Non-intrusive Balance Tomography Using Reinforcement Learning in the Lightning Network

YAN QIAO, School of Computer Science and Information Engineering, and Intelligent Interconnected Systems Laboratory of Anhui Province, Hefei University of Technology, China
KUI WU, Department of Computer Science, University of Victoria, Canada
MAJID KHABBAZIAN, Department of Electrical and Computer Engineering, University of Alberta, Canada

The Lightning Network (LN) is a second layer system for solving the scalability problem of Bitcoin transactions. In the current implementation of LN, channel capacity (i.e., the sum of individual balances held in the channel) is public information, while individual balances are kept secret for privacy concerns. Attackers may discover a particular balance of a channel by sending multiple *fake* payments through the channel. Such an attack, however, can hardly threaten the security of the LN system due to its high cost and noticeable intrusions. In this work, we present a novel *non-intrusive balance tomography* attack, which infers channel balances silently by performing legal transactions between two pre-created LN nodes. To minimize the cost of the attack, we propose an algorithm to compute the optimal payment amount for each transaction and design a path construction method using reinforcement learning to explore the most informative path to conduct the transactions. Finally, we propose two approaches (NIBT-RL and NIBT-RL-β) to accurately and efficiently infer all individual balances using the results of these transactions. Experiments using simulated account balances over actual LN topology show that our method can accurately infer 90% ~ 94% of all balances in LN with around 12 USD.

CCS Concepts: • **Security and privacy** → **Systems security**; *Distributed systems security*; **Economics of security and privacy**; • **Theory of computation** → *Reinforcement learning;*

Additional Key Words and Phrases: Lightning Network, security and privacy, system attack, network tomography, reinforcement learning

ACM Reference Format:
Yan Qiao, Kui Wu, and Majid Khabbazian. 2024. Non-intrusive Balance Tomography Using Reinforcement Learning in the Lightning Network. *ACM Trans. Priv. Sec.* 27, 1, Article 12 (February 2024), 32 pages. https://doi.org/10.1145/3639366

This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) (No. RGPIN-2018-03896 and No. RGPIN-2016-05660), Anhui Provincial Natural Science Foundation (No. 2008085MF203), and the Fundamental Research Funds for the Central Universities of China (Grant No. PA2021GDGP0061).
Authors' addresses: Y. Qiao, School of Computer Science and Information Engineering, and Intelligent Interconnected Systems Laboratory of Anhui Province, Hefei University of Technology, Hefei, China, 230601; e-mail: qiaoyan@hfut.edu.cn; K. Wu (Corresponding author), Department of Computer Science, University of Victoria, Victoria, BC, Canada, V8P5C2; e-mail: wkui@uvic.ca; M. Khabbazian, Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada; e-mail: mkhabbaz@ualberta.ca.

ACM 2471-2566/2024/02-ART12
https://doi.org/10.1145/3639366

1 INTRODUCTION

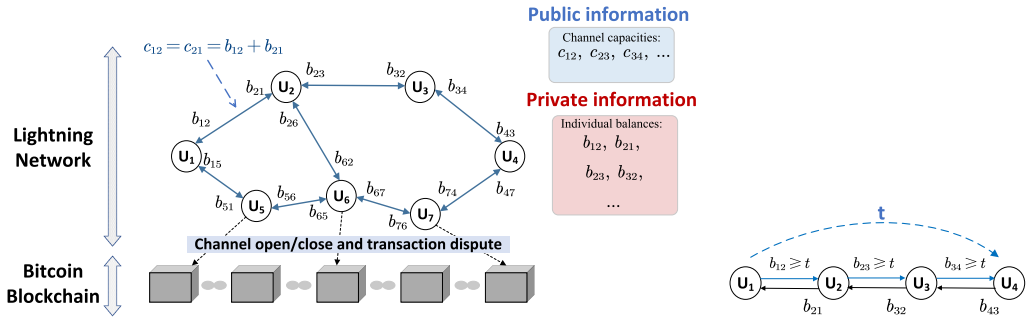
As blockchain-based cryptocurrencies, such as Bitcoin [22] and Ethereum [39], have been widely adopted today, the number of cryptocurrency-based transactions is increasing at a fast pace. Due to the inherent feature of decentralization, however, blockchain-based cryptocurrencies often rely on a global algorithm, such as the Proof-of-Work consensus algorithm [13], to confirm each transaction. This greatly limits the scalability of blockchain-based cryptocurrencies. For instance, the transaction rate of Bitcoin blockchain is within tens of transactions per second, whereas other traditional payment networks such as Visa can support peaks of up to 47,000 transactions per second [36].

To address the scalability issue of blockchain, extensive work has been done in the past years in several orthogonal directions [18]. Among them, ***payment channel network*** **(PCN)** [25, 29, 34] is one of the most promising proposals. Users who would like to make payments over a PCN open payment channels on PCN and put a certain amount of funds as a deposit secured by a smart contract. Then payments are made by re-adjusting the fund allocation on the channels. These transactions are carried out off-chain, and the PCN updates the blockchain only when needed [25]. Currently, among several fully fledged PCN systems [25, 29, 34], the **Lightning Network (LN)** [25] is recognized as most prominent in the Bitcoin community.

The framework of the LN system is illustrated in Figure 1. LN users linked by payment channels form a network in which payments can be routed between any two users. The individual funds deposited by each user is named user balance, denoted by b_{ij} in Figure 1(a), where b_{ij} is the balance of user i and b_{ji} is the balance of user j). The sum of the bidirectional balances in each channel is called channel capacity (denoted by c_{ij} in Figure 1(a)). The vast majority of transactions on LN can be made off-chain without involving the main blockchain except for some special situations, e.g., (i) channel establishment, (ii) channel close-out, and (iii) the rare events of non-cooperative behaviors (e.g., dispute). Thus, the payment overhead on the Bitcoin blockchain can be drastically reduced. Users can make payments to each other, subject to the sufficient balances on the path (as shown in Figure 1(b)). To make a tradeoff between the routing efficiency and the user privacy, LN publishes the channel capacity (i.e., the sum of the bidirectional balances in the channel) together with the IP address of each LN node but preserves the balance of each node on the channel. It also applies onion-routing protocol [14] for anonymity. With onion-routing, intermediate users (e.g., users U_2 and U_3 in Figure 1(b)) do not know who pays to whom; they only know from whom the payment is received (i.e., the immediate upstream neighbor) and to whom the payment should be forwarded (i.e., the immediate downstream neighbor).

Although the balances of the channels are essential for finding a feasible way for remote payment [26, 30, 40], current LN implementations still preserve the information for private access only. The main reasons are as follows: (1) From the view of users, the particular balances in their accounts are concerned with privacy issues, and most users are not willing to reveal such sensitive information to the public. (2) From the view of LN operators, publishing the individual balances may cause security risks, because misbehaving users may use this information to lock down other users' balances to obtain a dominant position in LN [24]. Moreover, recent studies show that if the balance information is disclosed to the public, it would be possible to use the real-time balances to track payments from a sender to a receiver, thereby compromising anonymity and privacy [16, 35]. Weighing the benefits and the damages, LN hides the balance information as its current standard.

Nevertheless, a recent study tried to disclose the private balance information under the protection of the onion-routing policy of LN [15]. In this regard, the onion-routing policy of LN is a double-edged sword. It protects the privacy of the payer and payee but in the meantime provides attackers with opportunities to threaten the security of the system. As shown in Reference [15], attackers can directly measure an unknown balance of a payment channel by executing multiple



(a) An example of LN (b) A remote payment from U_1 to U_4

Fig. 1. Framework of LN system: The channel capacity c_{ij} is defined as $c_{ij} = b_{ij} + b_{ji}$. c_{ij} is public but the balance of each user b_{ij} is private. The payer U_1 can send payment of amount t to the payee U_4 hop-by-hop via the path $U_1 \rightarrow U_2 \rightarrow U_3 \rightarrow U_4$ only if b_{12}, b_{23}, b_{34} are all higher than t.

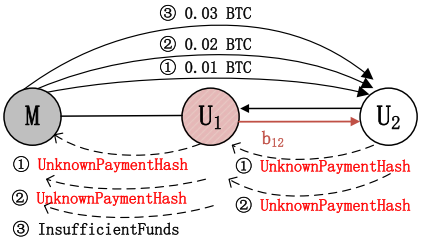


Fig. 2. Direct measurement attack on channel balances [15]: M tries to disclose the balance of b_{12} by conducting three *fake* payments to normal user U_2 through victim user U_1.

fake payments. In the example illustrated in Figure 2, assume that the attacker M wants to discover the balance b_{12}. It first opens a payment channel with U_1 and conducts three fake payments to U_2, which carried 0.01, 0.02, 0.03 BTC, respectively. The first two payments arrive at U_2, but U_2 cannot deposit them due to the incorrect payment hash. Thus, U_2 returns an error message to U_1, which then forwards it to M. The third payment cannot go through U_1 because U_1's balance, b_{12}, is insufficient. (i.e., $b_{12} < 0.03$ BTC). In this case, U_1 returns a message indicating "InsufficientFunds" to M. In this way, M can determine that the balance b_{12} is between 0.02 BTC and 0.03 BTC. Such an attack is quite simple but effective, since it is not easy to trace back to M due to the protection with onion-routing [15].

Although this direct attack can easily obtain particular balance information in LN, it hardly causes a severe threat to the security of the LN system. If an attacker conducts such an attack on multiple LN users, then the attacker may cause tangible disturbances in the network and may be detected easily. The attacker is subject to the following problems: (1) **Dishonest concerns**. The attacker needs to send a large number of fake payments to the normal users who are the next hop of the victim users (i.e., user U_2 in Figure 2). The fake payments, which can never be deposited, occupy the computational resources as well as the opportunities of the payees for receiving normal payments. When these disturbances impact a wide range of LN users, the LN operators may investigate the dishonest users and find out the attacker; (2) **Time-consuming**. To obtain one balance of a channel, the attacker needs to perform tens of iterations, on average, consuming nearly one minute. The attacker would need months to disclose a large number of balances in LN; (3) **Expensive**. To measure the balances on multiple nodes, the attacker has to open multiple channels,

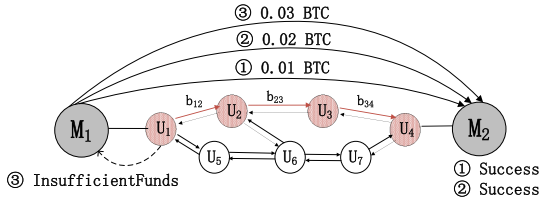


Fig. 3. Infer channel balances with balance tomography: M_1 and M_2 collaboratively infer the balances of b_{12}, b_{23}, and b_{34} by conducting three legal payments from M_1 to M_2 through victim users U_1, U_2, U_3, and U_4.

causing high fees in channel opening for the attacker [5]. In summary, the power of the attacker is quite limited.

Is it possible to attack a large number (or even the complete set) of the private balance information in LN simultaneously in a *silent*, *efficient*, and *low-cost* way? We provide a positive answer in this article by proposing an indirect way to *infer* the balance information based on legal payments, which are routed through the victim channels. More specifically, we use a pair of accounts connected to LN with two payment channels and perform legitimate transactions between our two accounts to infer balances of all the intermediate channels traversed by the transactions. The two accounts could be created by the same attacker or by two collusive attackers. Payments are made from one attack account to the other attack account, just like transferring money back-and-forth between the same person's left hand and the right hand. By doing this, all balances information in LN can be obtained silently—without any abnormal events; efficiently—all balances are inferred simultaneously; and economically—only need to open two channels. We name this attack *balance tomography*, whose framework is shown in Figure 3.

In summary, this article makes the following contributions:

— We pioneer the use of network tomography [38] for inferring all channel balances in LN. Through conducting *legitimate* payments between our two accounts, we infer the balances using the results of these payments. If a payment is successfully fulfilled, then all balances along the payment path are larger than the payment amount; otherwise, the balance of the channel that failed the payment is smaller than the payment amount. Thus, balance inference incurs min-type operations and is more challenging than existing network tomography research that studied additive metrics, such as delay [20], where the value of a path is the total value of all the links on the path.

— We propose a path construction method (named *RLPath*) to explore the most informative path in LN based on **reinforcement learning (RL)**. We first develop an algorithm (*optPayAmt*) that can efficiently compute the optimal payment amount and the maximum *gain* that the payment can obtain on each end-to-end path with logarithmic complexity. We then formulate the path construction problem for balance tomography as an RL problem and adopt a model-free on-policy RL scheme: Sarsa (λ) [33], which takes the maximum *gain*s of paths as the reward function to accumulate the weights of actions in each episode.

— We design two adaptive balance tomography approaches to infer all balances accurately and efficiently. Based on *RLPath*, we develop a non-intrusive balance tomography approach (named NIBT-RL) that shrinks the possible ranges of all balances to their minimum by conducting the optimal payments on the constructed paths. To avoid the heavy invoking of *RLPath* and accelerate the running speed of NIBT-RL, we also propose a fast framework named NIBT-RL-β that allows *RLPath* to output a batch of paths instead of a single optimal path at a time and shrinks the possible ranges of all balances on the paths simultaneously

in each updating round. In addition, both approaches can be further accelerated by the parallel mode.
— We evaluate NIBT-RL and NIBT-RL-β with *simulation* over real LN topology.[1] Experimental results show that NIBT-RL-β only degrades 2% ∼ 3% of the accuracy of NIBT-RL but saves the computing time by two orders of magnitude. Comparing our attacking method with the existing direct attacking method in a network with 55,428 balances, our attack can accurately infer more than 90% of all balances with the cost of about 12.26 USD in total without generating any abnormal event, while the direct attack would cost 5,709.96 USD and would generate nearly 780,000 fake payments.

2 CHALLENGES AND RELATED WORK

2.1 Challenges in Inferring Balances

At first glance, probing balances of a channel may seem like bandwidth inference in traditional computer networks. The balance inference problem in LN is, however, more challenging than the bandwidth inference problem, as explained below.

2.1.1 Cost. In the balance inference problem, one needs to open at least one payment channel in LN, deposit enough funds in the channel, and make multiple payments. These operations impose three kinds of cost [17]: (1) the transaction fee of opening and closing channels, (2) the routing fee collected by intermediate LN nodes once the payments have been fulfilled, and (3) the *potential* loss for locking funds in the channel. An efficient balance inference method should minimize the overall cost. Among the above three, the first cost (about 1.53 USD[2] per channel) dominates the second cost (about 1.09×10^{-6} USD per-hop per-USD transfer). The third cost is difficult to evaluate, because it totally depends on how you invest the locked funds in other places. Hence, we only consider the first two costs in the article.

2.1.2 Constraint on the Maximum Payment. LN limits the maximum amount one can transfer in a single payment to about 0.043 BTC and the maximum amount one can put in a channel to about 0.167 BTC [25]. These constraints make it difficult to infer the exact balances of a channel with large capacity.

2.1.3 Balance Dynamics. Once a payment is successfully delivered, all the balances of the channels on the payment path will decrease accordingly by the payment amount. Therefore, unrestricted probing payments may lead to extensive skewed channels, which may block the normal transactions as well as the subsequent probings. In addition, the inferred results may become meaningless if the inference process alters the values of balances.

2.1.4 Dishonesty Concerns. To reduce the risk of being perceived, the attacks should not cause too much interruptions to the general functions of the LN system. Otherwise, it may cause dishonesty concerns from both the users and the LN operators.

2.2 Related Work

In recent years, a large number of deanonymization attacks on Bitcoin's peer-to-peer network have been proposed [1]. Most of them use graph analysis [12] or transaction clustering [4] to link

[1]Note that we did not perform any attack in the real LN network to avoid any damage to the LN community.
[2]In LN, the cost is calculated in the unit of Satoshis. In the article, we use the market price as of May 25, 2020, for cost estimation, which is 1 Satoshi = 8.8982×10^{-5} USD. The first cost is estimated according to the average transaction fee from February 25, 2020, to May 25, 2020 [5].

users' IP addresses with their pseudonyms in the Bitcoin network. However, there are only a few works proposed to pry into the private balances of the users.

Herrera-Joancomartí et al. [15] first proposed to infer the balances of users by repeatedly probing the LN channel with multiple fake payments. As the attack in Reference [15] cannot be used to infer the channels whose capacities exceed the maximum payment constrained by LN, Dam et al. [37] improved the attack by using an additional channel to conduct a two-way probing for the large-capacity channels. Tikhomirov et al. [35] proposed a probing approach that sequentially probes each channel in the target list by binary search. Rahimpour et al. [28] utilized multi-path payments to speed up the balance discovery for single balance. Biryukov et al. [3] proposed a parallel channel probing method that can discover the balances in multi-hop channels. As the probing results of above approaches heavily depend on the error messages returned from the victim nodes, Kappos et al. [16] proposed to use two malicious accounts A and D to form a routing path $A \rightarrow B \rightarrow C \rightarrow D$. Then, balance $B \rightarrow C$ can be probed by conducting fake payments through the path. This probing method is more generic, since the probing results can be directly determined by account D. Specifically, if the payment hash can arrive D, then we can conclude that the balance from B to C has sufficient funds; otherwise, the balance is insufficient to route the payment.

Although the above work made various attempts to address part of the challenges, they suffer from two pitfalls. First, all of these attacks fail to make full use of each payment. As each payment is designed to attack a single channel, the number of required payments surges as the number of target channels increases. Meanwhile, an extensive number of attackers' accounts is required to cover all the target channels, incurring expensive attacking cost. Second, all of these approaches utilize fake payments. As LN system limits the maximum number of payments that a channel can simultaneously hold (483 by default), the extensive number of fake payments may affect the normal transactions that should have had the opportunity to be served by LN. Hence, the general functions of LN would be harmed by the large number of fake payments for various reasons. For example, the payer of a transaction may fail to send the payment to the payee because one of the intermediate channels holds too many fake payments. Also, the intermediate nodes that have participated in forwarding the payments may fail to collect the routing fees, as the payee cannot redeem the fake payments. Hence, although the present LN specification does not specify any punishment for making such fake payments, there is no reason to believe that the LN community is not concerned when facing spikes of fake payments, especially when these payments hinder the general functions of the system.

In this article, we aim to tackle all the challenges mentioned above by designing an efficient balance inference approach, which has low cost, can make full use of each payment, can infer large balances, does not change current balances and, above all, generates no fake payments (i.e., non-intrusive).

Remark 1. In our preliminary work [27], the proposed method NIBT can infer **the balances of channels covered by a given path set**. The experiments in Reference [27] used six path sets (generated by naïve methods) to evaluate the performance of NIBT, where these path sets can only cover 0.6% ~ 1.7% of all balances in the LN system. The results indicated that the accuracy of NIBT is sensitive to the given path sets: different path sets lead to diverse accuracies of NIBT, ranging from 50% to 93%. In this extended article, we develop a path construction method using reinforcement learning that not only can provide high quality for balance inference but also can cover all balances in the network. Using the new path construction method, we propose two novel balance tomography approaches (named NIBT-RL and NIBT-RL-β, respectively) to accurately infer all balances covered by the constructed paths. Experimental results in this article demonstrate that the new approaches can steadily provide high accuracy (ranging from 90% to 94%) for inferring **all balances of channels in the LN system**.

3 NON-INTRUSIVE BALANCE TOMOGRAPHY USING REINFORCEMENT LEARNING

3.1 Overview

In this section, we design a novel balance inference approach, where we first open two LN accounts (e.g., M_1 and M_2, as shown in Figure 3) and infer all balances in the topology by conducting multiple payments between the two nodes.

This is the first time that the concept of *network tomography* [38] is introduced to this area. In the context of *network tomography*, the performance of internal links (or nodes) can be inferred by conducting multiple packet-level probes on end-to-end paths. As the probes may cause additional load to the network, it is necessary to optimize the scheduling of the probes to minimize the measurement cost. Therefore, there are two key problems for *network tomography*: (1) How to construct the optimal paths to send the probes? (2) How to make the inference based on the probing results? The *balance tomography* in LN also faces the same two problems. In addition, to optimize the measurement cost in LN, it needs to determine the optimal amount of payment that should be transferred on each selected path. Therefore, for *balance tomography* in LN, there are three critical problems that should be addressed: (1) How to construct the optimal paths to conduct the payments? (2) How to determine the optimal payment amount on each path? (3) How to infer all of the balances based on the payment results?

In the following sections, we will propose our three methodologies to solve the above problems: (1) We first propose an optimal payment amount algorithm (denoted by *optPayAmt*) to compute the optimal payment amount on each path. (2) We then propose a path construction method based on reinforcement learning (denoted by *RLPath*) to construct the optimal paths. (3) Finally, we propose our two balance tomography approaches (denoted by NIBT-RL and NIBT-RL-β) to infer all balances by conducting optimal payments on the constructed paths. Both of the two approaches can be accelerated with the parallel mode. Particularly, our balance tomography approaches disclose the balances with the following steps: First, we connect our two accounts to the LN network and conduct *RLPath* algorithm to construct an optimal routing path between the two accounts. Then, we compute the optimal payment amount by *optPayAmt* algorithm and make the payment on the path. Finally, we observe the payment result and shrink the value ranges of the balances on the path. The above steps are repeated until all ranges of balances in LN have been reduced to their minimum.

Before introducing the methods in detail, we first explain why our methods can tackle the challenges mentioned in Section 2.1.

- **Low cost:** As mentioned earlier, among the three kinds of cost for performing balance inference, the fees for opening and closing channels dominate the others. Compared to the direct probing method (shown in Figure 2) that attacks one node with one channel [15], we just need to open two channels in total to infer all balances. Besides, the routing fees can also be minimized by our path construction method.
- **Non-interference in current balances:** A successful payment changes the balances of the channels on the payment path. To restore the balances, we require the recipient to quickly refund the sender using the reverse path.
- **Non-intrusive for other LN nodes:** All payments that we use for balance inference are legal payments without any misbehavior, and the intermediate nodes can obtain an incentive (i.e., the transaction fees) once the payments are fulfilled. In addition, since our method does not interfere with the current balances, it causes no disturbance on LN users' transactions.
- **Applicable for large channels:** All existing approaches for balance disclosure conduct *fake* payments to probe the channels[15, 21, 35]. It is impossible for these methods to probe the channels whose balances exceed the maximum amount allowed in one payment (0.043 BTC). This is because they do not create any successful payment and thus cannot accumulate

multiple payment amounts to probe a large balance. With our approach, we can let the recipient node hold multiple payments simultaneously and then pay back the payments to the sender node after the information of the large balance has been probed.

3.2 Analysis and Problem Formulation

In this section, the balance tomography problem is mathematically formulated. The main related symbols in this article are listed in Table 1.

We model LN with a directed graph $G(\mathbf{N}, \mathbf{E})$, where $\mathbf{N}$ is the set of LN users and $\mathbf{E}$ is the set of channels between the users. For a channel $e_{ab} \in \mathbf{E}$ between user a and user b, b_{ab} represents the balance from the user a to user b. $c_{ab} = b_{ab} + b_{ba}$ represents the capacity of the channel. l_{ab} and u_{ab} denote the lower and upper bounds of balance b_{ab}, respectively. We use $\mathbf{B}$ to denote the set of all balances, $\mathbf{C}$ to denote the set of all capacities, and $\mathbf{L}$ and $\mathbf{U}$ denote the sets of lower and upper bounds, respectively.

As shown in Figure 3, M_1 and M_2 are our two monitoring nodes, which connect to LN and make payments to each other to discover channel balances. We define a path $p_i = \{b^i_1, \ldots, b^i_{n_i}\}$ as a group of balances along the path direction from one of our accounts to the other, where b^i_k is the kth balance on path p_i. The lower bound and the upper bound of b^i_k is represented by l^i_k and u^i_k, respectively.

Remark 2. The two nodes that our monitoring nodes connect to may not deposit any funds in the channels. For example, in Figure 3, the balance from U_1 to M_1 and the balance from U_4 to M_2 may both be zero in the beginning. In this case, a payment from M_1 to M_2 would not be possible. This is the so-called *inbound capacity problem* [19], which may occur for any newly opened channel. The problem can be easily solved with various methods. For example, our two accounts can increase their inbound capacity by spending in LN [19]. Note that we do not need to infer the balances on the two channels directly connected to the two monitors, e.g., the balance from M_1 to node U_1 and the balance from U_4 to M_2, as these balances are known to the monitors. Hence, $p_i = \{e^i_1, \ldots, e^i_{n_i}\}$ does not include the first and the last channels.

We use an indicative variable $p_i(m)$ to denote the result of a payment on p_i with an amount m, where $p_i(m) = 0$ means the payment has been successfully fulfilled, and $p_i(m) = k$ $(0 < k \leq n_i)$ means the payment failed at the kth channel. Note that LN offers the above information to the sender (i.e., a payment is either successful or failed at an intermediate channel).

Clearly, there is a correlation between the possible ranges of balances on the path and the payment result $p_i(m)$:

(1) $p_i(m) = 0$ if and only if all balances on the path are no smaller than m;
(2) $p_i(m) = k$ $(0 < k \leq n_i)$ if and only if all balances before the kth channel are no smaller than m and the balance of the kth channel is smaller than m.

Therefore, we can deduce the upper bounds and lower bounds of balances on p_i based on the payment results on the path. Given a series of payment results $\mathbf{P}_i(m^i_1, \ldots, m^i_{t_i}) = \{p_i(m^i_1), p_i(m^i_2), \ldots, p_i(m^i_{t_i})\}$, the ranges $[l^i_k, u^i_k]$ of $b^i_k (1 \leq k \leq n_i)$ can be evaluated as

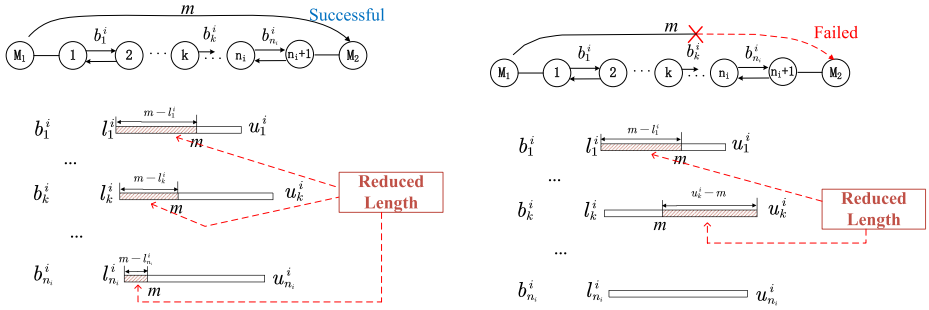
$$l^i_k = \max_{p_i(m^i_j)=0 \,||\, p_i(m^i_j)>k, 1\leq j\leq t_i} m^i_j \tag{1}$$

$$u^i_k = \min_{p_i(m^i_j)=k, 1\leq j\leq t_i} m^i_j. \tag{2}$$

For the path set $\mathbf{P} = \{p_1, p_2, \ldots, p_n\}$, suppose $\{\mathbf{P}_1(m^1_1, \ldots, m^1_{t_1}), \mathbf{P}_2(m^2_1, \ldots, m^2_{t_2}), \ldots, \mathbf{P}_n(m^n_1, \ldots, m^n_{t_n})\}$ are the payment results on the paths. Using these payment results, the range $[l_{ab}, u_{ab}]$

Table 1. Defined Symbols

Symbols	Description
$G(\mathbf{N}, \mathbf{E})$	The topology of LN
$\mathbf{N}$	The set of LN users
$\mathbf{E}$	The set of payment channels
e_{ab}	The payment channel between user a and user b
b_{ab}	The balance of channel e_{ab} from user a to user b
c_{ab}	The capacity of the channel e_{ab}
u_{ab}	The upper bound of balance b_{ab}
l_{ab}	The lower bound of balance b_{ab}
$\mathbf{B}$	The set of all balances in $G(\mathbf{N}, \mathbf{E})$
$\mathbf{C}$	The set of all capacities in $G(\mathbf{N}, \mathbf{E})$
$\mathbf{L}$	The set of lower bounds of all balances
$\mathbf{U}$	The set of upper bounds of all balances
M_1, M_2	The two attacker's nodes (i.e., the monitoring nodes)
p_i	The ith end-to-end path between M_1 and M_2
b_k^i	The kth balance on path p_i
l_k^i	The lower bound of b_k^i
u_k^i	The upper bound of b_k^i
n_i	The number of balances on path p_i
$p_i(m)$	The result of a payment on p_i with an amount m
$\mathbf{P}_i(m_1^i, \ldots, m_{t_i}^i)$	The set of results of payments on p_i
η	The predefined payment budget
$Gain^i(m)$	The *gain* of payment m on path p_i
$Gain_k^i(m)$	The *gain* of payment m on balance b_k^i
$H_k^i(m)$	The kth additive term in (14)
$\hat{u}_k^i$	The minimum value of $\{u_j^i\}$, $1 \le j \le k$
EI_k	The *effective interval* of $H_k^i(m)$
λ_j^i	The *positive interval* of $Gain^i(m)$
$\bar{\pi}$	The average width of the balance ranges
$\bar{d}$	The average degree of LN nodes



(a) Length of balance ranges that can be reduced by a successful payment.

(b) Length of balance ranges that can be reduced by a failed payment.

Fig. 4. An example of the length of balance ranges that can be cut off by a payment.

can be updated further as

$$l_{ab} = \max_{b_k^i = b_{ab}, 1 \le i \le n} l_k^i \tag{3}$$

$$u_{ab} = \min_{b_k^i = b_{ab}, 1 \le i \le n} u_k^i, \tag{4}$$

where $b_k^i = b_{ab}$ means payments on path p_i traverse the balance b_{ab}, where b_{ab} is the kth balance on p_i. According to Equations (3) and (4), the range $[l_{ab}, u_{ab}]$ of balance b_{ab} can be narrowed down given each payment result.

Inferring all balances in a given LN topology can be formulated as

$$Minimize \quad \sum_{b_{ab} \in \mathbf{B}} (u_{ab} - l_{ab}) \tag{5}$$

$$s.t. \quad \sum_{1 \le i \le n} \left| \mathbf{P}_i \left(m_1^i, \ldots, m_{t_i}^i \right) \right| \le \eta \tag{6}$$

$$l_{ab} = \max_{b_k^i = b_{ab}, 1 \le i \le n} l_k^i \tag{7}$$

$$u_{ab} = \min_{b_k^i = b_{ab}, 1 \le i \le n} u_k^i, \tag{8}$$

where $|\mathbf{P}_i(m_1^i, \ldots, m_{t_i}^i)|$ denotes the number of payments on path p_i. η is defined as the payment budget that limits the total number of payments we could conduct.

Remark 3. (**The budget** η) As discussed in Section 2.1, the cost for opening/closing channels in our method is fixed (i.e., fees for opening/closing two channels). Hence, we only need to control the routing fee. Since the per-hop transfer fee is extremely low, we can treat the routing fee for each end-to-end payment roughly the same, and as such the η value can well reflect the actual monetary cost. Besides, when setting a suitable budget η, we should also consider another important factor—the time used for balance inference, which is directly linked to the total number of payments. For this reason, we use the total number of (end-to-end) payments η to control the "budget," i.e., inequality (6).

According to the goal in (5), the quality of a payment can be quantified by the total length of the ranges that the payment can cut off. For example, consider the path $p_i = \{b_1^i, b_2^i, \ldots, b_{n_i}^i\}$ shown in Figure 4. The corresponding lower and upper bounds of these balances are $\{l_1^i, l_2^i, \ldots, l_{n_i}^i\}$ and $\{u_1^i, u_2^i, \ldots, u_{n_i}^i\}$, respectively. When the payment that carries m amount is fulfilled (as shown in

Figure 4(a)), it can reduce the total length of balance by $\sum_{1\le j\le n_i}\max\{0, m-l_j^i\}$; Otherwise, if the payment failed at the kth balance (as shown in Figure 4(b)), then it can reduce the total length of balance by $\sum_{1\le j\le k-1}\max\{0, m-l_j^i\}+\max\{0, u_k^i-m\}$. We define the expected length of the balance ranges that a payment can cut off as the *gain* of the payment. The *gain* of a payment m on path p_i can be formulated as

$$\begin{aligned} Gain^i(m) = {} & Gain_1^i(m) + prob\left(b_1^i \ge m\right) Gain_2^i \\ & + \cdots \\ & + prob\left(b_1^i \ge m\right)\cdots prob\left(b_{n_i-1}^i \ge m\right) Gain_{n_i}^i, \end{aligned} \tag{9}$$

where $0 \le prob(b_k^i \ge m) \le 1, 1 \le k \le n_i$ is the probability that payment amount m can go through b_k^i. $Gain_k^i$ is the *gain* on b_k^i that can be calculated as

$$Gain_k^i(m) = prob\left(b_k^i \ge m\right)\left(m - l_k^i\right) + prob\left(b_k^i < m\right)\left(u_k^i - m\right). \tag{10}$$

Remark 4. When m exceeds the maximum payment amount allowed in one payment (denoted by z), m will be split into multiple sub-payments: $z, z, \ldots, m - \lfloor \frac{m}{z} \rfloor \cdot z$. Then, $Gain^i(m)$ will be calculated by summing up the gain of each sub-payments. To avoid redundancy, we only present partial of our methods based on Equation (9) in this section under the assumption that $m \le z$, while the evaluations in Section 4 are carried based on the full version of our methods.

Before performing any measurement, we do not have any information regarding the balances. As such, the best one can do is to follow the "principle of insufficient reason" [10], i.e., "assigning uniform prior distributions to unknown parameters." Using this principle, we assume initially that the balance b_k^i follows the uniform distribution on the range $[l_k^i, u_k^i]$. The probability density function of the balance b_k^i is thus

$$f(x) = \begin{cases} \frac{1}{u_k^i - l_k^i} & l_k^i \le x \le u_k^i \\ 0 & otherwise. \end{cases} \tag{11}$$

Therefore, the probability of $prob(b_k^i \ge m)$ can be calculated by

$$prob(b_k^i \ge m) = \frac{u_k^i - m}{u_k^i - l_k^i}. \tag{12}$$

$Gain_k^i(m)$ in (10) can be rewritten as

$$\begin{aligned} Gain_k^i(m) = {} & \frac{u_k^i - m}{u_k^i - l_k^i}\left(m - l_k^i\right) + \frac{m - l_k^i}{u_k^i - l_k^i}\left(u_k^i - m\right) \\ = {} & \frac{u_k^i - m}{u_k^i - l_k^i} \cdot 2\left(m - l_k^i\right). \end{aligned} \tag{13}$$

Then, $Gain^i(m)$ in Equation (9) can be rewritten as

$$Gain^i(m) = \sum_{k=1}^{n_i} 2\left(m - l_k^i\right) \cdot \prod_{j=1}^{k} \frac{u_j^i - m}{u_j^i - l_j^i}. \tag{14}$$

Remark 5. In Equations (12)~(14), $\forall k : 1 \le k \le n_i$, if $m < l_k^i$, $m - l_k^i = 0$ and $\frac{u_k^i - m}{u_k^i - l_k^i} = 1$. If $m > u_k^i$, then $m - l_k^i = u_k^i - l_k^i$ and $\frac{u_k^i - m}{u_k^i - l_k^i} = 0$.

Hence, the objective function in (5) can be written as

$$Maximize \quad \sum_{p_i \in \mathbf{P}} \sum_{1 \le j \le t_i} Gain^i\left(m_j^i\right), \tag{15}$$

where $\mathbf{P}$ is the set of all paths, and $\{m_1^i, \ldots, m_{t_i}^i\}$ is the set of all payment amounts on path p_i.

Due to the iterative bound update process, the optimization problem does not render an analytical solution. To tackle this, we design a heuristic strategy that conducts the end-to-end payment with the maximum *gain* in each round until the number of payment exceeds η or there is no payment that can obtain any *gain*. In the following sections, we first propose an optimal payment amount algorithm that can efficiently compute the optimal amount with the maximum *gain* on each path. We then develop a path construction algorithm using reinforcement learning to construct the near-optimal path that renders potential optimal payment over all end-to-end payments. Finally, we propose our balance tomography strategy to infer all balances with limited payment budget.

3.3 Computing the Optimal Payment Amount on Each Path

For path p_i, $Gain^i(m)$ can be maximized by searching $m^* = \arg\max Gain^i(m)$ on the range $[\min_{1 \le k \le n_i} l_k^i, \max_{1 \le k \le n_i} u_k^i]$. However, as the ranges of balances are generally in the order of $10^3 \sim 10^7$ Satoshis, it is intractable to find m^* through brute force. However, we will demonstrate next that the optimal amount m^* can be found by a binary search with logarithmic complexity.

Let us use $H_k^i(m)$ to denote the additive terms in Equation (14):

$$H_k^i(m) = 2(m - l_k^i) \cdot \prod_{j=1}^{k} \frac{u_j^i - m}{u_j^i - l_j^i}. \tag{16}$$

Denoting $\hat{u}_k^i = \min\{u_j^i\}$, $1 \le j \le k$, we have $\forall k : 1 \le k \le n_i$,

$$H_k^i(m) \begin{cases} > 0 & l_k^i < m < \hat{u}_k^i \\ = 0 & \text{Otherwise.} \end{cases} \tag{17}$$

Considering Equation (17), we define the interval $(l_k^i, \hat{u}_k^i)$ as the *effective interval*, denoted by EI_k, of the function $H_k^i(m)$.

Let $l^i = \min_{k=1}^{n_i} l_k^i$ and $u^i = \max_{k=1}^{n_i} \hat{u}_k^i$. The interval (l^i, u^i) is the widest range where $Gain^i(m)$ can be positive. Sort the values $\{l_1^i, \hat{u}_1^i, l_2^i, \hat{u}_2^i, \ldots, l_{n_i}^i, \hat{u}_{n_i}^i\}$ in the ascending order. Then, these ordered values can divide (l^i, u^i) into *at most* $2 \times n_i - 1$ sub-intervals, which are denoted by $\lambda^i = \{\lambda_1^i, \lambda_2^i, \ldots, \lambda_{2 \times n_i - 1}^i\}$. Each sub-interval λ_j^i falls in one of the following three (exclusive) cases:

(1) $|\lambda_j^i| = 0$, i.e., λ_j^i is a point.
(2) $|\lambda_j^i| > 0$ and $\forall k, m : 1 \le k \le n_i, m \in \lambda_j^i, H_k^i(m) = 0$.
(3) $|\lambda_j^i| > 0$, there exist some non-empty set $A_j^i \subseteq \{1, 2, \ldots, n_i\}$, such that $\forall k, m : k \in A_j^i, m \in \lambda_j^i, H_k^i(m) > 0$. In other words, some $H_k^i(m)$ are positive in the sub-interval λ_j^i.

For the first two cases, we get $Gain^i(m) = \sum_{k=1}^{n_i} H_k^i(m) = 0$. For the third case, we get $\forall m : m \in \lambda_j^i, Gain^i(m) = \sum_{k=1}^{n_i} H_k^i(m) = \sum_{k \in A_j^i} H_k^i(m) > 0$. Hence, in the third case, we call λ_j^i as a *positive interval* of $Gain^i(m)$.

Proposition 1. *$Gain^i(m)$ is a strictly concave function with only one maxima on its positive intervals.*

ALGORITHM 1: Optimal Payment Amount Algorithm (*optPayAmt*)

Input: p_i, $\mathbf{L^i}$, $\mathbf{U^i}$
Output: m^*, $Gain^{*i}$
1 $\mathbf{EI}^i \leftarrow effectiveIntvl(\mathbf{L^i}, \mathbf{U^i}, p_i)$
2 $\lambda^i \leftarrow positiveIntvls(\mathbf{EI}^i)$
3 **for** $\lambda^i_j \in \lambda^i$ **do**
4 $\quad [Gain^{*i}_j, m^*_j] \leftarrow binarySearch(\lambda^i_j, p_i)$
5 $[Gain^{*i}, m^*] \leftarrow \max_{\lambda^i_j}(Gain^{*i}_j, m^*_j)$
6 **return** m^*

The proof of Proposition 1 can be found in the Appendix.

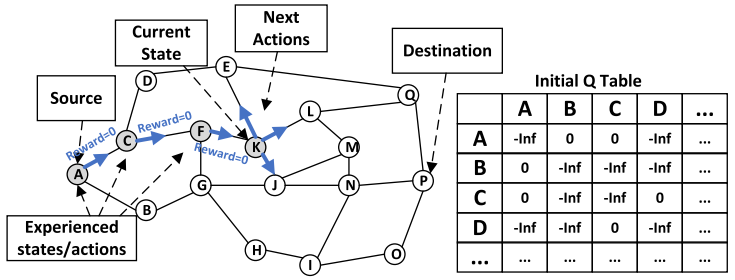
By Proposition 1, we can find the optimal payment amount m^* by searching the maximas on each *positive intervals* and selecting the maximum value among all these maximas (as shown in Algorithm 1). On each *positive interval* λ^i_j, we use a binary search to find the maxima. This requires $O(\log|\lambda^i_j|)$ iterations. We name this process *binarySearch* and omit its pseudo-code in Algorithm 1.

The time complexity for calculating the *gain* of a payment along path p_i is $O(|p_i|)$, where $|p_i|$ is the number of balances on path p_i. By Proposition 1, the gain function is concave on each positive interval, thus, we can use a binary search to find the optimal payment amount. Therefore, the maximum number of iterations on interval λ^i_j is $\log|\lambda^i_j|$. The maximum complexity for searching an optimal payment amount on λ^i_j is $O(|p_i| \cdot \log|\lambda^i_j|)$. Hence, the time complexity of Algorithm 1 is $O(|p_i| \cdot |\lambda^i| \cdot \log|\lambda^i_m|)$, where $|\lambda^i_m|$ is the average width of positive intervals in λ^i.

3.4 Constructing End-to-end Paths Using Reinforcement Learning

In this section, we aim to find a near-optimal end-to-end path that can render a near-maximum *gain* between our two accounts. The optimal path can be obtained by simply forming a complete path set that includes all possible end-to-end paths between our two nodes and comparing their maximum *gains*. However, the time complexity for searching all end-to-end paths between a pair of nodes is $O(n!)$, where n is the number of nodes in the network. In a network with thousands of nodes, it is impossible to form the complete set of all end-to-end paths. A practical way is to design a path optimization method that can find the optimal path in a heuristic manner. In the research area of optimization, there are several classical heuristic approaches, such as genetic algorithms and simulated annealing. However, these methods utilize static policies that interact over multiple episodes with a separate instance of the environment. In our problem, the environment will be updated once a new payment result is observed. Hence, the policy should have the ability to interact with the environment and make sequential decisions based on both previous knowledge and the new observations. Fortunately, **reinforcement learning (RL)**, which searches for an optimal (or near-optimal) solution in sequential decision systems, can well address the above issue. In the following, we first formulate our path construction problem as an RL problem and then propose our RL-based path construction algorithm.

In RL, we need to explicitly define four components: (1) the *environment* $\mathcal{E}$, which interacts with the RL *agent*; (2) the state space $\mathcal{S}$ that the agent can transit; (3) the *action* space $\mathcal{A}$, from which the *agent* can take an act; and (4) the *reward* function $r(s, a)$, which determines the *agent*'s reward when the *agent* takes action a in state s. Figure 5 illustrates the above four components in the context of path construction:



	A	B	C	D	...
A	-Inf	0	0	-Inf	...
B	0	-Inf	-Inf	-Inf	...
C	0	-Inf	-Inf	0	...
D	-Inf	-Inf	0	-Inf	...
...	...	...	...	...	...

Fig. 5. Formulating the path construction problem as an RL problem.

(1) **Environment $\mathcal{E}$:** The environment $\mathcal{E}$ is defined as the LN system together with the ranges of all balances. In this environment, we aim to find an optimal path between our two accounts that has the maximum *gain* over all end-to-end paths;
(2) **State space $\mathcal{S}$:** When the *agent* starts to explore an optimal path hop-by-hop from the source node to the destination, a state s is defined as the current node and the ranges of balances that we currently know;
(3) **Action space $\mathcal{A}$:** An action a is defined as choosing a next hop from the available successors that directly connect with the current node. All available successor nodes form the action space;
(4) **Reward function $r(s, a)$:** When the current state is s, the reward function of taking action a is defined as follows: $r(s, a) = 0$, if the next state is not the destination; $r(s, a) = Gain(m^*, \hat{p})$, if the next state is the destination, where $Gain(m^*, \hat{p})$ denotes the maximum gain on path $\hat{p}$, where $\hat{p}$ is the path constructed by all previous hops in the current episode, m^* is the optimal payment amount on $\hat{p}$, computed by Algorithm 1.

In RL, the *agent* can provide an optimal (or near-optimal) control policy for a **Markov decision process (MDP)**. We adopt the tabular method [33], where a *Q table* records the cumulative weight across all states and actions. In our model, the rows of the *Q table* represent the nodes of states, and the columns represent the successor nodes by taking the corresponding actions. $Q(s, a)$ is initialized to 0 if a is a feasible action for s (i.e., there is a link connecting the node of s and the node of a); Otherwise, $Q(s, a)$ is set to $-\infty$. The values in *Q table* are updated *in each single step* according to the reward function. In our path construction model, it is more reasonable to issue a reward to every experienced state/action that contributes to the reward than to just reward the current step. Therefore, we adopt a model-free on-policy RL scheme—Sarsa (λ) [33]. In particular, Sarsa (λ) looks up the next policy value when it updates the *Q-value*. It uses an *eligibility trace* to record all the experienced actions in current episode and updates the *Q-value* for all the experienced actions by the current reward. The attenuation parameter λ, which ranges from 0 to 1, determines the importance of previous actions on obtaining the reward: when $\lambda = 0$, only the last action is eligible for the reward; when $\lambda = 1$, all experienced actions can obtain an equal reward.

The pseudo-code of path construction using RL (*RLPath*) is shown in Algorithm 2. The algorithm receives the LN topology $G(\mathbf{N}, \mathbf{E})$, the lower and upper bounds of balances $\mathbf{L}$ and $\mathbf{U}$, the source and destination nodes M_s and M_d, the attenuation parameter λ, and the number of maximum episodes I as inputs. It outputs the set of all discovered paths **pathSet** with their corresponding *gains* $Gain(\mathbf{pathSet})$. *RLPath* first initializes the Q_T (the Q table), the set of the discovered path **pathSet**, and the number of current episode i (line 1~line 3). It then performs I episodes to explore the end-to-end paths and update the Q table (line 4~line 22). In each episode, it first assigns the current state s by the source node M_s (line 6), adds the first node to the path trace (line 7), chooses

ALGORITHM 2: Path Construction Using Reinforcement Learning (*RLPath*)

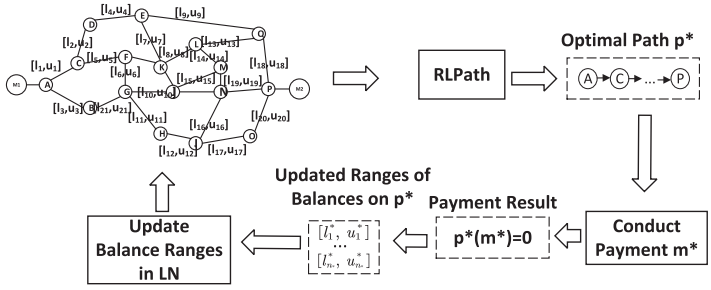
Input: $G(\mathbf{N}, \mathbf{E})$, $\mathbf{U}$, $\mathbf{L}$, M_s, M_d, λ, I
Output: **pathSet**, $Gain$(**pathSet**)

```
1  Q_T ← init(G(N, E))
2  pathSet ← NULL
3  i ← 0
4  while i ≤ I do
5      i++
6      s ← M_s
7      pathTrace ← addTrace(s)
8      [a, s'] ← chooseAction(Q_T, s, ε)
9      while True do
10         pathTrace ← addTrace(s')
11         if s' == M_d then
12             R ← optPayAmt(pathTrace, U, L)
13             [pathSet, Gain(pathSet)] ← addSet(pathTrace, R)
14         else
15             R ← 0
16         [a', s''] ← chooseAction(Q_T, s', ε)
17         Q_T ← updateQTable(Q_T, R, s', a', pathTrace, λ)
18         if s' == M_d || s'' == NULL then
19             Break
20         else
21             s ← s'
22             s' ← s''
23 return pathSet, Gain(pathSet)
```

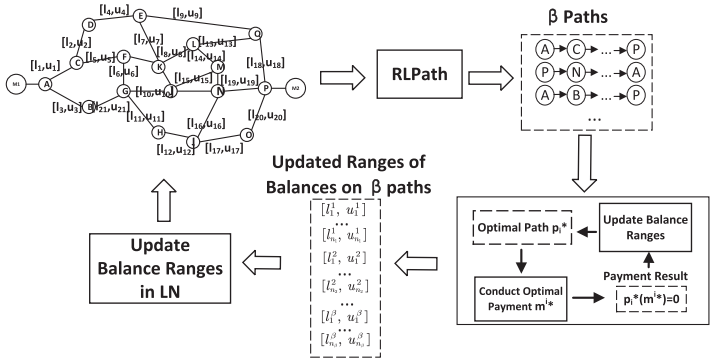
an action a based on Q_T by ϵ-greedy, and observes the next state s' (line 8). Then, it uses a while loop to construct a path from M_s to M_d hop-by-hop (line 9~line 22). In the while loop, the next state s' is first added to the path trace (line 10). If s' is the destination, then the reward R is assigned to the maximum *gain* of the path computed by Algorithm 1 (line 12). The path trace together with its *gain* will be added to **pathSet** and $Gain$(**pathSet**), respectively (line 13); otherwise, the reward will be set to 0 (line 15). Afterwards, the algorithm chooses the next action a' based on s' (line 16) and updates Q_T (line 17). Finally, the exploration will move forward until it reaches the destination or meets a dead end.

After I episodes, Algorithm 2 outputs all discovered paths and the corresponding *gains*. Our experimental results in Section 4.2 show that as the number of episodes goes up, the average path *gain* discovered by Algorithm 2 increases. This demonstrates that the algorithm is effective in exploring the optimal paths.

In Algorithm 2, the top two time-consuming operations are *optPayAmt* and *chooseAction*. As mentioned in Section 3.3, the time complexity of *optPayAmt* is $O(|pathTrace| \cdot |\lambda^i| \cdot \log |\lambda^i_m|)$. As the number of nodes in *pathTrace* and the number of intervals in λ^i is limited, the time complexity of *optPayAmt* is mainly affected by the width of interval λ^i_m, which is no more than the average width of the balance ranges $\bar{\pi}$. The average time complexity of *chooseAction* is $O(\bar{d})$, where $\bar{d}$ is the average degree of LN nodes. Therefore, the time complexity of Algorithm 2 is $O(I \cdot (\log \bar{\pi} + \bar{d}))$. In theory, the **Lightning Network (LN)** can accommodate payment paths with cycles or loops.



(a) The framework of NIBT-RL.



(b) The framework of NIBT-RL-β.

Fig. 6. The frameworks of our non-intrusive balance tomography methods.

In practice, however, such looped paths are typically avoided due to their inefficiency and high cost. Despite this, Algorithm 2 can construct paths with loops as well. However, focusing our search on only simple paths within LN can significantly reduce the computational time required by Algorithm 2 due to the reduced $\bar{d}$ in non-loop paths. Our experiments revealed that including both loop and non-loop paths in the search for optimal paths increased accuracy by approximately 1%, while it led to a substantial increase in computational time, by nearly an order of magnitude, compared to searching only among non-loop paths. Given that LN employs a source routing policy, where payers determine the routing paths for their payments, Algorithm 2 prioritizes the construction of simple paths to optimize efficiency.

3.5 Inferring All Balances in the Topology

Using the above results, we propose our two balance tomography approaches, named NIBT-RL and NIBT-RL-β, to infer the minimum ranges of all balances in the LN topology.

The framework of NIBT-RL is shown in Figure 6(a). Using the topology of LN as input, *RLPath* first constructs the end-to-end path set and provides the path with the maximum *gain*, then NIBT-RL computes the optimal payment that should be made on the path. After the payment result is observed, the ranges of balances on the path are updated according to the result. NIBT-RL repeats the above operations round-by-round until the number of payments exceeds a budget η or no balance ranges can be further cut off. The detailed pseudo-code of NIBT-RL is shown in Algorithm 3.

Algorithm 3 receives the LN topology $G(\mathbf{N},\mathbf{E})$, the set of channel capacities $\mathbf{C}$, the two end nodes M_1 and M_2 that are connecting with our two accounts, the attenuation parameter λ, the maximum number of episodes I, and the payment budge η as inputs. It outputs the inferred upper and lower

ALGORITHM 3: Non-intrusive Balance Tomography Using RL (NIBT-RL)

Input: $G(\mathbf{N}, \mathbf{E})$, $\mathbf{C}$, M_1, M_1, λ, I, η
Output: $\mathbf{L}$, $\mathbf{U}$
1 $\mathbf{L} \leftarrow 0$
2 $\mathbf{U} \leftarrow \mathbf{C}$
3 $n_p \leftarrow 0$
4 **while** $Gain^* > 0 \& n_p \leq \eta$ **do**
5 $\quad \mathbf{pathSet}_1 \leftarrow RLPath(G(\mathbf{N}, \mathbf{E}), \mathbf{U}, \mathbf{L}, M_1, M_2, \lambda, I)$
6 $\quad \mathbf{pathSet}_2 \leftarrow RLPath(G(\mathbf{N}, \mathbf{E}), \mathbf{U}, \mathbf{L}, M_2, M_1, \lambda, I)$
7 $\quad p^* \leftarrow topMaxGain(\mathbf{pathSet}_1, \mathbf{pathSet}_2)$
8 $\quad [m^*, Gain^*] \leftarrow optPayAmt(p^*, \mathbf{L}, \mathbf{U})$
9 $\quad Result \leftarrow conductPayment(p^*, m^*)$
10 $\quad [\mathbf{L}, \mathbf{U}] \leftarrow updateRange(p^*, m^*, Result)$
11 $\quad n_p + +$
12 **return** $\mathbf{L}, \mathbf{U}$

bounds of all balances. At the beginning of the algorithm, the lower and upper bounds of the balances are initialized to 0 and their channel capacities, respectively (line 1~line 2). The number of payments is initialized to 0 (line 3). The algorithm then starts to update the upper and lower bounds of the balances until there is no payment that can obtain any *gain* or the number of payments n_p exceeds the threshold η (line 4~line 11). In the *while* loop, it first constructs end-to-end paths by Algorithm 2 (line 5~line 6) and picks out the path p^* with the maximum *gain* (line 7). Note that the algorithm constructs two path sets $\mathbf{pathSet}_1$ and $\mathbf{pathSet}_2$ in each round, where M_1 and M_2 alternately act as the source and destination. Then, it computes the optimal payment on p^* by Algorithm 1 (line 8) and conducts the payment on the path (line 9). After observing the payment result, it updates the lower and upper bounds of the balances on the path (line 10) and proceeds to the next round.

The most time-intensive step is constructing the path set by *RLPath* (line 5~line 6). As the time complexity of Algorithm 2 is $O(I \cdot (\log \bar{\pi} + \bar{d}))$, the time complexity of Algorithm 3 is $O(2 \cdot \eta \cdot I \cdot (\log \bar{\pi} + \bar{d}))$, where $\bar{\pi}$ is the average width of balance ranges, and $\bar{d}$ is the average node degree. In practice, NIBT-RL is quite time-consuming in large-scale networks because of the following two factors: (1) Since a large-scale network has a large number of nodes, the algorithm requires a substantial number of episodes (i.e., a large I) to discover the optimal path. (2) As a large-scale network contains a large number of balances, the algorithm requires to make many payments (i.e., requires η to be large) to shrink all balance ranges to their minimum.

To speed up NIBT-RL, we design a fast version named NIBT-RL-β, which makes a tradeoff between accuracy and efficiency. The framework of NIBT-RL-β is illustrated in Figure 6(b). In NIBT-RL-β, we define a batch parameter β to control the number of paths selected from the constructed paths in each round. Then, multiple payments are conducted sequentially on the β paths to shrink the ranges of the balances covered by these paths to their minimum. When these balance ranges become small enough, NIBT-RL-β updates the system and calls *RLPath* for the next round of path construction. As the frequency of invoking *RLPath* is greatly reduced in NIBT-RL-β, the running time is significantly reduced. Our experimental results in Section 4.2 demonstrate that NIBT-RL-β has a much faster running speed than NIBT-RL with only a slight loss on inference accuracy.

Note that limiting the total number of payments (i.e., payment budget η) in NIBT-RL-β is no longer effective for controlling the stopping time of the algorithm, because the payments have been divided into groups to cut down the balance ranges on β paths in each round. Nevertheless,

ALGORITHM 4: Non-intrusive Balance Tomography Using RL with Batch β (NIBT-RL-β)

Input: $G(\mathbf{N}, \mathbf{E})$, $\mathbf{C}$, M_1, M_2, λ, β, I, δ
Output: $\mathbf{U}$, $\mathbf{L}$

1 $\mathbf{U} \leftarrow \mathbf{C}$
2 $\mathbf{L} \leftarrow 0$
3 $\mathbf{pathSet}_1 \leftarrow RLPath(G(\mathbf{N}, \mathbf{E}), \mathbf{U}, \mathbf{L}, M_1, M_2, \lambda, I)$
4 $\mathbf{pathSet}_2 \leftarrow RLPath(G(\mathbf{N}, \mathbf{E}), \mathbf{U}, \mathbf{L}, M_2, M_1, \lambda, I)$
5 $\mathbf{P} \leftarrow topMaxGain(\mathbf{pathSet}_1, \mathbf{pathSet}_2, \beta)$
6 **while** $maxGain(\mathbf{P}) > \delta$ **do**
7 $Gain^* \leftarrow maxGain(\mathbf{P})$
8 **while** $Gain^* > \delta$ **do**
9 **for** $p_i \in \mathbf{P}$ **do**
10 $[m^{*i}, Gain^{*i}] \leftarrow optPayAmt(p_i, \mathbf{U^i}, \mathbf{L^i})$
11 $[p_{i*}, m^{*i}, Gain^*] \leftarrow optPay(Gain^{*1}, \ldots, Gain^{*n})$
12 $Result \leftarrow conductPayment(p_{i*}, m^{*i})$
13 $[\mathbf{L}, \mathbf{U}] \leftarrow updateRange(p_{i*}, m^{*i}, Result)$
14 $\mathbf{pathSet}_1 \leftarrow RLPath(G(\mathbf{N}, \mathbf{E}), \mathbf{U}, \mathbf{L}, M_1, M_2, \lambda, I)$
15 $\mathbf{pathSet}_2 \leftarrow RLPath(G(\mathbf{N}, \mathbf{E}), \mathbf{U}, \mathbf{L}, M_2, M_1, \lambda, I)$
16 $\mathbf{P} \leftarrow topMaxGain(\mathbf{pathSet}_1, \mathbf{pathSet}_2, \beta)$
17 **return** $\mathbf{L}$, $\mathbf{U}$

we can use an alternative threshold δ to control the stopping time. Specifically, the algorithm will stop shrinking down the balance ranges once the maximum *gain* is smaller than δ. Obviously, the number of required payments is inversely proportional to the threshold δ.

The pseudo-code of NIBT-RL-β is given in Algorithm 4. The algorithm picks the top β paths from $\mathbf{pathSet}_1$ and $\mathbf{pathSet}_2$ (line 5) and shrinks the ranges of balances on these paths in a *while* loop until the maximum *gain* on these path is smaller than δ (line 8∼line 13). In the *while* loop, the algorithm first computes the maximum *gain* of each path by Algorithm 1 (line 10) and conducts the optimal payment on the path with the highest *gain* (line 11∼line 12). After observing the payment result, it updates the lower and upper bounds of the balances (line 13) and then proceeds to the next round. After the end of the *while* loop, Algorithm 4 calls *RLPath* to construct the next β paths (line 14∼line 16).

As the frequency of invoking *RLPath* is greatly reduced, its time complexity is much lower than Algorithm 3. Let $\omega(\delta)$ denote the number of iterations of the first *while* loop in Algorithm 4 and $\bar{\eta}(\delta)$ denote the average number of payments made in the second *while* loop. The time complexity of Algorithm 4 is $O\big(\omega(\delta) \cdot (\bar{\eta}_i(\delta) \cdot \beta \log \bar{\pi} + 2 \cdot I \cdot (\log \bar{\pi} + \bar{d}))\big)$. In a large-scale network, the time for constructing paths dominates the total time of NIBT-RL-β. Hence, for large-scale networks, the time complexity of Algorithm 4 is $O\big(\omega(\delta) \cdot 2 \cdot I \cdot (\log \bar{\pi} + \bar{d})\big)$. There is also a correlation between the batch parameter β and the number of iterations $\omega(\delta)$: If we turn up β, then $\omega(\delta)$ will drop, and the time complexity of NIBT-RL-β is reduced accordingly. However, as the value of β increases, the quality of the top-β paths cannot be guaranteed, and the accuracy of NIBT-RL-β may decline. In Section 4, we will evaluate NIBT-RL and NIBT-RL-β from different angles.

3.6 Parallel Mode

We propose a special form of parallel processing to accelerate the running speed of Algorithm 3 and Algorithm 4. Specifically, we divide the LN network into several sub-networks and infer balances simultaneously in each sub-network with separate pairs of attacking nodes.

Graph partitioning is one of the most common ways to enable parallel computation on large networks. In the past, numerous works have been proposed to partition the VERTICES of a network into roughly equal disjoint subsets while minimizing the edges crossing the subsets [2, 8, 11]. However, in our parallelization problem, we expect to partition the EDGES of the network (i.e., the channels in LN) into roughly equal disjoint sets, so the attacking workload can be balanced across the subsets. Recently, some edge partitioning methods have been proposed to partition the edges in a network as even as possible [7, 31, 41]. Among them, the **neighbor expansion (NE)** proposed in Reference [41] leads to the state-of-the-art partitioning results in terms of *replication factor* [7]. Particularly, suppose network $G(\mathbf{N}, \mathbf{E})$ is partitioned into ω parts: $G_1(\mathbf{N}_1, \mathbf{E}_1)$, $G_2(\mathbf{N}_2, \mathbf{E}_2), \ldots, G_\omega(\mathbf{N}_\omega, \mathbf{E}_\omega)$, where $\forall i \in \{1, 2, \ldots, \omega\}$, $\mathbf{N}_i \subseteq \mathbf{N}$, $\mathbf{E}_i \subseteq \mathbf{E}$, $\cup_i \mathbf{E}_i = \mathbf{E}$, and $\cup_i \mathbf{N}_i = \mathbf{N}$. $\forall i, j \in \{1, 2, \ldots, \omega\}$, $i \neq j$, $\mathbf{E}_i \cap \mathbf{E}_j = \phi$, $|\mathbf{E}_i| \approx |\mathbf{E}_j|$. The *replication factor* of the partitioning is defined by

$$RF = \frac{1}{|\mathbf{N}|} \sum_{i=1}^{\omega} |\mathbf{N}_i|. \tag{18}$$

NE starts with a random core node in each round and expands the core-node set by a variant breadth-first search until the number of edges in the set exceeds a predefined threshold. For each sub-network $G_i(\mathbf{N}_i, \mathbf{E}_i)$, we create a pair of accounts $M_1^{(i)}$ and $M_2^{(i)}$ to perform the balance inference. Obviously, the more parts we partition the LN, the more cost we need to spend on opening the accounts. Hence, there exists a tradeoff between the cost and the running speed. In Section 4.2, we compare NE with an alternative edge partitioning method and vary the number of sub-networks ω to evaluate the performance of our approach in the parallel mode.

4 PERFORMANCE EVALUATION

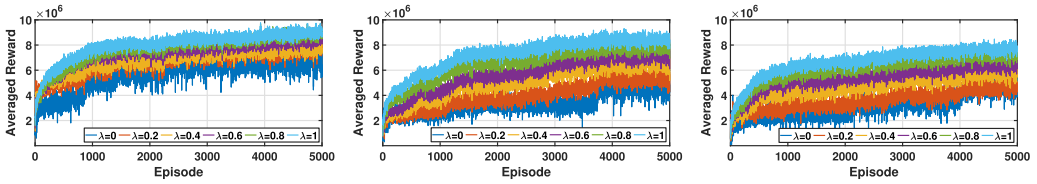
4.1 Experimental Settings

4.1.1 Lightning Topology. We generated the LN topology using a snapshot of LN taken on July 5, 2020. This snapshot includes 6,072 nodes and 30,026 payment channels. Our experiments were carried out on three subsets of the whole network: (1) a small network with 513 nodes and 8,474 balances; (2) a medium network with 1,354 nodes and 20,916 balances; (3) a large network with 3,732 nodes and 55,428 balances. To evaluate the performance of our method under different snapshots, we also carried out our experiments on five more snapshots of LN that are taken from December 2019 to March 2020.

4.1.2 Capacities and Balances. We generate the capacities of channels based on the public LN statistics [6], where the upper bound of the capacity is 0.167 BTC. Once the capacity c_{ij} for channel e_{ij} is generated, the balance of one side b_{ij} is then drawn from $[0, c_{ij}]$ following uniform distribution. The balance on the other side is then set to $b_{ji} = c_{ij} - b_{ij}$.

4.1.3 Connecting Our Accounts to LN. In our preliminary work [27], the experimental results showed that connecting our two accounts to the LN nodes with the highest node degrees can lead to the best performance of NIBT over other connecting methods. In this section, we directly connect our two accounts to the two nodes that have the highest degrees and use *RLPath* to construct the end-to-end paths between our two accounts.

4.1.4 RL Model. In the RL model, the learning rate is set to 0.1, as our data shows that this value achieves the highest performance in our methods. For the ϵ-greedy policy in RL, we let $\epsilon = 0.7$. This means that the policy has 70% chance of choosing the action with the maximum Q value and 30% chance to explore a new path.



(a) Small network (b) Medium network (c) Large network

Fig. 7. Average reward over different episodes.

4.1.5 Payments. Payments are conducted sequentially: A new payment is made only after the result of the last payment is obtained. According to LN specification, the maximum amount in one transaction is 0.043 BTC. Therefore, when the conducted payment amount exceeds this value, we need to divide the payment into several sub-payments, each of which carries an amount of at most 0.043 BTC.

4.1.6 Estimation of Cost. We estimate the cost for opening and closing one channel as 1.72×10^4 Satoshis ($\approx$ 1.53 USD) according to the average transaction fee from February 25, 2020, to May 25, 2020 [5]. The routing fee charged by intermediate nodes includes two parts: a constant base fee that nodes charge per transfer and a flexible additional fee that is proportional to the transferred amount. For LN nodes, the default values for base fee and additional fee are 1,000 msat (10^{-3} Satoshi) and 1 msat (10^{-6} Satoshi) per transferred Satoshi. As reported in Reference [21], 98% and 65% of LN nodes set their base and additional charges, respectively, equal to or smaller than the default values. Hence, without a risk of underestimating the cost, we assume that all channels use the default values as their base and additional charges for transfers.

The source codes of our methods with the detailed experimental settings and LN snapshots are available at https://github.com/duoduoqiao/NIBT-RL.

4.2 Results

Figure 7 plots the average rewards over different RL episodes ranging from 1 to 5,000. As shown in the figure, *RLPath* can discover more informative paths as the number of episodes increases. Figure 7 compares the average reward in each episode under different value of λ. When $\lambda = 0$, *RLPath* will not attach any reward to the previous steps except of the last step of the episode; when $\lambda = 1$, all previous steps will be issued an equal reward. As we can see in Figure 7, the average reward in each episode increases as λ increases. This implies that earlier steps have the same importance as the last step in each episode in our path construction. Comparing the curves in different scales of networks, we can see that within the same episodes, *RLPath* can obtain the highest reward in small network. This means that larger networks require more episodes to discover optimal paths. In the following experiments, the value of λ is set to 1, and the number of episodes is set to 2,000.

Table 2 compares the performance of NIBT-RL and NIBT-RL-β, where β ranges from 50 to 500. We did not limit the budget η or threshold δ in the two algorithms. Instead, we only stopped when no payments could cut down more balance ranges. We separated balances into two groups: (1) small balances that are smaller than 0.043×10^8 ($\approx$ 382.6 USD), taking up about 89.8% of all balances and (2) large balances that are larger than 0.043×10^8, taking up about 10.2% of all balances. Note that the range of a balance equals the upper bound minus the lower bound of the balance. When the range of a balance is smaller than 10 Satoshis ($\approx$ 0.00089 USD), we say the balance is determined. The more balances determined in a method, the more accurate the method is. Table 2 shows that: (1) As the scale of network increases, the accuracy slightly drops. Furthermore, the

Table 2. Performance of NIBT-RL and NIBT-RL-β in Different Scale of Networks

Networks	Methods	Average Path Length	Fraction of Determined Balances			Number of Required Payments	Routing Fees (USD)	Computational Time (Seconds)
			Total	Small balances	Large balances			
Small Network (513 nodes 8,474 balances)	NIBT-RL	11.22	94.12%	98.12%	59.61%	64,899	3.29	4.54×10^5
	NIBT-RL-50	9.10	92.23%	97.12%	52.87%	64,403	2.86	3.04×10^3
	NIBT-RL-100	8.72	92.03%	96.78%	51.12%	64,343	2.58	2.33×10^3
	NIBT-RL-200	8.11	91.97%	96.97%	52.69%	64,723	2.46	2.67×10^3
	NIBT-RL-300	8.01	92.02%	96.95%	50.94%	65,055	2.16	1.89×10^3
	NIBT-RL-500	7.48	91.67%	96.68%	50.69%	64,837	2.06	1.76×10^3
Medium Network (1,354 nodes 20,916 balances)	NIBT-RL	11.78	94.10%	98.45%	58.03%	160,948	7.44	1.29×10^6
	NIBT-RL-50	9.61	91.66%	96.78%	49.37%	157,412	4.69	1.06×10^4
	NIBT-RL-100	9.13	91.60%	96.52%	49.47%	158,588	4.47	7.80×10^3
	NIBT-RL-200	8.56	91.33%	96.47%	48.92%	159,355	4.58	7.02×10^3
	NIBT-RL-300	8.41	91.44%	96.42%	48.81%	159,577	4.48	6.36×10^3
	NIBT-RL-500	8.14	91.11%	96.40%	47.57%	159,668	4.29	6.20×10^3
Large Network (3,732 nodes 55,428 balances)	NIBT-RL	12.31	93.37%	97.92%	55.29%	425,293	16.04	4.89×10^6
	NIBT-RL-50	9.99	91.43%	96.36%	50.28%	416,714	11.10	4.13×10^4
	NIBT-RL-100	9.46	91.13%	96.21%	48.15%	417,297	10.13	3.27×10^4
	NIBT-RL-200	8.93	90.99%	96.16%	47.98%	418,938	8.85	2.82×10^4
	NIBT-RL-300	8.65	90.63%	96.05%	45.65%	418,098	8.73	2.63×10^4
	NIBT-RL-500	8.42	90.61%	96.04%	45.72%	419,439	8.53	2.60×10^4

number of required payments, the routing fees, and the attacking time grow (roughly) linearly with the number of balances in the topology; (2) NIBT-RL can accurately infer more balances than NIBT-RL-β by 2% ~ 3%. However, its routing fee is 15% ~ 88% higher, and its attacking time is 2 orders of magnitude higher than NIBT-RL-β; (3) As the batch parameter β increases, the accuracy slightly drops, and the number of required payments goes up. This indicates that turning up β may affect the overall quality of the constructed paths. However, as the batch size rises, shorter paths have more chance to be selected to carry the payments (long paths are generally more informative than shorter paths). Therefore, as β is turned up, the average path length slightly drops, and the total routing fee declines accordingly. Moreover, when we increase β from 50 to 500, the computing time is reduced by at least 40%.

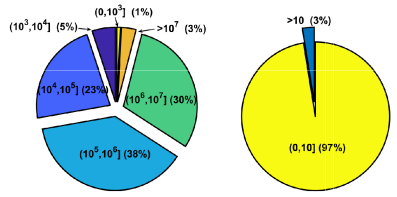
To verify the performance of our method on different LN snapshots, we apply NIBT-RL-300 on five more snapshots of LN from December 2019 to March 2020 in Table 3. The results on the five snapshots are similar with the results in Table 2. Particularly, the fluctuation is within 1% in terms of the total determined balances. The results demonstrate that our method performs quite stably in different network environments.

In the following, we evaluate different aspects of NIBT-RL-300 in the small network in Table 2. All results are averaged over 20 runs.

Figure 8 and Figure 9 plot the distributions of the ranges of balances in the small and large groups, respectively. Each slice in the pies denotes the proportion of the marked ranges. For example, $(0, 10^3](1\%)$ in Figure 8(a) means that 1% of balance ranges are between 0 and 10^3. From Figure 8(a), 94% of the original balance ranges are larger than 10^4 Satoshis ($\approx$ 0.8898 USD), and 72% are larger than 10^5 Satoshis ($\approx$ 8.898 USD) before we perform balance inference. After inference, 97% of balances have the ranges smaller than 10 Satoshis ($\approx$ 0.00089 USD). From Figure 9(a), all balance

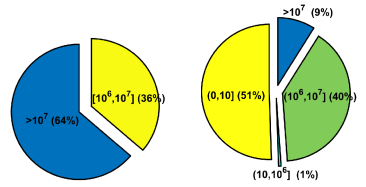
Table 3. Performance of NIBT-RL-300 in Different LN Snapshots

LN Snapshots	Network Size	Average Path Length	Fraction of Determined Balances			Number of Required Payments	Routing Fees (USD)	Computational Time (Seconds)
			Total	Small balances	Large balances			
Snapshot 1 Dec. 10, 2019	3,298 nodes 51,152 channels	8.63	90.56%	95.67%	46.13%	384,324	7.52	1.48×10^4
Snapshot 2 Dec. 26, 2019	3,269 nodes 51,638 channels	8.54	90.98%	96.19%	45.85%	391,137	8.13	1.64×10^4
Snapshot 3 Jan. 8, 2020	3,250 nodes 52,998 channels	8.61	90.69%	96.06%	46.27%	393,225	8.46	1.86×10^4
Snapshot 4 Feb. 5, 2020	3,323 nodes 53,372 channels	8.56	91.53%	96.63%	48.80%	402,198	8.74	1.76×10^4
Snapshot 5 Mar. 2, 2020	3,422 nodes 53,056 channels	8.76	90.77%	96.94%	46.69%	404,534	8.53	2.19×10^4



(a) Ranges of balances before inference. (b) Ranges of balances after inference.

Fig. 8. Inferring the ranges of small balances.



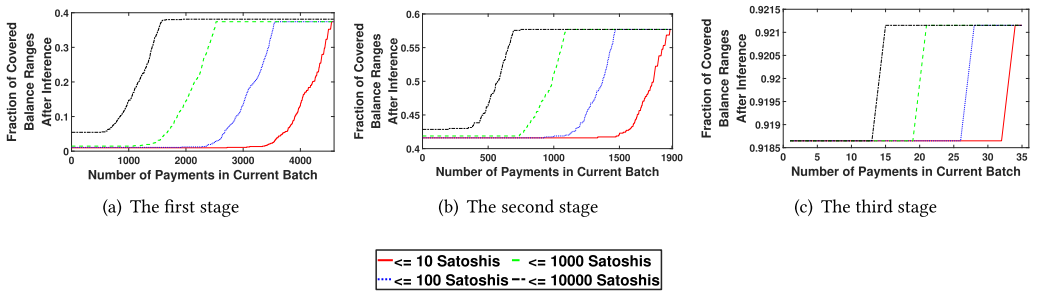
(a) Ranges of balances before inference. (b) Ranges of balances after inference.

Fig. 9. Inferring the ranges of large balances.

ranges are larger than 4.3×10^6 ($\approx$ 382.6 USD), 64% of them are larger than 10^7 ($\approx$ 889.82 USD). After inference, 51% of them have the ranges smaller than 10 Satoshis ($\approx$ 0.00089 USD) and only 9% of them are larger than 10^7 Satoshis ($\approx$ 889.82 USD).

Remark 6. The results shown in Figure 8 and Figure 9 are similar with the experimental results shown in Figure 5 and Figure 6 in Reference [27]. Actually, the results in Reference [27] showed the performance of NIBT under the best candidate path set (candidate-1: contains top-200 shortest paths between the top-two-degree nodes), while Figure 8 and Figure 9 showed the average performance of NIBT-RL-300 without any prerequisite. In other words, the results in Reference [27] demonstrated that NIBT can work well when all routing paths are short enough (averagely 2.14 hops), while the results in Figure 8 and Figure 9 showed two new pieces of information: (1) RL can effectively find the optimal payment paths for the attack; (2) if the paths are good enough, then the attack can still perform well even when the average length of the payment paths is as long as 8.01.

Figure 10 plots the inference performance of NIBT-RL-300 in three stages under different number of payments in one running case. We limit the number of payments by controlling the threshold δ in Algorithm 4. The first stage (shown in Figure 10(a)) is NIBT-RL-300 inferring the ranges of balances covered by the first batch of paths output by *RLPath*; the second stage (shown in Figure 10(b)) is inferring the balances covered by the 45th batch of paths; the third stage (shown in Figure 10(c)) is the inference for the 93rd batch of paths (i.e., the last batch). In the first stage, when 1,600 payments were made, the ranges of about 38% of the covered balances can be reduced to 10,000 Satoshis. When the budget is increased to accommodate 4,500 payments, the range of the covered balances could be reduced to 10 Satoshis. As the stage moves forward, more batches of paths are



(a) The first stage (b) The second stage (c) The third stage

Fig. 10. Accuracy of NIBT-RL-300 in different inference stages: There are totally 93 batches of paths output by *RLPath*. The first stage is inferring the balances on the first batch of paths, while the second and the third stages are the inferences for the 45th and 93rd batch, respectively.

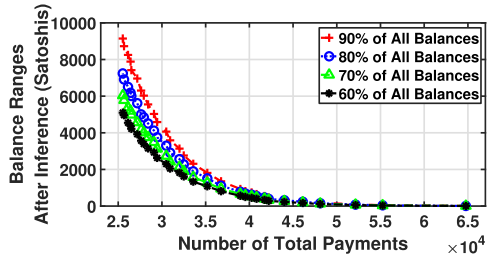
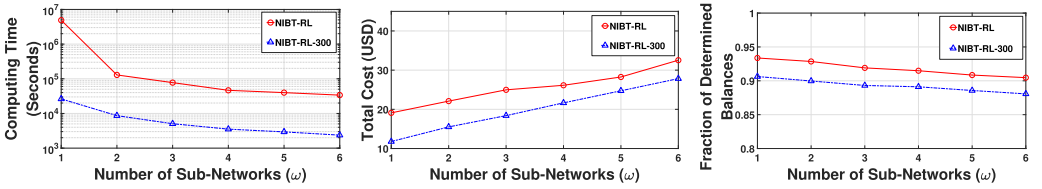


Fig. 11. The performance of NIBT-RL-300 under different number of total payments.



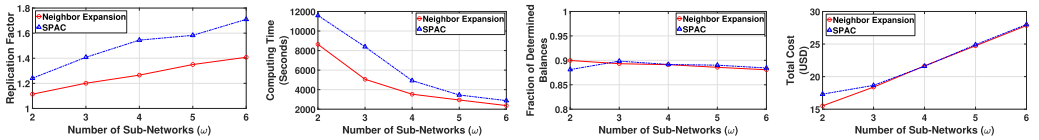
(a) Computing time (b) Total cost (c) Accuracy

Fig. 12. The performance of NIBT-RL and NIBT-RL-300 in parallel mode.

involved, and the fraction of determined balances gradually increases. Nevertheless, the tendencies of performance in these stages are highly similar. This tendency reveals that NIBT-RL-300 always shrinks large ranges first. Therefore, the number of required payments can be significantly reduced at the cost of a small loss in accuracy.

Figure 11 shows the correlation between the inference accuracy and the total number of payments used in NIBT-RL-300. From the figure, if we conduct only 25,000 payments in total, then 90% of all balance ranges can be shrunk to less than 10,000 Satoshis; 60% of the balance ranges are less than 5,500 Satoshis. When we tighten up the threshold δ and let the number of payments grow to 35,000, 90% of the ranges will be less than 2,000 Satoshis, and 60% will be less than 1,000 Satoshis; when the number of payments reaches 65,000, at least 90% of the balance ranges will be smaller than 10 Satoshis.

The performance of NIBT-RL and NIBT-RL-300 in parallel mode is shown in Figure 12 ($\omega = 1$ means the original LN topology before partition). In Figure 12(a), each recorded computing time is equal to the maximum running time of NIBT-RL/NIBT-RL-300 over all partitions. From the figure, when we use four accounts to perform the balance inference in two sub-networks simultaneously,



(a) Replication factor (b) Computing time (c) Accuracy (d) Cost

Fig. 13. Comparing the effects of graph partitioning methods on parallelization.

the computing time of NIBT-RL can be reduced by more than one order of magnitude. When LN is partitioned into 6 components, NIBT-RL-300 and NIBT-RL can be sped up by one and two orders of magnitude, respectively. In Figure 12(b), the total cost of the two methods grows near-linearly with the number of partitions because one more partition requires two more attacker's accounts to perform the inference in parallel, leading to two additional fees for channel opening. It is noteworthy that the number of balances that could be accurately determined declines slightly with the increase of the number of partitions (as shown in Figure 12(c)); that is because some optimal paths that exist in LN before partitioning may no longer exist in the sub-networks. Nevertheless, even when LN is partitioned into 6 sub-networks, the fraction of determined balances can also achieve at least 90% and 88% for NIBT-RL and NIBT-RL-300, respectively.

Aside from Neighbor Expansion, we also applied an alternative graph partitioning method, named **Split and Connect (SPAC)** [31], to compare the effects of different graph partitioning methods on parallelizing our method. Figure 13(a) compares the *replication factor* (see Equation (18)) of the two graph partitioning methods, and Figure 13(b), Figure 13(c), and Figure 13(d) compare the computing time, the accuracy, and the cost of NIBT-RL-300, respectively, under the two partitioning methods. From the figures, Neighbor Expansion has a lower replication factor than SPAC, which means Neighbor Expansion can partition the graph more evenly. As the subsets partitioned by SPAC are not as even as Neighbor Expansion, the computing time in the relative larger subset will delay the overall attacking process. As a result, NIBT-RL-300 based on SPAC requires more computing time than that based on Neighbor Expansion. Nevertheless, NIBT-RL-300 has a similar performance in terms of accuracy and cost under the two partitioning methods, which also demonstrates the stability of our method with different LN topologies.

In summary, (1) although *RLPath* only uses 2,000 episodes, NIBT-RL can accurately infer 93%∼94% of all balances, and NIBT-RL-β can determine 90%∼92% of all the balances. NIBT-RL-β reduces NIBT-RL's computing time by 2 orders of magnitude at the cost of 2% ∼ 3% loss in accuracy. (2) A larger batch size may slightly harm the quality of paths but in return can reduce routing fees and computation time. (3) Using NIBT-RL-300 as an example, we demonstrate that NIBT-RL-β can accurately infer more than 97% of small balances (smaller than the maximum payment amount in one transaction) and nearly 51% of large balances (larger than the maximum payment amount in one transaction). (4) There is a tradeoff between the inference accuracy and the payment budget: Relaxing the requirements on accuracy can significantly reduce the payment budget. (5) We can make a tradeoff between the cost and efficiency using the parallel mode of NIBT-RL and NIBT-RL-300. For example, if we spend about 3 USD more on opening two additional accounts to perform the inference, then the running time of NIBT-RL can be reduced by more than one order of magnitude, and the running time of NIBT-RL-300 can be reduced by about 70%. Nevertheless, the partition of LN may slightly harm the accuracy of inference (about 0.5%∼0.6% decrease).

4.3 Comparison with the State-of-the-art

Although our attacking method is, *in principle*, different from the existing balance disclosure attacks, we still compare our method with the state-of-the-art from different angles to provide a

Table 4. Comprehensive Comparison with the State-of-the-art Work

Methods/ Metrics	DM [15]/Improved DM [37]/ Active Probing [23]	Torrent[28]	Generic attack [16]	Parallel Probing [3]	NIBT-RL/ NIBT-RL-β
Type of Payments	F	F	F	F	L
Type of Probing	D	R	D	R&D	R
Probing Path	S	M	S	M	M
#of Targets	1	1	1	1	All
Success Rate	89%~98%	87%~97%	100%	80%~90%	91%~94%
Required Accounts (For all balances)	$\|\mathbf{N}\|$	1 or 2	$\|\mathbf{N}\| \times 2$	1 or $\|\mathbf{N}\|$	2
Attacking Time (For all balances)	1~2 months	1~2 weeks	1~2 months	1~2 weeks	7 hours

Notations: F: Fake payments, L: Legal payments, D: Direct probing, R: Remote probing, S: Single path, M: Multiple paths.

better view on the cons/pros. Table 4 lists the main features of our attacking method and those of six state-of-the-art balance disclosure attacks. As the DM [15], the Improved DM [37], and the Active Probing [16] use the same attacking strategy, we group them into one category. From the table, our attacking method presents two distinctive features: (1) Our attack is designed for discovering all balances in the Lightning topology, while the former attacks are designed for attacking one target balance. (2) Our attack is performed with legal payments only while all former attacks use fake payments. Among existing attacks, Torrent and Parallel Probing use remote probing method, while other methods use direct probing method. The advantage of remote attacking is that the attackers do not need to connect to the target channel directly, so it only needs to open one account to attack different victims through intermediate channels. However, the main drawback is that the target balance may be blocked by some intermediate channels that have low balances (i.e., bottleneck). Torrent and Parallel Probing apply multi-path probing strategies to reduce the impact of the bottleneck channel on single path. All direct attacks need to open $|\mathbf{N}|$ accounts to discover all balances in the topology. As Generic attack opens two channel accounts for each target (an outgoing channel and an incoming channel), it can theoretically attack all balances, including the large ones. However, regardless of the cost on opening double number of accounts, creating an incoming account to connect every individual channel is quite difficult to implement. The attacking time for remote and multi-path probing is much shorter than the direct and single-path probing methods. On the one hand, the parallel probing strategy can considerably accelerate the attacking process. On the other hand, the remote probing can save a lot of time on opening and closing the attacking accounts. In conclusion, compared with the state-of-the-art, our attacking method is economical, efficient, and completely non-intrusive.

We further compare the detailed implementation of our attacking method with that of the two representative methods, DM [15] and Torrent [28], in the large LN network. Table 5 presents a summary of the comparison results, where the results of NIBT-RL-β are averaged over NIBT-RL-50 ~ NIBT-RL-500. From the table, DM needs to conduct, on average, 14 fake payments to accurately discover a balance. About half of these payments cause the error message of "unknown payment hash." The other half of the payments fail due to insufficient balance. As Torrent carries payments through multiple paths to attack one target balance, it requires $6 \sim 7$ times of payments than DM. NIBT-RL-β, however, creates no error messages. Moreover, it requires to conduct less than 8 payments, on average, to infer a balance, where about 6.67 of the payments fail because of "insufficient balances" and about 1.76 of them are successfully fulfilled.[3] As DM connects the target channels

[3]The number of successful payments includes both the actual and reverse payments.

Table 5. Compare NIBT-RL-β with Two Representative Methods on Implementation Details

Methods/ Metrics	NIBT-RL-β	DM	Torrent
Error payments per balance (Due to fake invoices)	0	7.07	43.85
Failed payments per balance (Due to insufficient Funds)	6.67	7.02	51.94
Successful payments per balance	1.76	0	0
Prob. to disclosure small balances	96%	100%	95.89%
Prob. to disclosure large balances	47%	0	86.57%
Total Cost (USD)	12.26	5709.96	1.53

directly, it can accurately discover all balances that are smaller than the maximum payment allowed in one transaction. Torrent and NIBT-RL-β, however, probe the target channels remotely. There exists a chance that the target balances are blocked by the low balances on the routing paths. DM cannot infer the balances larger than the maximum payment allowed in one transaction, because payment recipients do not hold fake payments. NIBT-RL-β can infer about 47% of these balances, because the recipients in NIBT-RL-β can hold multiple successful payments. As Torrent carries out a large amount of payments simultaneously to attack the target channel with the maximum flow strategy, it has a relative higher chance to disclose a large balance channel. The total cost for NIBT-RL-β includes the fees for opening/closing channels and the fees for transferring successful payments. Since we only need to open two accounts in total and the transaction fees collected by intermediate nodes are extremely low, the total cost is only 12.26 USD for inferring more than 50,000 balances. As all payments in Torrent and DM are fake payments, their costs only include the fees for opening/closing channels. As Torrent can attack the balances remotely, it only needs to open one account, costing about 1.53 USD. DM can only measure the balances of node that is directly connected with attacker's account. In the large LN network, the number of nodes that DM needs to connect is 3,732, which costs 5,709.96 USD in total.

4.4 Skewed Channels Detection in LN System

A skewed channel means one balance in the channel is much higher than the other, which is generally formed when more payments flow from one direction of the channel than the other direction. If such a situation lasts for a long time, then one balance in the channel will become zero, which means no payment flow can go through from this direction. Skewed channels seriously hinder the liquidity of LN and cause other concerns, as stated in References [9, 32]. In this regard, "attackers" who would like to detect the balances of channels may not necessarily have malicious purpose. Instead, they may use the balance information for the social well-being, e.g., solving the unbalanced channel problem [9]. In this section, we show the performance of NIBT-RL-300 on discovering the skewed channels in LN system.

Figure 14 presents the number of all skewed channels in the topology and the number of skewed channels that can be detected during the inference. The *degree of the skewness* is defined as the ratio between the smaller balance over the capacity of the channel. For example, 0.1 means the balance of one side of the channel is 10% of the channel capacity, while the balance on the other side accounts for the remaining 90%. As shown in the figure, the majority of skewed channels can be

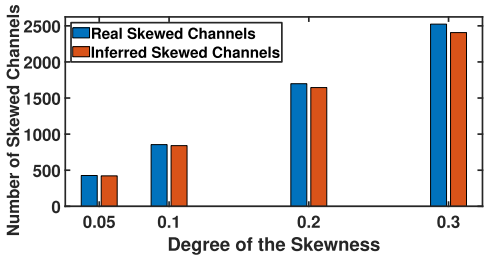
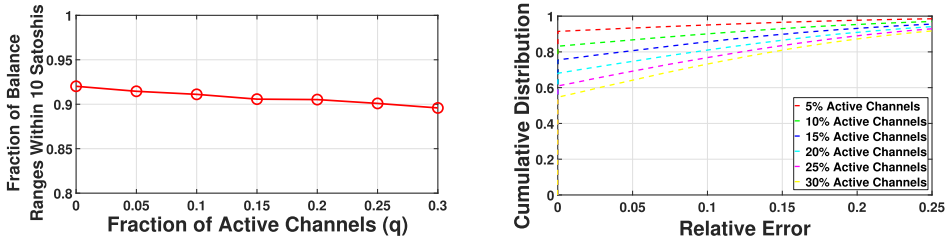


Fig. 14. The number of skewed channels that were detected: The *degree of the skewness* is defined as the ratio between the smaller balance over the capacity of the channel.



(a) Accuracy (b) Relative errors

Fig. 15. The performance of NIBT-RL-300 in dynamic environment.

detected during the balance inference. In particular, our method can detect 95% slightly skewed balances (i.e., those with skewness of 0.3) and 99% of seriously skewed channels (i.e., those with skewness of 0.05).

4.5 Performance in Dynamic Environment

So far, all existing attacks for balance disclosure assume that the inferred balances must keep stable during the attack period. Otherwise, the results of the previous probing will be useless, and the attacks need to be re-carried from the beginning. As the probing payments in our attacks are designed for a group of balances instead of a single balance, the previous results can provide partial information even if one of the traversed balances changed. Besides, as RL has the ability to interact with the system, our attack can sense the changes in the balances and adjust the policy of path construction. To verify the performance of our attack in a dynamic environment, we let q percent of all channels become highly active and let the rest of the channels keep stable. In each probing round, we allow all active channels randomly adjust their balances. During the attack period, once we find a balance of a channel that was inconsistent with its previously inferred range, the range will be updated by the latest result, and this channel will be marked as an active channel. The reward policy of our methods in the dynamic environment is modified as follows: The gains from active channels will no longer contribute to the reward of the path. In other words, the paths that cover active channels will have a lower priority of being probed. As NIBT-RL and NIBT-RL-β have similar performance in the dynamic environment in our experiments, we only show the results of NIBT-RL-300 in Figure 15.

Figure 15(a) plots the fraction of balance ranges within 10 Satoshis after inference. $q = 0$ means that all channels are stable. As we can see from the figure, the increased number of active channels has little impact on the final lengths of the balance ranges after inference, which means, although active balances have a lower priority, their balance ranges can still be cut down to their

minimum. Figure 15(b) shows the relative error[4] of the inferred ranges. From Figure 15(b), as the fraction of active channels grows, the number of balances that can be accurately inferred drops accordingly. Nevertheless, as our methods always update the balance ranges to their latest states, the deviations of the inferred ranges are not significant. Specifically, when there are 5% active channels, the relative errors of about 96% balances are less than 0.05. Even when the fraction of active channels goes up to 30%, more than 80% of the inferred balances have less than 0.15 relative errors.

5 DISCUSSION

In this section, we first discuss the potential countermeasures that LN may apply to prevent from our attack. Then, we discuss the possibilities of utilizing our attacking method as a measurement tool to benefit the efficiency of LN.

5.1 Countermeasures

Different from existing balance disclosure methods that use fake payments, all payments in our attack are legal. Therefore, the general countermeasures that detect the attacks by a flood of cancelled payments [3] cannot work for our attack. According to the implementation of our attacking method, we provide two countermeasures for LN to defend against our attack.

The first countermeasure is to modify the error messages in LN. In our attack, the optimal payments and the probing results are computed based on the critical condition that we can know where the payments were failed from the error messages, since the returned error messages that indicate insufficient funds contain the address/ID information of the failed channels. Hence, we suggest that if a channel is not able to transfer a payment due to insufficient balance, then LN will deliver an error message without any address/ID information on that channel. When the attackers do not know where the payment may fail, it is impossible for them to discover the accurate balances of the most LN channels with our attack.

The second countermeasure is to apply a random routing to the payments that were specified by the intermediate channels instead of the source routing pre-specified by the senders. By doing this, the attackers can hardly make the designed payments on the optimal paths constructed by our algorithms, which will cause significant growth of probing budget. In addition, the LN can further hide the actual routing from the senders, which will crack the multi-path attacks (including ours) that probe the target balances through collaborative multi-path probing. However, making these modifications to the current routing policy in LN requires to re-design the pricing scheme and the onion routing protocol.

5.2 Benefiting the Functionality of LN

Although our attack is designed for disclosing all balances in LN topology, it can also be used as a measurement tool to benefit an efficient routing, since all payments are legal and all participant nodes have gained incentives. Suppose a user wants to make a large payment in a transaction. Commonly, the payment is likely to fail due to the insufficient balance on the routing path. With our balance disclosure method, the user can first form a sub-network that covers the recipient and perform our method on the sub-network to disclose all balances. Then, the user can quickly find a feasible routing path to carry the large transaction, if any. There exists a tradeoff between the time/financial cost and the success rate of finding a feasible path. If the sub-network formed

[4]Let $\hat{b}_i$ denote the mean value of an inferred balance range, and let b_i denote the true value of the corresponding balance. The relative error is computed by $\frac{|b_i - \hat{b}_i|}{b_i}$.

by the user has a small scale, then our method can quickly discover all balances at a very low cost, whereas the success rate will be relatively low as well. Otherwise, if the user performs our method on an extended sub-network, then the success rate can be elevated, whereas the user needs to wait for a relative longer time for the discovered results as well as to pay a little higher transaction fee.

6 CONCLUSION

We pioneered *balance tomography* attack for inferring balances of LN without any interference or fake payments. Our attack first opens two accounts each opening a channel in LN. It then conducts multiple legitimate payments sequentially between the two accounts. To optimize each payment, we proposed an algorithm that can efficiently find the optimal amount on each path with logarithmic complexity. In addition, we developed an RL-based path construction algorithm that can construct the most informative path between our two accounts. Finally, we designed two balance inference approaches, NIBT-RL and NIBT-RL-β, to infer all balances by conducting the optimal payments on the constructed paths. We evaluated our attack using a snapshot of the LN Network. Experimental results show that 90% ∼ 94% of all balances can be accurately inferred with a small cost.

APPENDIX

Proposition 2. *$Gain^i(m)$ is a strictly concave function with only one maxima on its positive intervals.*

Proof. A detailed proof is lengthy and tedious, so we only provide the critical steps.

Suppose $\lambda_j^i = (\underline{\lambda_j^i}, \overline{\lambda_j^i})$ is a *positive interval* of $G^i(m)$. That is, there exists some non-empty set $A_j^i \subseteq \{1, 2, \ldots, n_i\}$, such that $\forall k, m : k \in A_j^i, m \in \lambda_j^i, H_k^i(m) > 0$. On λ_j^i, the first derivative of $G^i(m)$ with respect to m is

$$\frac{\mathrm{d}G^i(m)}{\mathrm{d}m} = \sum_{k \in A_j^i} a_k^i \cdot \left(\frac{1}{m - l_k^i} - \sum_{s=1}^{k} \frac{1}{u_s^i - m} \right) \cdot \left(m - l_k^i\right) \cdot \prod_{j=1}^{k} \left(u_j^i - m\right), \tag{19}$$

where $a_k^i = \frac{2}{\prod_{j=1}^{k}(u_j^i - l_j^i)}$. The second derivative of $G^i(m)$ with the respect to m is

$$\frac{\mathrm{d}^2G^i(m)}{\mathrm{d}m^2} = \sum_{k \in A_j^i} a_k^i \cdot \left(\sum_{s=1}^{k} \frac{m - l_k^i}{u_s^i - m} \cdot \left(-\frac{2}{m - l_k^i} + \sum_{1 \le t \le k, t \ne s} \frac{1}{u_t^i - m} \right) \cdot \prod_{j=1}^{k} (u_j^i - m) \right). \tag{20}$$

To prove that $G^i(m)$ is a strictly concave function, we need to show that the second derivative of $G^i(m)$ is negative. For this, we only need to consider the items $\sum_{1 \le t \le k, t \ne s} \frac{1}{u_t^i - m}$ and $\frac{2}{m - l_k^i}$ in

Equation (20), because all other items in Equation (20) are positive due to the fact that λ_j^i is a positive interval.

We expand λ_j^i to $(0,\overline{\lambda_j^i})$, then $\sum_{1\le t\le k, t\neq s}\frac{1}{u_t^i-m}$ is an increasing function of m from $\sum_{1\le t\le k, t\neq s}\frac{1}{u_t^i}$ to $\sum_{1\le t\le k, t\neq s}\frac{1}{u_t^i-\overline{\lambda_j^i}}$, while $\frac{2}{m-l_k^i}$ is a decreasing function[5] of m from $+\infty$ to $\frac{2}{\overline{\lambda_j^i}-l_k^i}$, there must be a point m_c such that when $m < m_c$, $\frac{\mathrm{d}^2G^i(m)}{\mathrm{d}m^2} < 0$.

Now, we prove that, $m_c \ge \overline{\lambda_j^i}$. To ease notation, we define a function $f(x)$ on domain $(b, \min_{i=1}^n a_i)$, where $b < \min_{i=1}^n a_i$, as

$$f(x) = -\frac{2}{x-b} + \sum_{i=1}^{n}\frac{1}{a_i - x}. \tag{21}$$

It is easy to prove that there exits a point x_c that $f(x) < 0$ when $x < x_c$, and $f(x) > 0$ when $x > x_c$, and x_c will reach its minimum value when both $a_i (1 \le i \le n)$ and b achieve their minimum values.

Based on the above analysis, for Equation (20), m_c will reach its minimum value when $\forall k : k \in A_j^i, u_k^i = \overline{\lambda_j^i}$ and $l_k^i = 0$. To compute the minimum value of m_c, we rewrite $G^i(m)$ by

$$\tilde{G}^i(m) = 2\cdot\left(\overline{\lambda_j^i} - m\right)\cdot\left(1 - \frac{(\overline{\lambda_j^i} - m)^{|A_j^i|}}{(\overline{\lambda_j^i})^{|A_j^i|}}\right), \tag{22}$$

where $|A_j^i|$ is the size of A_j^i. Then, the second derivative of $\tilde{G}^i(m)$ with respect to m is

$$\frac{\mathrm{d}^2\tilde{G}^i(m)}{\mathrm{d}m^2} = -2\cdot|A_j^i|\cdot(|A_j^i|+1)\cdot\frac{(\overline{\lambda_j^i} - m)^{|A_j^i|-1}}{(\overline{\lambda_j^i})^{|A_j^i|}}. \tag{23}$$

According to Equation (23), if $m < \overline{\lambda_j^i}$, $\frac{\mathrm{d}^2\tilde{G}^i(m)}{\mathrm{d}m^2} < 0$. Therefore, $m_c \ge \overline{\lambda_j^i}$. □

REFERENCES

[1] Marianna Belotti, Nikola Božić, Guy Pujolle, and Stefano Secci. 2019. A vademecum on blockchain technologies: When, which, and how. *IEEE Commun. Surv. Tutor.* 21, 4 (2019), 3796–3838.

[2] Charles-Edmond Bichot and Patrick Siarry. 2013. *Graph Partitioning*. John Wiley & Sons.

[3] Alex Biryukov, Gleb Naumenko, and Sergei Tikhomirov. 2022. Analysis and probing of parallel channels in the Lightning Network. In *International Conference on Financial Cryptography and Data Security*. Springer, 337–357.

[4] Alex Biryukov and Sergei Tikhomirov. 2019. Deanonymization and linkability of cryptocurrency transactions based on network analysis. In *IEEE European Symposium on Security and Privacy (EuroS&P'19)*. IEEE 172–184.

[5] Bitcoin Transaction Fee 2019. Bitcoin Transaction Fee historical chart. Retrieved from: https://bitinfocharts.com/en/comparison/bitcoin-transactionfees.html

[6] Bitcoin Visuals 2019. Bitcoin Visuals. Retrieved from https://bitcoinvisuals.com/lightning

[7] Florian Bourse, Marc Lelarge, and Milan Vojnovic. 2014. Balanced graph edge partition. In *20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1456–1465.

[8] Aydın Buluç, Henning Meyerhenke, Ilya Safro, Peter Sanders, and Christian Schulz. 2016. *Recent Advances in Graph Partitioning*. Springer.

[9] Marco Conoscenti, Antonio Vetrò, Juan Carlos De Martin, and Federico Spini. 2018. The CLoTH simulator for HTLC payment networks with introductory Lightning Network performance results. *Information* 9, 9 (2018), 223.

[10] Bradley Efron and Trevor Hastie. 2016. *Computer Age Statistical Inference*. Vol. 5. Cambridge University Press, UK.

[11] Wenfei Fan, Muyang Liu, Chao Tian, Ruiqi Xu, and Jingren Zhou. 2020. Incrementalization of graph partitioning algorithms. *Proc. VLDB Endow.* 13, 8 (2020), 1261–1274.

[5] In Equation (20), $\forall k : k \in A_j^i$, if $m < l_k^i$, $m - l_k^i = 0$.

[12] Michael Fleder, Michael S. Kester, and Sudeep Pillai. 2015. Bitcoin transaction graph analysis. *arXiv preprint arXiv:1502.01657* (2015).
[13] Arthur Gervais, Ghassan O. Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. 2016. On the security and performance of proof of work blockchains. In *ACM SIGSAC Conference on Computer and Communications Security*. Association for Computing Machinery, 3–16.
[14] David Goldschlag, Michael Reed, and Paul Syverson. 1999. Onion routing. *Commun. ACM* 42, 2 (1999), 39–41.
[15] Jordi Herrera-Joancomartí, Guillermo Navarro-Arribas, Alejandro Ranchal-Pedrosa, Cristina Pérez-Solà, and Joaquin Garcia-Alfaro. 2019. On the difficulty of hiding the balance of Lightning Network channels. In *ACM Asia Conference on Computer and Communications Security*. Association for Computing Machinery, 602–612.
[16] George Kappos, Haaroon Yousaf, Ania Piotrowska, Sanket Kanjalkar, Sergi Delgado-Segura, Andrew Miller, and Sarah Meiklejohn. 2021. An empirical analysis of privacy in the Lightning Network. In *International Conference on Financial Cryptography and Data Security*. Springer, Berlin, 167–186.
[17] Nida Khan and Radu State. 2019. Lightning network: A comparative review of transaction fees and data analysis. In *International Congress on Blockchain and Applications*. Springer, 11–18.
[18] Soohyeong Kim, Yongseok Kwon, and Sunghyun Cho. 2018. A survey of scalability solutions on blockchain. In *International Conference on Information and Communication Technology Convergence (ICTC'18)*. IEEE, 1204–1207.
[19] Dmitry Laptev. 2019. Solutions to inbound capacity problem in Lightning Network. Retrieved from medium.com/lightningto-me/practical-solutions-to-inbound-capacity-problem-in-lightning-network-60224aa13393
[20] Liang Ma, Ting He, Kin K. Leung, Don Towsley, and Ananthram Swami. 2013. Efficient identification of additive link metrics via network tomography. In *IEEE 33rd International Conference on Distributed Computing Systems*. IEEE, 581–590.
[21] Ayelet Mizrahi and Aviv Zohar. 2020. Congestion attacks in payment channel networks. *arXiv preprint arXiv: 2002.06564* (2020).
[22] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-peer electronic cash system. Retrieved from https://bitcoin.org/bitcoin.pdf
[23] Utz Nisslmueller, Klaus-Tycho Foerster, Stefan Schmid, and Christian Decker. 2020. Toward active and passive confidentiality attacks on cryptocurrency off-chain networks. *arXiv preprint arXiv:2003.00003* (2020).
[24] Cristina Pérez-Sola, Alejandro Ranchal-Pedrosa, Jordi Herrera-Joancomartí, Guillermo Navarro-Arribas, and Joaquin Garcia-Alfaro. 2020. Lockdown: Balance availability attack against Lightning Network channels. In *International Conference on Financial Cryptography and Data Security*. Springer, 245–263.
[25] Joseph Poon and Thaddeus Dryja. 2016. The Bitcoin Lightning Network: Scalable off-Chain Instant Payments. Retrieved from https://lightning.network/lightning-network-paper.pdf
[26] Pavel Prihodko, Slava Zhigulin, Mykola Sahno, Aleksei Ostrovskiy, and Olaoluwa Osuntokun. 2016. Flare: An approach to routing in Lightning Network. *White Paper* (2016). Retrieved from https://bitfury.com/content/downloads/whitepaper_flare_an_approach_to_routing_in_lightning_network_7_7_2016.pdf
[27] Yan Qiao, Kui Wu, and Majid Khabbazian. 2021. Non-intrusive and high-efficient balance tomography in the Lightning Network. In *ACM Asia Conference on Computer and Communications Security*. ACM, 832–843.
[28] Sonbol Rahimpour and Majid Khabbazian. 2022. Torrent: Strong, fast balance discovery in the Lightning Network. In *IEEE International Conference on Blockchain and Cryptocurrency (ICBC'22)*. IEEE, 1–7.
[29] Raiden 2020. Raiden Network. Retrieved from https://github.com/raiden-network/raiden
[30] Stefanie Roos, Pedro Moreno-Sanchez, Aniket Kate, and Ian Goldberg. 2017. Settling payments fast and private: Efficient decentralized routing for path-based transactions. *arXiv preprint arXiv:1709.05748* (2017).
[31] Sebastian Schlag, Christian Schulz, Daniel Seemaier, and Darren Strash. 2019. Scalable edge partitioning. In *21st Workshop on Algorithm Engineering and Experiments (ALENEX'19)*. SIAM, 211–225.
[32] Vibhaalakshmi Sivaraman, Shaileshh Bojja Venkatakrishnan, Kathleen Ruan, Parimarjan Negi, Lei Yang, Radhika Mittal, Giulia Fanti, and Mohammad Alizadeh. 2020. High throughput cryptocurrency routing in payment channel networks. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI'20)*. USENIX, 777–796.
[33] Richard S. Sutton and Andrew G. Barto. 2018. *Reinforcement Learning: An Introduction*. MIT Press.
[34] Thunder 2017. Thunder Network. Retrieved from https://github.com/blockchain/thunder
[35] Sergei Tikhomirov, Rene Pickhardt, Alex Biryukov, and Mariusz Nowostawski. 2020. Probing channel balances in the Lightning Network. *arXiv preprint arXiv:2004.00333* (2020).
[36] Manny Trillo. 2013. Stress test prepares VisaNet for the most wonderful time of the year. Retrieved from http://www.visa.com/blogarchives/us/2013/10/10/stress-testprepares-visanet-for-the-most-wonderful-time-of-the-year/index.html
[37] Gijs van Dam, Rabiah Abdul Kadir, Puteri N. E. Nohuddin, and Halimah Badioze Zaman. 2019. Improvements of the balance discovery attack on Lightning Network payment channels. *IACR Cryptol. ePrint Arch.* 2019 (2019), 1385.
[38] Yehuda Vardi. 1996. Network tomography: Estimating source-destination traffic intensities from link data. *J. Am. Stat. Assoc.* 91, 433 (1996), 365–377.

[39] Gavin Wood. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ether. Proj. Yell. Pap.* 151, 2014 (2014), 1–32.

[40] Ruozhou Yu, Guoliang Xue, Vishnu Teja Kilari, Dejun Yang, and Jian Tang. 2018. CoinExpress: A fast payment routing mechanism in blockchain-based payment channel networks. In *27th International Conference on Computer Communication and Networks (ICCCN'18)*. IEEE, 1–9.

[41] Chenzi Zhang, Fan Wei, Qin Liu, Zhihao Gavin Tang, and Zhenguo Li. 2017. Graph edge partitioning via neighborhood heuristic. In *23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 605–614.

Received 5 August 2022; revised 16 November 2023; accepted 20 December 2023