Bucket: Backtrack Routing for the Lightning Network

Amin Bashiri and Majid Khabbazian
Department of Electrical and Computer Engineering
University of Alberta, Canada

***Abstract*—The Lightning Network (LN) is a pioneering payment channel network developed to address Bitcoin's scalability challenges. In LN, source nodes select transaction paths without knowing intermediary channel balances as channel balances are known only to their owners. Consequently, payments often fail when channels lack sufficient funds, forcing the source to retry the entire transaction.**

This paper introduces *Bucket*, a novel atomic-payment solution designed to reduce latency by minimizing these retries without requiring nodes to share their channel balances or the source node to lock additional funds. Bucket achieves this by routing multiple independent payment alternatives simultaneously in a "bucket," each with its own route. Intermediate nodes decrypt all alternatives, select a viable next hop based on their local channel balance knowledge, group relevant alternatives, and forward them in a new bucket. If a chosen route becomes blocked, Bucket supports efficient backtracking by enabling immediate previous nodes (not necessarily the source) to quickly select alternative paths. Simulation results using real LN data demonstrate that Bucket significantly reduces latency and improves transaction success rates, especially in challenging routing scenarios.

***Index Terms*—Bitcoin, Lightning Network, Routing**

I. Introduction

A payment channel allows two parties to transact off-chain, with blockchain interaction required only at channel creation, closure, or in the event of a dispute [16]. To create a channel, one or both parties publish an initial funding transaction on the blockchain, locking funds into the channel [2]. The total amount of locked funds is called the *channel capacity*, while the current distribution of these funds between the parties is referred to as the *channel balance*. This balance changes as the parties pay each other. When the channel is closed, the most up-to-date balance is published to the blockchain to settle the final distribution of funds.

Multiple payment channels can be linked together to form a Payment Channel Network (PCN), enabling multi-hop payments that traverse one or more intermediaries to reach the final recipient [15], [24], [37], [5], [38]. The most prominent PCN today is the Lightning Network (LN) [29], deployed on Bitcoin.

LN employs onion routing to protect privacy: the source node selects all or most of the payment path [3], [4] and constructs an onion-encrypted packet for transmission. This ensures that neither intermediaries nor observers can identify the source or destination of the payment.

Another defining characteristic of LN is the privacy of channel balances. Channel owners do not disclose their balances, so when a source node selects a path, it does so without knowledge of intermediary balances. Consequently, if a channel along the path lacks sufficient balance to forward the payment—a situation more likely for larger payments and longer paths—the payment attempt fails. An error message is then propagated all the way back to the source, which must retry the payment using a new path. This trial-and-error process increases latency and degrades the user experience.

Currently, LN exhibits high payment latency: the 90th-percentile latency is around 30 seconds, and the 95th-percentile latency reaches 3 minutes [11], with 5% of payments taking even longer. These delays highlight the need for more efficient routing strategies.

Bucket is a method introduced in this work to significantly reduce payment latency while requiring minimal changes to the LN protocol and existing implementations. Bucket addresses the root cause of high latency by avoiding full re-attempts when a path fails due to insufficient balance in an intermediary channel. Instead, it enables early and efficient identification of viable routes and rerouting of payments by intermediary nodes. Simulations using real-world LN data show that Bucket achieves substantial improvements—reducing 95th-percentile latency by approximately 76% and 90th-percentile latency by around 69%.

Bucket is fully compatible with source-based routing, the approach used by major LN implementations such as `c-lightning` [7] and `LND` [20], in which the sender determines the complete payment path. It preserves key privacy guarantees: the source does not reveal the destination's identity (which remains encrypted), and intermediary nodes are not required to disclose their channel balances.

Moreover, Bucket avoids the need for the sender to lock up more funds than the actual payment amount or to consume more Hash Time-Locked Contract (HTLC) slots—a scarce network resource—than required by standard single-path routing. It also preserves the atomicity of payments.

II. Related Work

The payment failure rate in LN, particularly for larger payments, has been a persistent concern. This section examines past attempts at mitigating this issue and highlights Bucket's advantages and uniqueness.

Improvements to Source Routing. Numerous studies have attempted to enhance the routing algorithm of source-routed onion packets [30], [32], [23], [33], [19], [39], [26], [41],

[18], [21]. These works do not leverage the intermediary nodes knowledge of channel balances and do not provide backtracking. Bucket does not overlap with these works as we propose a novel method to facilitate backtrack routing. Although the payment is still source-routed, the intermediary nodes will have influence in changing the path. All the methods mentioned above can perfectly complement Bucket.

Splitting Payments. A considerable amount of research has focused on proposing methods to split payments, particularly large payments, into smaller ones to utilize multiple paths with low balances for routing [1], [28], [36], [12], [25], [17]. However, unlike Bucket, these methods are inefficient in their consumption of HTLC slots. Moreover, due to atomicity, the failure of any single part results in the failure of the entire payment. Consequently, as demonstrated in [31], these methods may not effectively reduce latency. While relaxing the atomicity constraint [13] can mitigate this issue, most real-world use cases require atomicity to ensure reliable transactions.

Adding Redundancy. Another approach involves introducing redundancy to payments [6], [31], [35] and splitting them across multiple paths. While effective in reducing payment failure rates and latencies, these methods require the payer to lock up funds that are several times the original payment amount and are inefficient in utilizing the limited HTLC slots. Similar to the previously mentioned methods, if needed, these methods can complement Bucket by adding redundancy to payments, with each payment routed using backtrack routing to achieve a higher success rate.

Local Routing. Local routing approaches [14], [33] allow intermediary nodes to actively participate in routing decisions and can support mechanisms such as backtracking, as used in *Bucket*. While both approaches improve payment success rates and preserve privacy, they introduce notable drawbacks and trade-offs.

First, both rely heavily on *multi-path routing*, which increases the number of required *HTLCs* in the network. Given the strict per-channel HTLC limits in LN, this can lead to *HTLC exhaustion* and make the network more vulnerable to *congestion attacks* [27], [22].

Second, these protocols require the creation and maintenance of *spanning trees* to support coordinate-based or structured routing. This adds synchronization and messaging overhead, particularly in dynamic environments where channel balances change frequently. Although SpeedyMurmurs reduces this overhead through on-demand stabilization, the maintenance burden remains a scalability concern.

III. Solution Overview

At the high level, the proposed method involves sending a group of onion packets for a single payment, each created to travel on a different path. This allows the intermediary nodes to choose the best one to forward and then try the other ones if the first try fails instead of sending an error all the way back to the source to repeat the whole process.

Example. Let us explain the functionality of Bucket in a simple example. For this, consider the network illustrated in Figure 1. Suppose that node A wishes to send a payment to node E. A can choose either $A \rightarrow B \rightarrow C \rightarrow E$ or $A \rightarrow B \rightarrow D \rightarrow E$ for this purpose. Sending the payment through the above paths requires A to create onion packets $O_1 : B(C(E))$ and $O_2 : B(D(E))$ respectively.

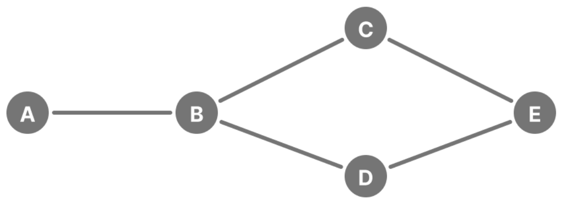

Fig. 1. The channels graph of a sample network.

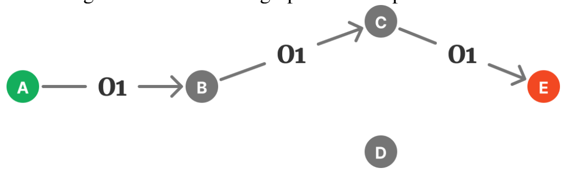

Fig. 2. Source node A sends a payment to the destination node E through the intermediary node C.

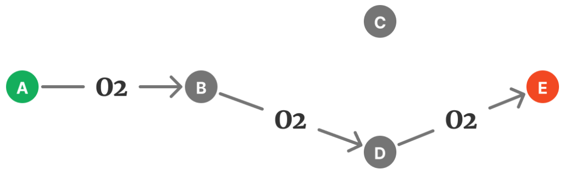

Fig. 3. Source node A sends a payment to the destination node E through the intermediary node D.

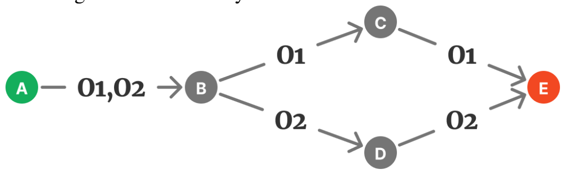

Fig. 4. Source node A sending a bucket containing multiple onion packets through the intermediary node B.

Since A does not have information about the balances of channels $B \rightarrow C$ and $B \rightarrow D$, irrespective of which of the above two paths node A selects, A's initial attempt may fail, in which case A has to try the alternative path: If A chooses to proceed through node C, it will forward the onion packet $O_1 : B(C(E))$ as illustrated in Figure 2. If the channel $B \rightarrow C$ does not have enough balance to forward the payment, it will fail, and an error will be propagated back to the source. The source node can then re-attempt the payment with the alternative path through node D and send the O_2 onion packet as shown in Figure 3. While functional, this sequential retry process can be time-consuming and inefficient, resulting in a degraded user experience.

Bucket seeks to address these inefficiencies without requiring nodes to reveal their balance information. This is done by allowing multiple onion packets to be forwarded simultaneously for a single payment without initially requiring the source node to use more funds than the original payment amount or consume more HTLCs than needed for a regular payment at any given time.

Sending the payment using Bucket is illustrated in Figure 4. As shown in the figure, Node A places the two onion packets $O_1 : B(C(E))$ and $O_2 : B(D(E))$ in a bucket and forwards the bucket to node B. The intermediary node B, knowing the balances of its channels with nodes C and D, can select one of the two onion packets and forward it to the next node. There's still a risk of failure since node B is not aware of the balances of channels $C \to E$ and $D \to E$. However, the source node A can be sure that between $B \to D$ or $B \to C$, the capable channel will be chosen by node B. This is because node B now has the flexibility to make the best decision based on its knowledge of local channel balances.

Furthermore, Bucket allows backtracking: Suppose there is enough balances on both channels $B \to D$ or $B \to C$, and that B forwards O_1 to C. If $C \to E$ channel happens to lack adequate balance, node C will return an error onion packet to node B. Recognizing that the payment via node C failed, node B can then attempt to send O_2 instead of returning the error back to node A and requiring the re-attempt of the whole process.

We note that intermediary nodes choose only one of the neighbors to forward the payment to at any given time, and if the payment through that neighbor fails, they will try another neighbor. In this case, the HTLC in the $B \to C$ channel is freed before creating an HTLC in the $B \to D$ channel. Therefore, Bucket does not consume more HTLC slots than a regular payment at any given time.

To maintain privacy in Bucket, all onion packets transferred through the network will have the same size and appear distinct at any stage, similar to the current implementation.

Unlike methods that introduce redundancy or multi-path splitting, Bucket does not require the source node to lock up more funds and consume more HTLC slots than standard single-path payments. These properties are formally stated in the following proposition.

Proposition 1. *Let L denote the length of the longest single payment path used by* Bucket*, and let amt denote the payment amount. Then, at any point in time, the number of outstanding HTLCs used by Bucket is at most L, and the total amount of funds locked in the network is bounded by $L \cdot amt$.*

Proof. At any point in time, each intermediary node in Bucket forwards the payment to at most one of its neighbors. Forwarding to multiple neighbors simultaneously would expose the node to financial risk, as it may be unable to claim the corresponding incoming HTLCs if only one of the outgoing ones is fulfilled.

Therefore, the set of outstanding HTLCs at any moment forms a prefix of a single payment path, rather than a branching structure. Since the maximum path length in Bucket is L, there can be at most L outstanding HTLCs associated with a given payment. As each HTLC locks up at most amt, the total amount of funds locked across the network is bounded by $L \cdot amt$. □

We note that Bucket generalizes the standard single-path payment model in LN. A regular payment without rerouting or backtracking is simply a special case of Bucket, where only a single path is used and no additional behavior is triggered. In such cases, Bucket behaves identically to the existing implementation—using a single onion packet and a single HTLC per hop.

Thus, using Bucket is entirely optional for each payment. A source node may choose to send a standard single-path payment, incurring no additional overhead. However, from the sender's perspective, Bucket offers a strong incentive to adopt the method due to its lower latency. Similarly, intermediary nodes are motivated to participate, as it increases their likelihood of successfully forwarding the payment and collecting fees.

IV. Implementation

In LN, a node A forwards a payment to the next node B by sending a so called `update_add_htlc` message. Table I shows the format of an `update_add_htlc` message. The message includes all necessary details for the forwarding process, such as the channel to use, the payment amount, the timeout for settlement, and an onion packet containing routing instructions.

Field	Size	Description
`type`	2 bytes	Message type (`128` for `update_add_htlc`).
`channel_id`	32 bytes	Unique identifier for the channel.
`id`	8 bytes	Unique identifier for this HTLC, assigned by the sender.
`amount_msat`	8 bytes	Amount to forward in millisatoshis (1 sat = 1000 msat).
`payment_hash`	32 bytes	SHA256 hash of the payment preimage for HTLC resolution.
`cltv_expiry`	4 bytes	The CLTV (CheckLock-TimeVerify) value for this HTLC.
`onion_packet`	1366 bytes	Full onion packet with routing information for the payment.

TABLE I
THE FORMAT OF `UPDATE_ADD_HTLC` MESSAGES IN LN.

Upon receiving the message, the node first ensures that the request is valid and compatible with the current state of its channel with the sender. This validation involves checking whether the channel has sufficient capacity to accommodate the payment and whether the provided timeout is adequate for successful processing. If the request fails these checks, the node rejects the HTLC and promptly notifies the sender of the failure.

If the HTLC passes validation, the node processes the included onion packet to determine the next step in the route. The routing instructions specify the payment amount to forward, the timeout for the next hop, and the channel to use. Using this information, the node prepares a new `update_add_htlc` message to send to the next node. Simultaneously, the node updates its internal channel state to include the new HTLC, ensuring that the payment is securely tracked until it is either completed or fails. The node then proceeds with the forwarding process by transmitting the updated `update_add_htlc` message to the next hop, progressing the payment along the route.

How Bucket Works. Bucket was designed with simplicity of implementation in mind. In Bucket, node A forwards each payment alternative using a standard `update_add_htlc` message, with a minor adjustment to indicate that the message is part of a Bucket. Initially, one might consider using the same `id` for multiple payments within a Bucket to logically group them. However, according to the LN specification (BOLT #2), each HTLC in a channel must have a unique `id`. Thus, using identical IDs for multiple HTLCs within the same channel is prohibited and would result in rejection of these payments by compliant nodes.

To maintain full compatibility with the LN protocol, Bucket instead assigns a unique `id` to each HTLC, even when forwarding multiple alternatives in the same Bucket. Logical grouping of these alternatives can still be accomplished by encoding additional metadata within the existing `onion_packet` field. Specifically, Bucket leverages the extensible Type-Length-Value (TLV) records supported by the Lightning protocol (BOLT #4) to embed a custom-defined `bucket_id`. Intermediary nodes that support Bucket recognize this TLV field and use the `bucket_id` to group related HTLCs, enabling efficient forwarding decisions without compromising protocol compliance.

Additionally, nodes can signal support for Bucket functionality using feature bits during channel negotiation as specified in BOLT #9. By introducing a custom optional feature bit, nodes explicitly advertise their ability to handle Bucket-marked HTLCs. These signaling approaches ensure seamless integration with existing LN implementations, promoting adoption without necessitating significant protocol modifications.

When node B receives the first payment from a Bucket, it processes and forwards the payment to the next node, say node C, as usual. Each payment alternative in a Bucket has a unique `id`, but node B recognizes related payments through the embedded `bucket_id` encoded in the TLV record within the `onion_packet`. When B subsequently receives another payment alternative belonging to the same Bucket, it examines the next node indicated by the routing information.

If the subsequent payment is also intended for node C, node B immediately processes and forwards it accordingly. However, if the alternative specifies a different next node, say node D, node B temporarily holds this alternative for potential future use. Should node C later notify node B of its inability to complete the payment (due to insufficient channel balance or other issues), node B can then forward the previously held alternative to node D without involving the source node, thus avoiding unnecessary payment retries.

More generally, whenever node B receives additional payment alternatives within the same Bucket, it evaluates their next hops. If an alternative targets a next node X with which node B already has an outstanding HTLC for another payment alternative in the same Bucket, node B immediately forwards the payment to node X. If no outstanding HTLC exists with node X, node B stores the alternative locally. This allows for quick re-routing decisions if earlier attempts fail.

Case study. To illustrate the effectiveness of Bucket in reducing payment latency, consider the network shown in Figure 5, where node A intends to send a payment to node E. There are six possible paths available, shown explicitly in the figure. Suppose that channels $B \to G$, $C \to E$, $B \to D$, $C \to G$, and $C \to F$ do not have enough balance to forward the payment. Without knowledge of these channel balances, the conventional sequential routing approach used by LN requires node A to try these paths one by one. This means node A must attempt each path until discovering that the first five paths fail, only succeeding on the sixth path.

With Bucket, instead of sequential attempts, node A simultaneously constructs and forwards six onion packets corresponding to these paths within a single Bucket to node B. Node B processes each packet as it arrives, immediately dropping those that cannot be forwarded due to insufficient channel balance. Specifically, upon receiving the first packet destined for node G, node B drops it due to insufficient funds on the $B \to G$ channel. The second packet, targeting node C, is successfully forwarded by node B. The third packet is similarly dropped because channel $B \to D$ lacks sufficient funds. However, packets four, five, and six, all intended for node C, are forwarded immediately since an outstanding HTLC with node C already exists.

Out of the four packets node C receives (packets 2, 4, 5, and 6), it forwards only packet 6 successfully, as channels $C \to E$, $C \to G$, and $C \to F$ lack adequate balance. Thus, node D receives a single packet from node C, which it forwards through node F to the final destination, node E.

Latency. For latency analysis, assume each message transmission or HTLC update between nodes takes α seconds. Under the sequential routing method, node A must attempt each path in order, accumulating latencies of 2α for packet 1, 4α for packet 2, 2α for packet 3, 4α for packet 4, 4α for packet 5, and 5α for packet 6. This results in a total cumulative latency of 21α.

In contrast, Bucket's simultaneous forwarding reduces latency substantially. As node B and subsequent nodes quickly discard unviable packets without needing sequential retries, the latency experienced corresponds only to the successful packet, resulting in a total latency of just 5α. Thus, Bucket significantly enhances efficiency by reducing the overall payment latency.

V. Simulation Results

We conduct simulations to demonstrate Bucket's effectiveness in reducing payment latency. Previous studies have used different types of network graphs for simulations. Some [30], [39] have employed synthetic graphs like the Watts-Strogatz

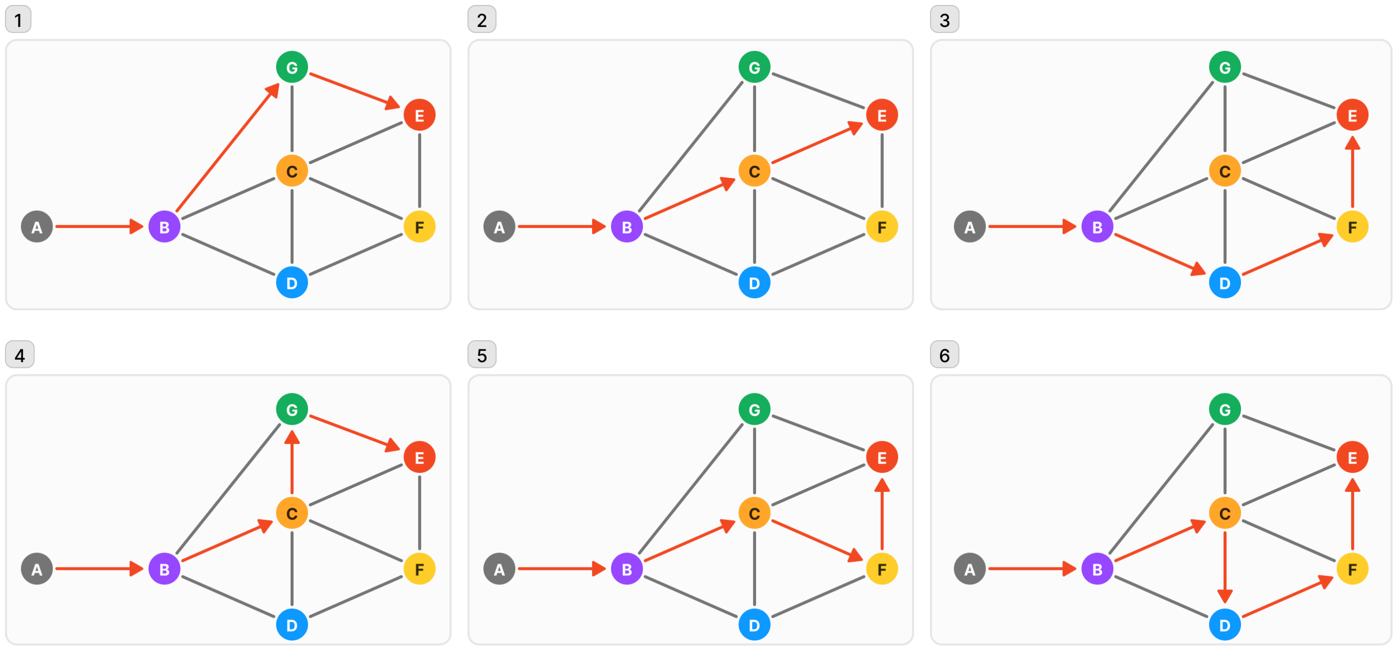

Fig. 5. Six examples of possible paths node A can take to reach node E.

model [40], while others have used real-world graph exports [14]. We adopt the latter approach to get more accurate results and run our simulations on a snapshot of the real-world LN graph, exported on May 20, 2024. At that time, the network contained 15,120 nodes and 51,579 channels.

One of the primary challenges in simulating LN is the lack of publicly available data for many of its critical attributes. Key examples include channel balances, the proportion of offline or unavailable nodes, and the amount distribution and frequency of payments within the network. Where possible, we base our simulation parameters on established studies in the field. In cases where such data is unavailable, we adopt well-founded assumptions to enhance the realism of the simulation environment.

Regarding channel capacities, while some prior studies have employed synthetic data [14], we use real-world capacities, as this information is publicly available. However, public data is lacking for channel balances, the capacity distribution between channel participants, requiring us to make informed assumptions. Prior research suggests that when the probability of payments flowing in each direction across a channel is not exactly 0.5, channels tend to saturate quickly [9], [34], [8], with most or all capacity accumulating on one side. Furthermore, publicly available data[1] indicates that the median channel age in the real-world LN is one year, with the 95th percentile reaching six years. Consequently, in our simulation, we randomly saturate each channel to one side based on this information.

[1]Channel age data was sourced from 1ml.com. While this data may not be entirely accurate, it provides a reasonable basis for our assumptions, even if only approximately reflective of real-world conditions.

In Section IV, we introduced the variable α to represent the latency of forwarding an onion packet over one hop, including the time required to update the commitment transactions or propagate an error message to the previous hop followed by a commitment update. While previous works have assumed this value to be 100ms [14], we find this assumption unrealistic, as updating commitment transactions typically requires multiple round trips. Since our simulation aims to compare the current implementation with Bucket, we continue to employ multiples of α to represent the time required for processing each transaction.

Bucket is recommended for transactions that are more difficult to route through the network because of the harsh environment, a high payment amount, or a long path. Bucket's benefits become increasingly apparent as payment amounts increase or when longer routes are needed. Smaller payment amounts are generally more likely to succeed, so they are not expected to use Bucket. However, as payment amounts grow, Bucket offers a substantial improvement in reducing latency. In this simulation, all transactions will transfer 10^5 satoshis (0.001 Bitcoin), which we consider a reasonable threshold for utilizing Bucket.

The maximum number of hops is capped at 20. This limit is determined by the fixed size of the `routing_info` field, which is 1300 bytes in the onion routing packet, and the fact that each hop payload requires 65 bytes. Based on this, the maximum number of hops is calculated as:

$$\text{Max hops} = \frac{\text{Routing information size (1300 bytes)}}{\text{Payload size per hop (65 bytes)}} = 20$$

For each routing method, we limit the maximum number of attempts or paths to 255 per payment. We assume that paths will include a minimum of four intermediary hops to ensure adequate privacy. While this assumption can vary, as different source nodes may require distinct trade-offs between privacy and latency, we consider four intermediary hops to be a reasonable baseline.

To make the simulation environment more realistic, we randomly mark 20 percent of the nodes as offline, as this has been observed for LN before [10]. If a payment attempts to pass through these offline nodes, it will fail and send an error onion packet back to the source. However, if these offline nodes are selected as either the source or destination of a payment, they will function normally, as we assume that the source and destination of legitimate payments will not be offline.

We start the simulation by randomly selecting nodes from the entire pool to generate a list of source and destination pairs. After each selection, we return the nodes to the pool before choosing another pair. Transactions between these pairs are executed in two separate simulations using different routing methods. Since the source and destination pairs and the initial graph are the same for both, and each tries up to 255 different paths per payment, the success rate will be identical but with different latencies.

For each routing method, we iterate through the source and destination pairs. A transaction is attempted only if the source and destination have sufficient balances in at least one of their channels. This condition is essential because the nodes know their channel balances and will only initiate a transaction if they satisfy the payment amount. If the conditions are unmet, we skip the source-destination pair without counting it as neither a success nor a failure and proceed to the next pair without recording any latency. This process continues until we attempt ten thousand transactions on each simulation.

Figure 6 illustrates the simulation results using box plots. To enhance clarity, the data in this figure excludes outliers. For completeness, we included all the simulation results in Table II. This table provides a detailed comparison of transaction latency at various percentiles.

The data presented in Table II demonstrates the substantial impact of Bucket, particularly for a global payment system like LN, where average latency alone does not sufficiently capture the performance improvements. In a payment network striving for broad adoption in everyday transactions, the true measure of a routing system's effectiveness lies in its performance under worst-case conditions. As the saying goes, a system is only as strong as its weakest link, or in this case, its highest latency percentiles. The improvements observed in the 90th and 95th percentiles are especially noteworthy, highlighting how Bucket significantly enhances performance in those critical cases. With Bucket, fewer users will experience extreme delays compared to the average, making the network more reliable and suitable for widespread use.

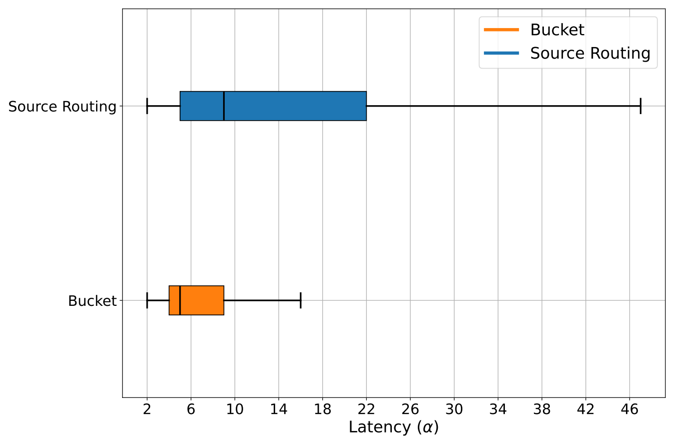

Fig. 6. Simulation results showing transaction latency improvements across different percentiles.

Latency Percentile	Source Routing (α)	Bucket (α)	Improvement
25th	5	4	20%
50th	9	5	44.44%
75th	22	9	59.09%
90th	59	18	69.49%
95th	112	27	75.89%

TABLE II
LATENCY COMPARISON ACROSS PERCENTILES BETWEEN SOURCE ROUTING AND BUCKET, ALONG WITH THE IMPROVEMENT PERCENTAGE.

VI. Conclusion

We introduced Bucket, a novel routing approach that enables intermediary nodes to re-attempt payments through alternative paths while allowing the source node to retain overall control of the process.

With Bucket, the source node sends multiple onion packets simultaneously within a virtual bucket. This allows intermediary nodes to decide which node to forward the payment to based on their knowledge of channel balances. Additionally, it allows nodes to backtrack in search of alternative paths without needing to send error messages back to the source, hence avoiding the need for the source to restart the payment process.

What sets Bucket apart from existing methods is that it does not lock up additional funds or consume more HTLC slots than a regular single payment, does not require nodes to disclose the balance information of their channels, and ensures that the destination node's identity remains encrypted.

Bucket requires minimal modifications to the existing protocol, yet our results demonstrate its ability to significantly reduce latency, particularly in challenging routing scenarios, such as those involving large payment values. Simulation results show that Bucket decreases the likelihood of users experiencing extreme delays. This enhances the network's reliability and making it more suitable for widespread adoption. Finally, Bucket complements existing payment methods, such as multi-part payments, and can be combined with them to further improve success rates and reduce latency.

References

[1] AMP: Atomic multi-path payments over lightning. `https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-February/000993.html`, Accessed: Dec. 23rd, 2024

[2] [lightning-dev] dual funding proposal. `https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-November/001682.html`, Accessed: Dec. 23rd, 2024

[3] Rendez vous mechanism on top of sphinx. `https://github.com/lightning/bolts/wiki/Rendez-vous-mechanism-on-top-of-Sphinx`, Accessed: Dec. 23rd, 2024

[4] Route blinding. `https://github.com/lightning/bolts/blob/master/proposals/route-blinding.md`, Accessed: Apr. 3, 2025

[5] Aumayr, L., Moreno-Sanchez, P., Kate, A., Maffei, M.: Blitz: Secure Multi-Hop payments without Two-Phase commits. In: USENIX Security Symposium. pp. 4043–4060 (2021)

[6] Bagaria, V.K., Neu, J., Tse, D.: Boomerang: Redundancy improves latency and throughput in payment-channel networks. In: Financial Cryptography and Data Security. Lecture Notes in Computer Science, vol. 12059, pp. 304–324 (2020)

[7] Blockstream: c-lightning (2018), `https://github.com/ElementsProject/lightning`, accessed: 2025-04-03

[8] Brânzei, S., Segal-Halevi, E., Zohar, A.: How to charge lightning: The economics of bitcoin transaction channels. In: Allerton Conference on Communication, Control, and Computing (Allerton). pp. 1–8 (2022)

[9] Dandekar, P., Goel, A., Govindan, R., Post, I.: Liquidity in credit networks: A little trust goes a long way. In: ACM conference on Electronic commerce. pp. 147–156 (2011)

[10] Decker, C.: C-lightning plugins 05: Probe plugin results. `https://blog.blockstream.com/c-lightning-plugins-05-probe-plugin-results/`, Accessed: Dec. 23rd, 2024

[11] Decker, C.: Channels, flows and icebergs. `youtu.be/HtU7ZlxvLL4?t=5810`, Accessed: Dec. 23rd, 2024

[12] Di Stasi, G., Avallone, S., Canonico, R., Ventre, G.: Routing payments on the lightning network. In: IEEE international conference on internet of things (iThings) and IEEE green computing and communications (GreenCom) and IEEE cyber, physical and social computing (CPSCom) and IEEE smart data (SmartData). pp. 1161–1170 (2018)

[13] Dziembowski, S., Kedzior, P.: Non-atomic payment splitting in channel networks. In: ACM Conference on Advances in Financial Technologies (AFT). vol. 282, pp. 17:1–17:23 (2023)

[14] Eckey, L., Faust, S., Hostáková, K., Roos, S.: Splitting payments locally while routing interdimensionally. Cryptology ePrint Archive (2020)

[15] Green, M., Miers, I.: Bolt: Anonymous payment channels for decentralized currencies. In: ACM SIGSAC conference on computer and communications security (CCS). pp. 473–489 (2017)

[16] Gudgeon, L., Moreno-Sanchez, P., Roos, S., McCorry, P., Gervais, A.: SoK: Layer-two blockchain protocols. In: Financial Cryptography and Data Security (FC). pp. 201–226 (2020)

[17] Guo, J., Shang, L., Wang, Y., Liang, T., Wang, Z., An, H.: PAMP: A new atomic multi-path payments method with higher routing efficiency. In: International Conference on Machine Learning for Cyber Security. pp. 575–583 (2022)

[18] Hong, H.J., Chang, S.Y., Zhou, X.: Auto-Tune: Efficient autonomous routing for payment channel networks. In: IEEE Conference on Local Computer Networks (LCN). pp. 347–350 (2022)

[19] Khalil, R., Gervais, A.: Revive: Rebalancing off-blockchain payment networks. In: ACM SIGSAC conference on computer and communications security (CCS). pp. 439–453 (2017)

[20] Lightning Labs: LND - Lightning Network Daemon (2017), `https://github.com/lightningnetwork/lnd`, accessed: 2025-04-03

[21] Liu, J., Chen, C., Zhou, L., Fang, Z.: Real-time recursive routing in payment channel network: A bidding-based design. In: IEEE International Symposium on Modeling and Optimization in Mobile, Ad hoc, and Wireless Networks (WiOpt). pp. 193–200 (2022)

[22] Lu, Z., Han, R., Yu, J.: General congestion attack on htlc-based payment channel networks. In: Proceedings of the International Conference on Blockchain Economics, Security and Protocols (Tokenomics) (2021), `https://arxiv.org/abs/2103.06689`, also available as arXiv:2103.06689

[23] Malavolta, G., Moreno-Sanchez, P., Kate, A., Maffei, M.: Silentwhispers: Enforcing security and privacy in decentralized credit networks. In: Annual Network and Distributed System Security Symposium (2017)

[24] Malavolta, G., Moreno-Sanchez, P., Schneidewind, C., Kate, A., Maffei, M.: Anonymous multi-hop locks for blockchain scalability and interoperability. In: Annual Network and Distributed System Security Symposium (2019)

[25] Mazumdar, S., Ruj, S.: Cryptomaze: Privacy-preserving splitting of off-chain payments. IEEE Transactions on Dependable and Secure Computing **20**(2), 1060–1073 (2022)

[26] Mazumdar, S., Ruj, S., Singh, R.G., Pal, A.: HushRelay: A privacy-preserving, efficient, and scalable routing algorithm for off-chain payments. In: IEEE International Conference on Blockchain and Cryptocurrency (ICBC). pp. 1–5 (2020)

[27] Mizrahi, A., Zohar, A.: Congestion attacks in payment channel networks. In: International conference on financial cryptography and data security. pp. 170–188. Springer (2021)

[28] Piatkivskyi, D., Nowostawski, M.: Split payments in payment networks. In: Springer International Workshop on Data Privacy Management, Cryptocurrencies and Blockchain Technology. pp. 67–75 (2018)

[29] Poon, J., Dryja, T.: The bitcoin lightning network: Scalable off-chain instant payments (2016)

[30] Prihodko, P., Zhigulin, S., Sahno, M., Ostrovskiy, A., Osuntokun, O.: Flare: An approach to routing in lightning network (2016)

[31] Rahimpour, S., Khabbazian, M.: Spear: fast multi-path payment with redundancy. In: ACM Conference on Advances in Financial Technologies (AFT). pp. 183–191 (2021)

[32] Roos, S., Beck, M., Strufe, T.: Anonymous addresses for efficient and resilient routing in f2f overlays. In: IEEE International Conference on Computer Communications (INFOCOM). pp. 1–9 (2016)

[33] Roos, S., Moreno-Sanchez, P., Kate, A., Goldberg, I.: Settling payments fast and private: Efficient decentralized routing for path-based transactions. arXiv preprint arXiv:1709.05748 (2017)

[34] Shabgahi, S.Z., Hosseini, S.M., Shariatpanahi, S.P., Bahrak, B.: Modeling effective lifespan of payment channels. arXiv preprint arXiv:2301.01240 (2022)

[35] Shen, Y., Ersoy, O., Roos, S.: Extras and premiums: Local pcn routing with redundancy and fees. In: International Conference on Financial Cryptography and Data Security. pp. 110–127 (2023)

[36] Sivaraman, V., Venkatakrishnan, S.B., Alizadeh, M., Fanti, G., Viswanath, P.: Routing cryptocurrency with the spider network. In: ACM Workshop on Hot Topics in Networks. pp. 29–35 (2018)

[37] Tripathy, S., Mohanty, S.K.: MAPPCN: Multi-hop anonymous and privacy-preserving payment channel network. In: International conference on financial cryptography and data security (FC). pp. 481–495 (2020)

[38] Tsabary, I., Yechieli, M., Manuskin, A., Eyal, I.: MAD-HTLC: because HTLC is crazy-cheap to attack. In: IEEE Symposium on Security and Privacy (SP). pp. 1230–1248 (2021)

[39] Wang, P., Xu, H., Jin, X., Wang, T.: Flash: efficient dynamic routing for offchain networks. In: International Conference on Emerging Networking Experiments And Technologies (CoNEXT). pp. 370–381 (2019)

[40] Watts, D.J., Strogatz, S.H.: Collective dynamics of ‘small-world’networks. nature **393**(6684), 440–442 (1998)

[41] Xue, H., Huang, Q., Bao, Y.: EPA-Route: Routing payment channel network with high success rate and low payment fees. In: IEEE International Conference on Distributed Computing Systems (ICDCS). pp. 227–237 (2021)