Hash Time Lock Contract* (HTLC) represents the core mechanisms of the Lightning Network, allowing transactions between any two nodes, regardless of the presence of a channel between them. To understand the motivations behind the timechain-level approach proposed in UV, it is essential to describe how the HTLC routing is used to guarantee atomically executed, trustless multi-hop transactions. Starting from the situation depicted in Fig. 2, assuming that Alice wants to initiate a transfer of $50,000$ sats to Dina and that Alice and Dina have no channel between them, the mechanism can be summarized as follows:

1) Dina generates a random secret R, computes its SHA256 hash H and sends it to the Alice node (see Fig. 3).
2) In a phase called *pathfinding*, Alice investigates her channel graph to find a path to the destination. Balance distribution in other channels is unknown, thus Alice can only use information about the total channel capacities announced by founders when opening the channel.
3) Alice prepares a packet containing a sequence of HTLCs. Each HTLC, one per channel traversed, is a Bitcoin transaction that would result in a channel update if applied, but that is currently locked to the knowledge of the secret preimage R that produced H.
4) When Bob receives the packet, he can decrypt only the first hop HTLC, from which he learns about the amount of the update and the next hop to forward the packet (Chan).
5) Upon receiving the packet from the previous hop, each node of the path can only decrypt its own HTLC and then forward the packet to the next hop.
6) When finally the packet reaches Dina, she recognizes the hash H value: this time, the HTLC update can be applied because she actually knows the locking secret R used to create H. Doing so, Dina will eventually reveal R to Chan, who will use to update the Bob-Chan balance, and this will reveal R to Bob, and so on, backwards until Alice is reached.

As an overall result, only two nodes had an actual net change in their overall balances: Alice and Dina. All the other participants just unbalanced some incoming channel, doing the same on the outgoing channel (see Fig. 3).

Please notice that, when preparing each HTLC update, Alice did not use exactly the amount of 50,000 sats: specifically, the amount of the HTLC received will be slightly bigger than the amount of the HTLC forwarded, so that the application of both incoming/outgoing channel updates will result in a net positive for the node. As incentivizing mechanism, the Lightning Network employs a fee structure based on two components: the base fee and the fee rate. The base fee represents a fixed charge assigned to each HTLC forwarding, regardless of its amount. Instead, the fee rate, expressed as a fee per million satoshis (fee ppm), is a proportional fee that scales with the transaction's amount. Each node can periodically broadcast an announcement about forwarding costs for each of its channels, representing a further metric to evaluate the available routing paths. Returning to the example, Bob receives and update for 50,200 sats on the incoming channel, but he is asked to forward a 50,100 sats HTLC, thus resulting in a net positive of 100 sats for the routing.

Finally, while the above steps describe the common case of a successful routing, two particular conditions can occur:

1) Node has insufficient local balance to forward the HTLC: the node will send a *update_fail_htlc* message to the previous node, thus cancelling the pending HTLC. This failure message goes back from hop to hop until the initiator, which can take an appropriate action, *e.g.*, trying a different path.
2) A node just does not forward the HTLC: this can happen for several different reasons, from being malicious to simply being offline or malfunctioning. As a consequence, all the HTLC already forwarded remain pending until a specific number of timechain blocks are mined, usually referred as *cltv_expiry*. When expired, an HTLC does not need the H value to be unlocked, so it can be broadcasted on the Bitcoin timechain, closing the channel with the latest valid state.

B. How the Routing Problem is Different in the Lightning Network

On the surface, the routing problem within the Lightning Network presents aspects and terminology that resemble the well-known routing issues of computer networks. Nonetheless, the unique attributes of the LN architecture introduce additional complexities that necessitate novel approaches and strategies, sometimes involving actions that seem to be counterintuitive. The primary challenges that contribute to this heightened complexity can be summarized as follows:

1) Unknown path liquidity: Since each channel creation is a consequence of a timechain base layer transaction, *i.e.*, the *funding transaction*, channel capacities are known. Conversely, the current distribution of the channel capacity across the two funding nodes is not known to the other nodes. As a consequence, when evaluating a potential path, each node has only the knowledge of its own local channel balances, that is, the first segment of the path, but can only make probabilistic assumptions about the other channels, *e.g.*, based on their capacity, failed attempts, and so on.

2) Channel capacity is not bandwidth: When most of the transactions in a channel only occur along one direction, more and more liquidity moves from one side to the other side of the channel, making the channel progressively unbalanced. At some point, the flow along that direction will not be possible anymore due to a lack of outgoing liquidity. For example, at the end of the routing scenario depicted in Fig. 3, Alice would not be able to route even a smaller $20,000$ sats to Dina along the same path due to missing liquidity on the Alice-Bob channel (only $19,800$ sats on the local side). But, surprisingly, the increased balance on the other side will make it easier to receive from that side. Thus, channel capacity is just the constant sum of two fluctuating liquidity levels split between opposite sides, with symmetric effects that depend on the direction considered.

3) Asymmetric congestion effects: As a consequence of the above point, LN routing requires a different approach when compared to classic congestion, where we usually assume that lower traffic means lower congestion. Counterintuitively, congestion along a given channel direction has a positive effect for traffic flows in the opposite direction.

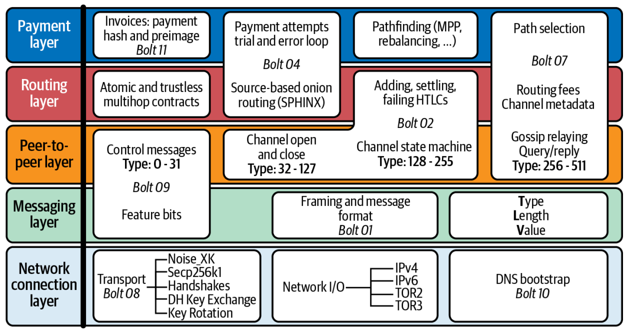
Payment layer
Invoices: payment hash and preimage Bolt 11
Payment attempts trial and error loop
Bolt 04
Source-based onion routing (SPHINX)
Pathfinding (MPP, rebalancing, ...)
Path selection
Bolt 07
Routing fees Channel metadata
Routing layer
Atomic and trustless multihop contracts
Adding, settling, failing HTLCs
Bolt 02
Peer-to-peer layer
Control messages Type: 0 - 31
Bolt 09
Feature bits
Channel open and close Type: 32 - 127
Channel state machine Type: 128 - 255
Gossip relaying Query/reply Type: 256 - 511
Messaging layer
Framing and message format Bolt 01
Type Length Value
Network connection layer
Transport Bolt 08
Noise_XK
Secp256k1
Handshakes
DH Key Exchange
Key Rotation
Network I/O
IPv4
IPv6
TOR2
TOR3
DNS bootstrap Bolt 10

Fig. 4. Layers of the Lightning Network architecture abstracted in the Ultraviolet model.

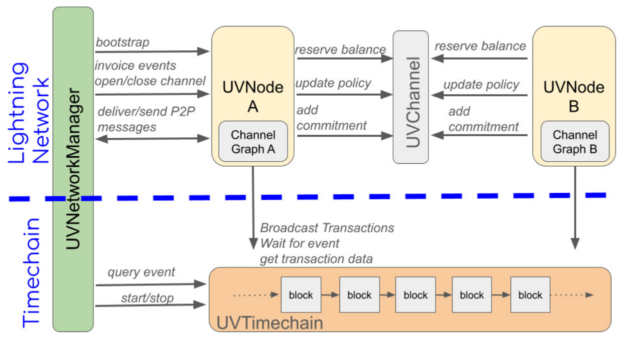

Fig. 5. Functional elements of the UV model along with their interactions.

4) Dynamic path cost: The forwarding fees, which impact the forwarding cost and thus the path viability, may change according to dynamic node policies, *e.g.*, as a reaction to what is happening along the channel. For example, if the $5M$ sats channel between Bob and Chan is selected by many others for its big capacity and low fees, then an increase of flows in the Bob-Chan direction could convince the Bob node operator (both human or algorithmic) to increase the forwarding fees to prevent the further depletion of remaining outgoing channel liquidity ($949,900$ sats). Conversely, on the other side, Chan could benefit from the big accumulated liquidity and lower the fees to forward in the opposite direction (toward Bob), with the intention to attract flows. It is easy to understand, even from this simple scenario, how the fee path cost could involve game-theory-related dynamics that go beyond the mere calculation of a cost function.

C. Mapping Ultraviolet Abstractions into LN Layers

Once the core elements of HLTC routing have been introduced, in this subsection we describe how each model abstraction of Ultraviolet can be contextualized according the layers shown in Fig. 4, commonly accepted as the reference architecture of any LN implementation [7]. With this regard, the abstractions adopted at each level can be summarized as follows:

1) Network Connection Layer: In the Lightning Network, TCP/IP and Tor-based protocol mechanisms are used to encapsulate messages, along with several encryption techniques: Ultraviolet completely abstracts this level, exposing an underlying generic simplified point-to-point transport layer and identifying each node by means of its unique public key.

2) Messaging Layer: Ultraviolet hides the internal format of messages, *e.g.*, how values and fields are encoded in order to be parsed and, in general, how the bytes are "packed" into the frame to be transmitted. Every data that needs to be exposed to the higher layers is simply stored as part of a high-level data structure,*e.g.*, as an attribute of a class object, with no requirement about the internal encoding used.

3) Peer-to-Peer layer: This layer almost matches the original LN layer, differentiating UV from most of the other models (see section IV). Peer-to-peer, control messages, and gossip channel announcements/updates are exchanged coherently with the BOLT protocol and the underlying timechain, both in terms of timings and corresponding transactions at the base layer. What is abstracted is the usage of mechanisms for authenticating/signing the messages since generating and validating the cryptographic proofs is not part of the design space.

4) Routing Layer: Any routing strategy and pathfinding strategy can be easily implemented in UV since the channel and node properties of the topology graph match those found in a real node instance. To get most of the potential for research and analysis purposes, such a graph can be generated from a synthetic scenario or even importing an actual topology from a real LN node.

Also, Fig. 4 presents a top-level payment layer that merely constitutes an external interface for addressing all the user requests. In the current implementation, UV also provides a basic frontend to test the most common functionalities, but the model is independent of the client used, and more sophisticated frontends are encouraged by the open-source nature of the UV project.

D. The UV Architectural Components

UV consists of a modular architecture of several components (see Fig. 5), where each one abstracts a subset of the functional aspects found in a Lightning Network environment. Thus, UV is not just a model of an LN node: it is a model of the interactions occurring in the whole LN environment, involving nodes, peer-to-peer networks, channels, users/operators, and the timechain base layer on top of which everything is orchestrated. For each component, here we describe and justify the abstractions implemented, showing the advantages and limitations of the design choices.

UVTimechain: It is the underlying component of the whole UV architecture, modelling the base layer where transaction events are collected into an immutable sequence of data blocks. Thus, events like channel opening/closures, HTLC expires, and all others having a base layer footprint will be tracked according to the corresponding timechain transaction instead of being merely considered as something that occurs at some point during the simulation. Only transactions related to LN are taken into account when considering confirmation times, expirations, and other block-related events: the remaining transactions, external to the LN activity, are considered as a whole, synthetically generated in order to replicate non-LN traffic. This gives the flexibility to evaluate the impact of Bitcoin-related events, such as congested blocks, high fees, and adversarial scenarios, while still maintaining a timechain-level that exposes only what is needed to evaluate their impact on the upper LN layer.

UVNode: This component abstracts both the software and hardware elements of a running node deployed on the LN and its operational/management aspects. These include opening/closing the channels, selecting peers, broadcasting network messages and routing. The behaviour of a node can be probabilistically characterized when initializing the UVNode instance or dynamically injected to recreate specific operational scenarios.

UVNetworkManager The UVNetworkManager essentially provides global control over the UV components, enabling centralized capabilities that would be impossible to replicate in a real deployment. For example, it allows the dynamic instantiation of nodes and topology graphs, starting/stopping the timechain blockflow or the P2P network message exchange. Further, this component has an "omniscient" vision of the network: it can query a node for internal channel status, allocate balance, and generate any kind of random event for testing purposes.

As shown in Fig. 4, there are two more modules which are directly related to the networked structure of the LN: the *UVChannel* and the *ChannelGraph*. The UVChannel, one instance per existing channel, matches the concept of the local LN channel, as seen from a node, and contains everything to determine the current status (balances, policies) and perform updates. As shown in Fig. 5, it is accessible by the two UVNode instances that created the channel with the funding transaction. Finally, the Graph Channel is a node-specific structure that represents the current vision of the whole network topology, both in terms of nodes and channels. Each node is responsible for updating its own graph according to the node/channel announcement messages received from the network.

III. Case Study: HTLC Failures Analysis

In this Section, we present a simple case study to show how some major properties of invoice events can impact the outcome of the HTLC path finding and routing problem described in Section II. Given the introductory nature of this first work, this should not be intended as an exhaustive study on the subject, but mainly a demonstrative usage of the model in terms of input parameters, and fine-grained output metrics to detect several internal critical conditions. While we summarized the most relevant values used in the subsections below, the complete set of configuration files and executions scripts to recreate the experiments can be found at [8].

A. Experimental Setup

The first step is to define a Lightning Network scenario in terms of topology, node profiles, and channel allocations. Two different approaches are supported in UV: (i) importing a real topology from a node and (ii) generating synthetic topologies from a probabilistic characterization of network properties and node profiles. Both approaches have their pros and cons. Importing a real topology graph certainly provides a solid starting point, but still represents a local node vision the network, which may be prone to biases. On the other hand, generating synthetic networks requires a characterization of many aspects of topology and nodes, but gives the freedom to experiment with unusual/hypothetical scenarios hardly found in node graphs.

For this analysis, we will be using the synthetic approach, defining three representative profiles as follows:

- *Small:* node capacity 100k-10M sats, 1-10 channels, with channels' size from 1k to 10M.
- *Medium:* node capacity 10M-200M sats, 10-100 channels, with channels' size from 1M to 10M.
- *Large:* node capacity over 200M, 100-500 channels, with channels' size from 10M to 100M.

This node profile's choice is based on currently observable real data [9], and while it remains a useful insight into the present network status, it will probably be subjected to obsolescence due to the rapidly evolving scenario surrounding the LN. Nevertheless, this further motivates the need for high-level methodologies and models capable of estimating the potential impact of network condition changes.

In order to evaluate the impact of different invoice event properties on HTLC pathfinding and routing success, the topology generated according to the above profiles has been tested using the following input parameters:

- *Invoice Generation Rate:* frequency of invoice events, expressed as events/node per block. For example, 0.1 indicates that, on average, each node is involved in one invoice generation every 10 blocks
- *Timechain duration:* lengths of the simulation expressed as the number of blocks generated at the base layer. This has been fixed to 1000 blocks, approximately corresponding to a real-world time of 1 week at the average rate of 144 blocks per day.
- *Min/max amount ranges:* the amount of each transaction generated. When a range is used, the values are randomly selected in the range.
- *Max fees:* the maximum amount of fees allowed by the sender, considering the cumulative amount (base and ppm fee) for each channel of the candidate path. Since the distribution of fees is not part of this study, they have been allocated using typical common values gathered from real publicly available LN data (see [9]).

For the sake of clarity, the choice of values for each of the above parameters are summarized in Table I. The aggregated results collected for the scenario and input parameters just described are discussed in the next Subsections.

TABLE I
PARAMETERS RANGES USED

Parameters	Values
Profiles distribution	small (60%), medium (30%), large (10%)
Instances	$48,000$ channels total ($3,000$ small, $15,000$ medium, $30,000$ large), 1000 nodes
Invoice Generation Rate	0.01, 0.1, 1
Amount Ranges	1000, 10k, 1M
Max Fee Rates	0, 100, 1k, 10k
Timechain Duration	1000 blocks (1 week, 144 per day average)

In order to be coherent with the LN internal mechanism, the results will be analyzed according to the two phases of the whole routing process, that is, *Pathfinding* and *HTLC Routing*.

B. Pathfinding Results

Pathfinding refers to the process of discovering and selecting a sequence of nodes and channels along which a routing attempt could be actually tried in the next phase (HTLC routing). Fig. 6 shows the overall distribution of the number of total paths connecting a sender node to a destination, considering all the invoice events and the topology generated from the parameters outlined in the previous subsection.

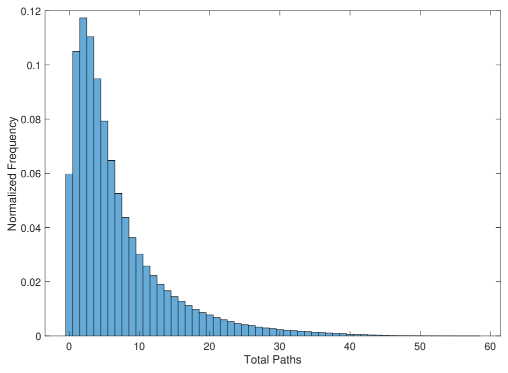

Fig. 6. Normalized frequency distribution of the computed total paths in the scenarios resulting from combining parameters in Table I.

For every invoice event, the goal of the pathfinding phase is to check each of these paths against a precise set of requirements, resulting in different outcomes that may be related to the specific properties of the event considered, e.g., amount, fees, current distribution of balances, and so on.

With this regard, Fig. 7 and 8 show, for a set of representative fee thresholds and amounts, the resulting fraction of *candidate paths* (green bar), along with the distribution of failure motivations for those paths not included among the candidates for routing.

Specifically, a path can be excluded from the candidates for the following three reasons, corresponding to the three rightmost bars on for each amount considered:

- *Fail capacity*: the channel capacity between two nodes of the path is lower than the amount specified in the HTLC. For this reason, the forwarding will certainly fail, and the path should be excluded from candidates. The channel capacity is a static attribute, so this is a permanent failure: any later attempt with the same path for equal or bigger amounts would result in the same outcome.
- *Fail fees:* the maximum fees allowed by the sender are insufficient to cover the cumulative forwarding fees required by intermediary nodes. While the fee policies along a path are not expected to change frequently, such a failure is not permanent, *i.e.*, it does not exclude any chance to retry the same path in the future
- *Fail local outbound liquidity:* the first channel of a path is the only one for which the sender knows the local balance. Thus, the sender can simply discard a path if the required outgoing liquidity is missing.

Please note that while such failure reasons are presented as fractions of the total paths, their sum is not necessarily equal to the total fraction of failures, since each single path could present multiple failure reasons at the same time, e.g., requiring too many fees to be traversed *and* not having enough capacity on some channel. As can be observed, the fraction of candidate paths decreases in both cases for increasing amounts, but only slightly (from 25% to 20%) in the case of 1000 sat fee threshold and much more strongly (from 33% to 2%) when applying a lower threshold of 100 sats.

A further inspection of the different failure reasons makes it evident that, as shown by the yellow bar in Fig. 7, exceeding the cumulative fee becomes the major reason for excluding a path when a combination of larger amounts and low fee requirements occurs.

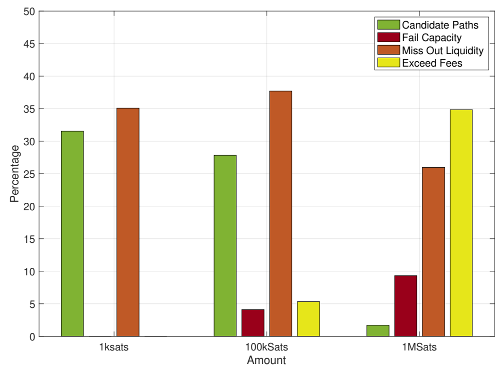

Fig. 7. Analysis of the pathfinding phase failures for three different amounts and a threshold of 100 sats of cumulative fees.

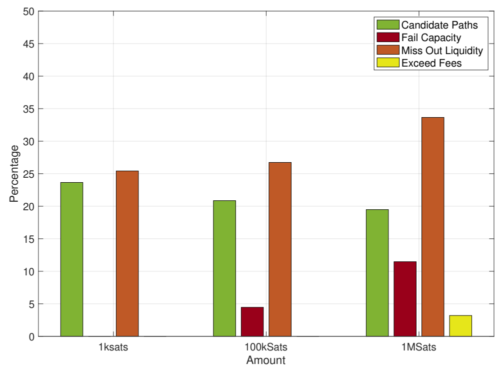

Fig. 8. Analysis of the pathfinding phase failures for three different amounts and a threshold of 1000 sats of cumulative fees.

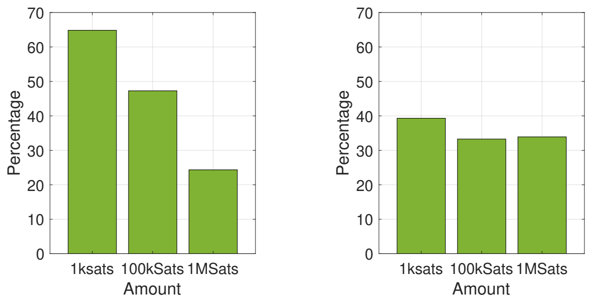

Fig. 9. Success of the attempted invoice processing events (pathfinding and HTLC routing) for fees set to 100 sats (left) and 1000 sats (right).

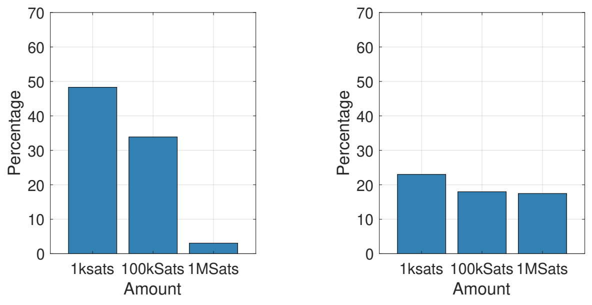

Fig. 10. Success of the whole invoice processing events (pathfinding and HTLC routing) for fees set to 100 sats (left) and 1000 sats (right).

Note that the extreme cases of 0 and 10k sat fee limits, not shown here for space reasons, can be easily deducted considering the same identical fractions, except for fee limit failures (yellow bar): a limit of zero fees would imply a 100% failure rate, while a limit of 10k would guarantee a 0% failure rate.

C. HTLC Routing Results

Once the set of candidate paths has been selected and sorted according to some metric, the *HTLC Routing* phase is started: a routing attempt is performed along each path, until a successful routing is detected or all the candidate paths have been considered and the routing can be considered as failed. Fig. 9 shows the routing phase's final outcome, as fraction of the successful HTLC routings, when considering only the invoice events that reached that stage. In particular, routing failures can happen because a channel does not have enough balance to forward the HTLC along the requested direction. As already discussed in Sec. II-A, this failure event can be detected only when actually attempting the routing, since channel balances are unknown to the sender that assembles the packet with the sequence of HTLC updates. It should be pointed out also that this is a temporary failure, since the channel balance distribution could be different in future routing attempts. As shown, a more restrictive fee limit of 100 sats (right picture) results in failures being more sensitive to amount increases (from 65% down to 25%). This might seem counter intuitive, but it is just the indirect consequence of a lower fee limit: reducing the number of paths that will be considered as candidates for routing will also reduce the number of routing attempts, and thus, the routing phase success rate. Finally, Fig. 10 aggregates all the above effects, showing the global success rate of the whole path finding and routing phases considering all the events generated during the timechain duration.

To summarize, even from this simple case study is evident how different interacting factors can severely impact the quality of path finding and routing outcomes: the Lightning Network can be currently considered in an adoption/discovery stage, and being able to rapidly capture, quantify and foresee such effects will be of primary importance for anyone involved in its development and research.

IV. Comparison With Current Models

As the major elements of the Lightning Network BOLT protocol became more and more consolidated since the initial launch [6], the need for modelization platforms surged, with different approaches and abstractions models that have been summarized in Table IV.

A first category of approaches, such as those in [10], [11], delegate the node execution model to a set of docker containers running actual node implementations (*e.g.*, LND or core lightning [17], [18]). These approaches still remain the most appropriate choice for optimizing and testing scenarios focused on realistic replication of the current LN implementations, *e.g.*, prior to the final deployment of a service or when testing edge cases for already existing LN applications. Nevertheless, considering more research-oriented aspects (*e.g.*, protocol changes, novel packet fields) would require several modifications on the complex codebase of a real full node implementation. Authors of [11] propose a similar approach, but with the possibility of deploying nodes on a large cloud environment [19]. Also within the same category, the authors of simln [12] present a tool to generate activity on top of a pre-existing network by issuing events on a subset of nodes controlled via authentication certificates. The implementation agnostic approach is certainly interesting, but the need for ad-hoc credentials for external access to existing nodes introduces major limitations, especially in deploying and controlling large-scale scenarios. A further effort worth mentioning is related to the One Million Channel Project [15], where data extrapolated from real LN traffic is fed to a real single node instance to test the scalability of the gossip message protocol implementation. However, given the specific goal of the work, the interaction is limited to the single instance to be tested, *e.g.*, to load the traffic and observe speed and memory usage metrics, thus limiting its effectiveness in evaluating global-scale scenarios.

In summary, while all the above approaches have specific use cases in which they excel, they become an uneasy path to follow when real LN node code modifications are required for experimenting with internal protocol mechanics and when a limitless control of nodes is needed to test global scale effects on large LN networks.

TABLE II
COMPARISON OF THE CURRENTLY AVAILABLE LIGHTNING NETWORK MODELS ACCORDING TO THE MOST RELEVANT FEATURES.

	Polar [10]	**Scaling Lightning [11]**	**SimLN [12]**	**PCNSim [13]**	**Cloth [14]**	**1M Channel Project [15]**	**LNsim [16]**	**UV [8]**
Model	Real nodes on containers	Real nodes on cloud	Event generator for externally deployed LN	Events on top of OMNeT++	Discrete events sim	Single node with simulated network	Simulator of nodes exchanges	Multi-threads exposing LN Design Space
Timechain	bitcoind	bitcoind	As LN backend	No	No	bitcoind	No	Thread layer exposing block flow transactions
P2P Messages	Real LN node	LN node instance	As LN backend	No	No	pre-generated from real traffic	No	Dynamically generated Gossip/Control
HTLC Model	Real LN node	LN node instance	As LN backend	add fail fulfil	No	No	No	Add, fail fulfill
Topology	Manually defined	Manually defined	As LN backend	Statically defined as input	Statically defined as input	Pre-imported	Generated using channel probabilities	Imported or generated from custom node profiles
Routing Pathfinding	As LN node backend	As LN node backend	As LN backend	Static Routing Table	Customizable	Matching the node deployed	Single Multipath	Customizable
Channels	As LN node backend	As LN node backend	As LN node backend	No policies, No balances (aggregated capacity)	Balances, base/ppm fee, timelocks	LN node, only local	balances, fees	Balances, base/ppm fee, timelocks, reserve
Control Model	Manual (with GUI)	Scripts via RPC	Static events (requires node credentials)	Static events	Static transaction events	Limited to local node	None: event probabilities as input	Global: node actions, timechain

A second class of approaches, more similar to the one proposed in this work, avoids running actual implementations by providing higher-level models to expose different aspects of a LN environment. Authors of pcnsim [13] introduce a channel network simulator built on top of the OMNeT++ [20]. In addition to entirely dismissing the concepts of timechain events and block generation, the major drawback is an over-simplification of some key aspects of LN, like merging the concepts of local/remote channel balances into a single "capacity" value, not modelling channel policies, and using a static routing table instead of performing a pathfinding phase. In cloth [14], authors present a discrete event simulator where a network scenario is statically defined and used as input. As compared to pcnsim [13], this work exposes more LN features, including channel policies and routing capabilities. However, the event-driven structure of the model ignores all the protocol HTLC/Gossip message exchanges, replacing such network dynamics with a sequence of events (*i.e.*, scheduled payments) that simply update the involved data structures. Also, the concept of timechain events is omitted, replaced by ambiguous references to real-world time: for example, it is not clear how a metric as "transactions/second", based on simulation time, could ever be consistent with events tied to timechain block generation, such as HTLC timelocks expiry, channel openings/closures, transactions confirmations, and so on.

Authors of lnsim [16] propose a simulator to demonstrate the effectiveness of a fee policy and routing strategy. The study is conducted on a limited scenario of 200 nodes, using a network topology based on some fixed probability of having a channel in common. While the purpose of investigating features such as multipath routing is certainly appreciable, ignoring the whole concept of timechain events and HTLC forwarding precludes the possibility of evaluating the side effects of simultaneous routing attempts. For example, each pending HTLC represents a "temporary reserved liquidity", reducing the actual viability of a channel on a given direction, thus affecting all the paths traversing such channel until the pending HTLC is removed. Finally, other works [21]–[23] (not included in the comparison table) focus on the economic and privacy implications of the Lightning Network's topology and traffic patterns, ignoring the design space of LN as a second layer protocol.

From all the above considerations, we can summarize the major contributions of the proposed model in the following design choices:

- Complete control on nodes/channels and timechain events, with no need of certificates/credentials to perform actions.
- Design Space focused: exposing queues management, packet fields, pathfinding strategies, HTLC routing, and Gossip P2P messages, without dealing with cryptographic primitives, byte encoding formats, network connection and all other complexities that are not part of the design space.
- Timechain-level: timeflow tied to the baselayer block generation and related transactions, which eventually dictates the evolution of fundamental events at LN level, such as channel opening/closures, HTLC expiry times, and so on.

V. CONCLUSIONS AND FUTURE WORK

In this work, we introduced the concept of timechain-level model, along with a demonstrative open-source implementation, Ultraviolet. The abstraction level proposed aims to fill the gap between the modelization of base layer events and LN protocol internals that, as discussed in a comparative analysis, is present in both real-node based and high-level simulation approches. After describing the main the elements of the model and how they are contexualized according the reference LN architecture, we presented a first case study that analyzes the distribution of HTLC routing failures in a set of representative network scenarios. Future developments include the extension of Ultraviolet modules to facilitate the investigation of promising research directions such as new channel funding mechanisms, routing stategies, channel rebalancing, along with more speculative topics related to the indirect effect of timechain aspects such as the mempool congestion, fees, and adversarial scenarios.

REFERENCES

[1] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, "Sok: Research perspectives and challenges for bitcoin and cryptocurrencies," in *2015 IEEE symposium on security and privacy*. IEEE, 2015, pp. 104–121.

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2009.

[3] A. Gervais *et al.*, "On the security and performance of proof of work blockchains," in *ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[4] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. Gün Sirer *et al.*, “On scaling decentralized blockchains: (a position paper),” in *International conference on financial cryptography and data security*. Springer, 2016, pp. 106–125.

[5] J. Poon and T. Dryja, “The bitcoin lightning network: Scalable off-chain instant payments,” 2016.

[6] O. Osuntokun *et al.*, “Bolt #1: Base protocol,” Lightning Network Specifications, Tech. Rep., 2019.

[7] A. M. Antonopoulos, *Mastering Bitcoin: Unlocking Digital Cryptocurrencies*. O’Reilly Media, Inc., 2014.

[8] Anonymous, “Anonymized for blind review,” Available: GitHub [Online]. Anonymized for blind review, Year, Repository anonymized for blind submission.

[9] “Bitcoin LN node insight - lnnodeinsight.com,” https://lnnodeinsight.com/, [Accessed 02-12-2023].

[10] J. James, “Polar: Lightning network simulator,” 2023, gitHub repository. [Online]. Available: https://github.com/jamaljsr/polar

[11] Scaling Lightning Contributors, “scaling-lightning: A simulator for the lightning network,” 2023, gitHub repository. [Online]. Available: https://github.com/scaling-lightning/scaling-lightning

[12] bitcoin-dev project, “sim-ln: A simulation of the lightning network,” https://github.com/bitcoin-dev-project/sim-ln, 2023, gitHub repository.

[13] G. A. F. Rebello, G. F. Camilo, M. Potop-Butucaru, M. E. M. Campista, M. D. de Amorim, and L. H. M. K. Costa, “Pcnsim: A flexible and modular simulator for payment channel networks,” in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, 2022, pp. 1–2.

[14] M. Conoscenti, A. Vetrò, and J. C. De Martin, “Cloth: A lightning network simulator,” *SoftwareX*, vol. 15, p. 100717, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352711021000613

[15] R. Russell and J. Netti, “Letting a million channels bloom,” Medium, 2019, available at: https://medium.com/blockstream/letting-a-million-channels-bloom-985bdb28660b.

[16] G. Di Stasi, S. Avallone, R. Canonico, and G. Ventre, “Routing payments on the lightning network,” in *2018 IEEE International Conference on Internet of Things (iThings)*, 2018, pp. 1161–1170.

[17] L. Labs, “LND - Lightning Network Daemon,” https://github.com/lightningnetwork/lnd, 2023, accessed: 2023-12-01.

[18] Blockstream and Contributors, “c-lightning - A Lightning Network implementation in C,” https://github.com/ElementsProject/lightning, 2023, accessed: 2023-12-01.

[19] K. Authors, “Kubernetes - Production-Grade Container Orchestration,” https://github.com/kubernetes/kubernetes, 2023, accessed: 2023-12-01.

[20] O. Team, “OMNeT++ - Discrete Event Simulator,” https://github.com/omnetpp/omnetpp, 2023, accessed: 2023-12-01.

[21] F. Béres, I. A. Seres, and A. A. Benczúr, “A cryptoeconomic traffic analysis of bitcoin’s lightning network,” *arXiv preprint arXiv:1911.09432*, 2019.

[22] Whiteyhat, “Lightning-network-simulator,” https://github.com/whiteyhat/Lightning-Network-Simulator, 2023, gitHub repository.

[23] Y. Guo, J. Tong, and C. Feng, “A measurement study of bitcoin lightning network,” in *2019 IEEE International Conference on Blockchain (Blockchain)*. IEEE, 2019, pp. 202–211.