Torrent: Strong, Fast Balance Discovery in the Lightning Network

Sonbol Rahimpour, Majid Khabbazian
University of Alberta, Edmonton Canada
Email: {rahimpou, mkhabbazian}@ualberta.ca

***Abstract*—The owners of channels in the Bitcoin's Lightning Network do not disclose their balances in order to protect their privacy, and conceal payment transfers through their channels. Nevertheless, recent studies have shown that channel balances can be discovered using simple probing techniques. These techniques are, however, slow, often rely on specific control messages, and require to open a new channel for every single channel balance discovery or have limited power in discovering balances of remote channels. In this work, we present a powerful balance discovery method called Torrent that overcomes these limitations of existing methods. Unlike the existing techniques, Torrent uses multi-path payments instead of single-path payments. This—together with a novel max flow algorithm designed for the Lightning Network—allows a single probing node to push a large flow of payments through any target channel, making it more likely to discover its balance. Moreover, Torrent can speed-up balance discovery through parallel payment transfers, and pre-computation of payment paths. In addition, Torrent can operate in the absence of routing control messages that are often relied on by the existing channel discovery methods.**

***Index Terms*—Lightning network, balance discovery, privacy**

I. Introduction

The low throughput and high latency of Bitcoin transactions have inspired many researchers to look for scalable solutions. Today, one of the most promising scalable solutions designed for Bitcoin is the Lightning Network. Deployed in 2018, the Lightning Network creates a network of connected payment channels through which Bitcoin payments are transferred from one end to another. These payment transfers are carried off-chain and do not involve the Bitcoin blockchain. As a result, payments in the Lightning Network can be made at a significantly higher rate and speed than what the Bitcoin blockchain can handle.

In addition to enabling faster transactions, the Lightning Network offers lower transaction fees and higher privacy than the Bitcoin blockchain. To enhance privacy, the Lightning Network employs several mechanisms including *onion routing* and *balance concealment* [1]. The onion routing hides the sender and the recipient of a payment by encapsulating the payment information in layers of encryption. As a result, each node on the payment path can identify only its immediate predecessor and successor.

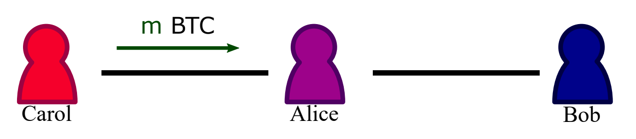
A successful payment will change the balance of all the channels on its path from the sender to the recipient of the payment. Therefore, if channel balances are made public, any network observer can extract payment transfer information (including the source, destination, and the value of a payment) by simply observing changes in the balance of channels. To prevent this, the Lightning Network requires nodes to conceal the balance of their channels from others.

Although balances are kept secret, several recent studies have shown that they can be discovered or at least estimated using simple probing techniques: suppose Carol is interested to know Alice's balance on a channel between Alice and Bob. A basic approach described in [2], is for Carol to first open a channel with Alice. If successful, as shown in Figure 1, Carol then sends a "fake payment" to Bob through Alice. Such a fake payment will trigger an error message. If Carol receives the error message from Alice, she concludes that Alice's balance is lower than the amount of payment because only Bob, the recipient of the payment, can determine that the payment is fake. Otherwise, if Carol receives the error message from Bob, she knows that the payment has successfully passed through the Alice-Bob channel, thus Alice's balance must be higher than the value of the payment. Carol can repeat sending fake payments with different values (following, say, a binary search pattern) to Bob, improving her estimate on Alice's balance with transmission of each such fake payment.

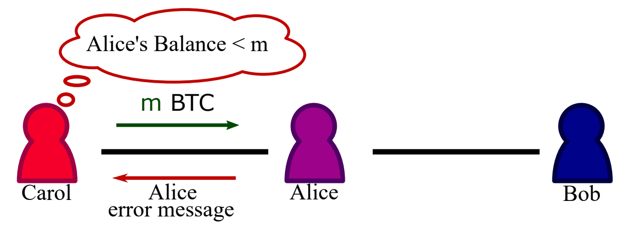
The above balance discovery method does not work if neither Alice nor Bob accepts Carol's request for opening a channel. Furthermore, the method does not scale: if Carol wants to discover balances of many channels, she has to open many channels, too. This is costly as Carol has to lock funds and pay transaction fees for opening each channel.

Recent extensions [3]–[5] have shown that Carol does not need to open a channel with Alice to discover her balance. Instead, she can use any channel, and send fake payments using a path $\mathcal{P}$ that goes through the Alice-Bob channel. If she receives an error message from Bob or any other node after Bob on path $\mathcal{P}$, she knows that the payment has passed through the Alice-Bob channel, hence Alice's balance must be higher than the payment's value. If she receives an error message from Alice, on the other hand, she can conclude that Alice's balance was not enough to let the payment go through. If Carol receives an error message from a node that is before Alice on path $\mathcal{P}$, however, she will not gain any information about Alice's balance. In other words, Carol cannot gain any information on Alice's balance if she cannot push enough flow of money to Alice through $\mathcal{P}$. Of course, Carol can try again by choosing a different path. However, there is no guarantee that the new path can carry enough flow to Alice. In fact, there

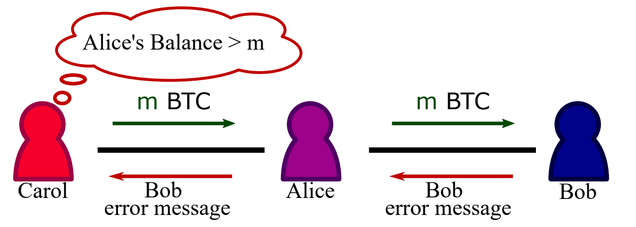
978-1-6654-9538-7/22/$31.00 ©2022 IEEE



(a) Carol sends a fake payment (with value m Bitcoin) to Bob through Alice.



(b) Carol receives an error message from Alice.



(c) Carol receives an error message from Bob.

Figure 1: A simple probing technique for discovering Alice's Balance.

may be no single path that can carry enough flow from Carol to Alice.

Existing extensions use a sequential probing process in which fake payments are sent sequentially. This slows down the balance discovery process. This is not desired because Carol may need to know balances as soon as possible since the Lightning Network is dynamic and balances can change quickly.

To address the above shortcomings, we present *Torrent*. Unlike the existing methods, Torrent can concurrently send several payments through multiple paths. These payments are temporarily held at a receiving node to prevents payment cancellations during the balance discovery process. This technique allows Carol to quickly push a large flow of money through Alice using concurrent payments. Using Torrent, Carol can discover Alice's balance if the maximum flow that she can bring to Alice is larger than Alice's balance. To achieve the maximum flow (or get close to it), we devise new algorithms that work within the unique limitations of the Lightning Network and our specific channel discovery problem. We show that, using these algorithms, Carol can quickly discover balances of many remote channels with opening at most two channels.

We remark that Carol will know Alice's balance if she discovers Bob's balance. It is because the channel capacity (which is the sum of the balances of its owners) is public information. Therefore, in all the above methods, an alternative way for Carol to discover Alice's balance is to discover Bob's balance. We also remark that Carol's intention in discovering balances may not be malign. For instance, Carol may want to discover channels with low balances so she can help her clients avoid payment routes that can fail.

II. Background

A. Payment Channels

Two parties can open a payment channel by publishing a transaction on the Bitcoin blockchain. This transaction locks their funds into a 2-of-2 multi-signature address that is controlled jointly by the two parties. When this opening transaction is confirmed, the two parties can exchange any number of private transactions off the Bitcoin blockchain without requiring any confirmation from the blockchain. Each of these so called off-chain transactions essentially changes the channel state by updating balances on the channel. If one party cheats and closes the channel with an earlier state, the other party can dispute and collect the whole channel's fund if the party reacts within a certain timeout.

B. The Lightning Network

The Lightning Network (LN) is a network of payment channels. LN allows any party to make a payment to another party provided that there is a path of payment channels between the two parties, and all the channels on the path have enough balances to transfer the payment.

To enable parties to discover paths to each other, LN nodes advertise their channels by gossiping their channel's information. This information includes the channel capacity, but dose not include the channel balances to comply with the channel concealment policy. Since channel balances are not known, a payer cannot guarantee that their payment will go through a path as one of the channels on the path may simply not have an enough balance. If this occurs, the payment will fail, and the payer will receive an error message, which notifies the channel that failed. This helps the payer to avoid the failed channel in their subsequent attempts.

C. Maximum Flow

The maximum flow problem is about finding the maximum amount of feasible flow that a node (source) can send to another node (sink) in a flow network. A flow network is a directed graph, where each edge has a capacity that restricts the maximum flow that can pass through the edge. The amount of flow that enters a node (other than the source and the sink) must be equal to the amount of flow that leaves the node.

The maximum flow problem was first formulated in 1954 by Harris and Ross for Soviet railway traffic flow [6]. Soon after, Ford and Fulkerson proposed the first algorithm to solve the problem [7]. Since then many improved solutions with lower computational complexities have been proposed. These solutions are used today in various applications including mining industry [8], optimizing spatiotemporal data scheduling methods [9], optimizing complex transactions in a traditional bank accounts, Bitcoin wallet accounts and Bitcoin exchanges [10] and controlling power transmission [11].

Despite the existing solutions, solving the maximum flow problem in the Lightning Network is challenging because balances are not known. In the absence of balance information, we can access an oracle[1]. The oracle can tell us whether a given path can carry a given flow, and if so, can be asked to make the payment and update balances of the channels on the given path. An interesting open question is at least how many oracle accesses we need to calculate the maximum flow.

III. Torrent

Torrent can be viewed as a multi-path payment approach to channel discovery; rather than probing the balance of a target channel through a series of sequential single-path payments—as done in the existing methods—Torrent uses multiple paths to concurrently transmit partial payments through the target channel. To ensure that these partial payments are not canceled during the channel discovery process, receiver(s) in Torrent temporarily hold the received payments.

Let f_{max} denote the maximum flow that Torrent can push through the target channel assuming that the target channel has an infinite balance. Torrent works based on the idea that the balance of a target channel can be discovered if f_{max} is greater than the balance of the channel. This is because, under the above condition, the balance of the target channel becomes the bottleneck for the maximum flow that can be transferred through the target channel. Under this condition, therefore, balance of the target channel is equal to the maximum amount of money that can be sent through the target channel.

In this work, we show this condition holds for a vast majority of channels in the Lightning Network.

Example 1. *Figure 2 shows a small payment networks with 11 channels and 10 nodes. Carol has two nodes and is interested to know the balance of Alice on the Alice-Bob channel. Assume that there are enough balances on Carol's channels. If we change Alice's balance from 200k to ∞, then the maximum flow that Carol's node on the left (the sender) can send to her node on the right (the receiver) is 230k Satoshis. Note that Carol can send*

$$\begin{array}{ll} 60k & \text{via path } (n_1, n_2, Alice, Bob, n_4, n_6) \\ 160k & \text{via path } (n_1, n_3, Alice, Bob, n_5, n_6) \\ 10k & \text{via path } (n_1, n_2, Alice, Bob, n_5, n_6) \end{array}$$

Also, note that cutting channels (Bob, n_4) and (n_5, n_6) will disconnect the sender and the receiver. These two channels can carry a total flow of

$$60k + 170k = 230k$$

Therefore, $f_{max} = 230k$, which is higher than Alice's balance of 200k. Consequently, in this example, Torrent is able to discover Alice's balance. We remark that the maximum flow that a single path from Carol's sender to her receiver can carry is at most 160k. Therefore, the best conclusion that the existing channel discovery methods can make in this example is that Alice's balance is at least 160k.

[1]The oracle [12] is the Lightning Network itself.

Types of Torrents. As mentioned earlier, Torrent requires the receiver(s) of payments to temporarily hold any received payment. To achieve this, Torrent can employ its own receiver(s). Clearly, in this case, receivers would follow Torrent and temporarily withhold received payments as needed. We refer to the case where Torrent uses its own receiver(s) as *Torrent Type A*.

Alternatively, in what we call *Type B*, Torrent uses Alice and/or Bob as receivers. This is possible if Alice and/or Bob support AMP [13], a multi-path payment method supported in the new versions of LND[2] [15]. Note that in AMP (as in any multi-path payment method) a receiver needs to hold partial payments until it receives the payment in full. Torrent can benefit from this property of AMP and use Alice and/or Bob as receivers instead of employing its own receiver(s).

Note that Torrent Type A not only needs to push a large flow into the target channel, but also has to push this flow out of the target channel and deliver it to its receiver(s). Torrent Type B, however, only needs to push a large flow towards the target channel. As a result, in general, Torrent Type B can push a larger flow than Type A. For instance, in the network of Figure 2, Carol can send a flow of size 110k+160k= 270k in Torrent Type B, while the maximum possible flow in Type A (as shown in Example 1) is 230k.

IV. Implementation

As before, suppose Carol is interested to know Alice's balance. To implement Torrent, Carol needs to open a set of channels, and use them to send/receive a large flow of payments through the target channel, i.e. Alice's channel. If Carol knows balances of all channels except the target channel, then she can use linear programming—as we will show later—to find an optimal set of payment paths that can transfer the maximum flow through the target channel. In practice, however, Carol does not know balances of channels. This makes the problem of finding the max flow challenging.

To solve the above unique max flow problem, we propose two algorithms. The first algorithm assumes that all channels, except the target channel, have a certain minimum balance. For example, it may assume the balance of every channel is at least 5% of its capacity. Using this assumption, the algorithm uses a linear program to solve the max flow problem. The advantage of this approach is that it can discover balances very fast because 1) it can pre-compute paths and the amount of payments that should be transferred on each path; 2) it can send all the payments in parallel as the amount of payments are known a priori. Of course, the minimum balance assumption may not hold for all channels. However, this does not negatively impact the algorithm if the feasible max flow is considerably higher than the balance of the target channel. Besides, the algorithm can always re-attempt a failed payment by trying smaller payments on the failed path.

The second algorithm is a greedy iterative algorithm. In each iteration, the algorithm updates its view of the network

[2]About 90% of nodes in the Lightning Networks are LND nodes [14].

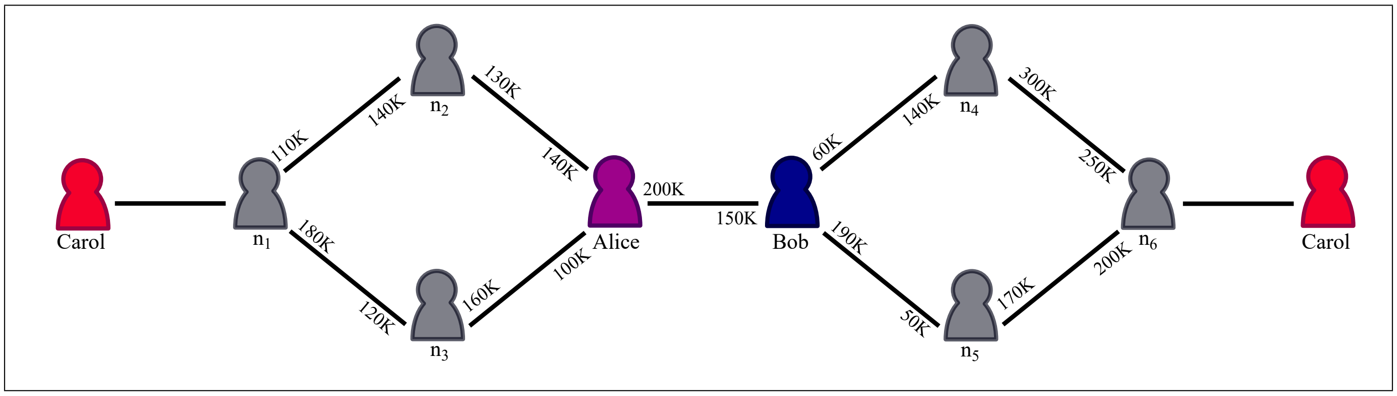


Figure 2: A small payment network. Carol has two channels with nodes n_1 and n_6, and is interested to discover balance of the channel between Alice and Bob. Balances of Alice and Bob on this channel are 200k and 150k Satoshis, respectively.

based on results of the previous iteration, then selects a set of short disjoint paths and pushes the maximum possible flow through these paths.

In the following, we explain these two algorithms for Torrent Type A. These algorithms can be easily converted and used for Type B, because Type B is essentially a special case of Type A where receivers are the owners of the target channel. We start by modeling the Lightning Network and defining our max flow problem.

A. Network Model and Problem Definition

We model the Lighting Network as a graph $G = (V, E)$, where V is the set of nodes and E is the set of channels. For any channel $(u, v) \in E$, let $b(u, v)$ denote the balance of node u, and $c(u, v)$ denote the channel capacity. Therefore, we have $c(u, v) = b(u, v) + b(v, u)$. Recall that $c(u, v)$ is public information while $b(u, v)$ is private to nodes u and v.

The Max Flow Problem in LN. Given a graph G, a capacity function $c(.)$, a source $s \in V$, a destination $d \in V$, and a target channel $(a, b) \in E$, the problem is to find the maximum flow that s can send to d through the target channel (a, b). In addition to finding the maximum flow, the problem asks for a set of paths and payments that can achieve the max flow. Note that, in this problem, the balance function $b(.)$ is unknown. However, there is an access to an “oracle”, which can be asked whether or not a given path can transfer a given payment. The oracle can also make a payment on a given path—assuming that the path can carry the payment—and accordingly update the balance of channels on the path. Note that this oracle is the Lighting Network itself.

B. Linear Programming

Given the balance function $b(.)$, the max flow problem can be solved in polynomial time using the following linear program (LP). This LP is similar to the one used to solve the traditional max flow problem with the main difference being that our LP maximizes the flow that enters the target channel while the traditional one maximizes the flow that leaves the source.

$$\begin{aligned} \max \quad & f_{ab} \\ \text{s.t.} \quad & 0 \le f_{uv} \le b(u, v), && \forall (u, v) \in E \\ & \sum_{u:(u,v)\in E} f_{uv} = \sum_{w:(v,w)\in E} f_{vw}, && \forall v \neq s, d \end{aligned} \tag{1}$$

In the above linear program, the variable f_{uv} represents the flow from u to v on channel $(u, v) \in E$. To avoid the target channel (a, b) to become the bottleneck, we set $b(a, b) = \infty$ in the above linear program.

LP Advantages. As mentioned earlier, in practice we do not have a direct access to the balance function $b(.)$ (we only have access to the oracle). Nevertheless, we can still benefit from solving the above LP by approximating the function $b(.)$. For instance, we may assume that $b(u, v) = 0.1 \times c(u, v)$ for every channel $(u, v) \neq (a, b)$. This assumption may not hold for all channels although the assumption is somewhat conservative—for each channel (u, v) either $b(u, v) \ge 0.5 \times c(u, v)$ or $b(v, u) \ge 0.5 \times c(u, v)$. Nevertheless, if by using the above assumption, we find that the max flow that can go through the target channel is considerably higher than, say, the capacity of the target channel, we can most likely discover the balance of the target channel in practice. The second advantage of this approach is that it can significantly speed up balance discovery because 1) we can use the method to pre-compute paths and payments, and 2) transmit all these payments in parallel. Note that if the max flow that Torrent can send through the target channel is higher than the balance of the channel, the balance of the channel can be safely assumed to be equal to the total amount of payments received at the receiver.

C. Greedy Algorithm

Our second algorithm is a greedy algorithm that works in iterations. In the first iteration, the algorithm finds a maximal set of disjoint paths[3] that go through the target channel. It then transmits payments on the selected paths until it saturates every single one of them. Before moving to the next iteration,

[3]These paths share the target channel as well as the source and destination channels; otherwise, they are disjoint.

the algorithm removes bottleneck channels, i.e. those from which it received an error message. For instance, if the algorithm receives an error message from node u indicating an insufficient balance on channel (u, v), then the algorithm removes the channel (u, v) in the direction of u to v. The algorithm, however, keeps the other direction (i.e. v to u) of channel (u, v) as the channel may be able to transfer payments in this direction.

After removing channels, the algorithm proceeds to the next iteration and repeats the above process. The algorithm terminates as soon as it concludes that it cannot transfer more payments though the target channel. This happens if it receives an error message indicating an insufficient balance on the target channel, or if the algorithm is unable to transmit more payments using new paths. When the algorithm terminates, it returns the amount of total payments collected at the receiver as the balance of the target channel.

Example 2. *Consider the simple payment network shown in Figure 2. If we run the greedy algorithm on this network, the algorithm can select paths*

$$p_1 = (n_1, n_2, Alice, Bob, n_4, n_6)$$

and

$$p_2 = (n_1, n_3, Alice, Bob, n_5, n_6)$$

in its first iteration. In this network, the algorithm can send up to 60k on path p_1 and up to 160k on path p_2. Since the sum of these two payments is higher than Alice's balance of 200k, at some point in this iteration the algorithm may receive an error message from Alice indicating insufficient balance. If this happens, the algorithm terminates and returns 200k as Alice's balance since this is the maximum it could transfer through her channel.

If Alice avoids sending error messages, the algorithm will move to next iterations and tries to find new paths and send new payments to increase the total received amount of 200k. Ultimately, the algorithm will terminate. Clearly, the algorithm can never deliver more than 200k to its receiver because Alice's channel is the bottleneck. Therefore, when the algorithm terminates, it will return the max amount it could deliver to its receiver (i.e. 200k) as Alice's balance.

V. Simulation Results

To evaluate the performance of Torrent, we downloaded a snapshot of the Lightning Network[4], which included 9169 nodes, and 76856 channels. We used the snapshot to generate the topology graph G and the capacity function $c(.)$. The snapshot does not include the balance information as this information is private. Therefore, we used the following two methods to assign balances to channels:

- **Fractional Balance.** In this method, balances of every channel $(u, v) \in E$, except the target channel, are set to a fixed fraction of the channel capacity. That is, for every $(u, v) \in E$ both $b(u, v)$ and $b(v, u)$ are set to $\alpha \cdot c(u, v)$, where $\alpha < 0.5$ is a fixed number.
- **Random Balance.** This method selects balance of every channel (u, v), except the source and destination channels, uniformly at random from the interval $[0, c(u, v)]$.

[4]The snapshot was downloaded in March 2021.

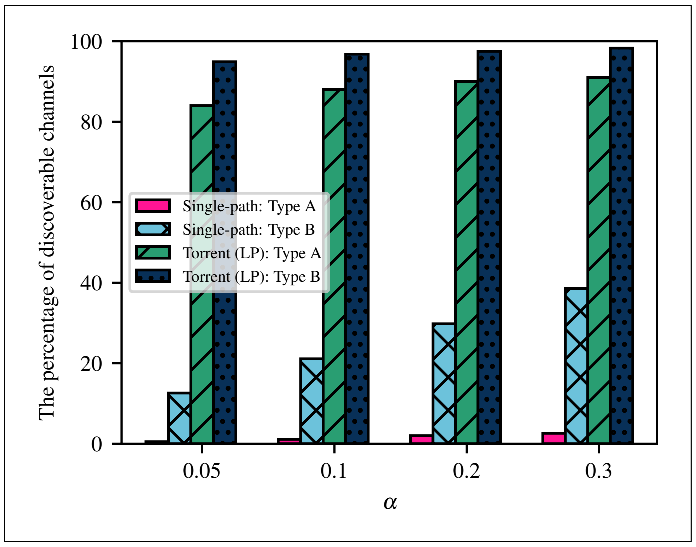


Figure 3: The percentage of discoverable channels by Torrents.

To run Torrent Type A, we open two channels with the two nodes that have the highest degrees in the network (recall that the network topology is public information).

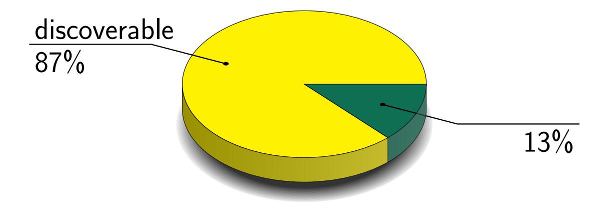
For Torrent Type B, we use only one of these two channels. In our simulations, we assume that the opened channels have enough balances.

A. Linear Programming

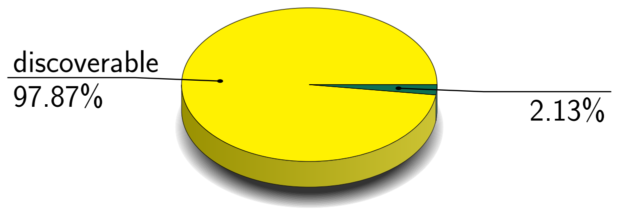
In our first set of simulations, we set the balances of all channels (except the source, target and destination channels) to a fraction $0.05 \leq \alpha \leq 0.3$ of their capacity (recall that the capacity of every channel is public information). Then, we run Torrents Type A and B by solving the max flow problem using linear programming. We marked a target channel as *discoverable* if Torrent could generate a flow that is at least equal to the capacity of the target channel. Note that this amount of flow is guaranteed to be at least equal to the balance of the target channel as balances of every channel are capped by the channel capacity.

Figure 3 shows the percentage of discoverable channels by Torrents Type A and B. As shown, Torrent (particularly Type B) can discover balances of a large percentage of channels even in this restricted setting. For instance, if we set $\alpha = 0.05$—that is we set the two balances of every channel to 5% of the channel capacity—then Torrent Type A and B can, respectively, discover balances of 84% and 94.9% of all channels in the network.

Figure 3 also shows the the percentage of discoverable channels when the best single path (i.e., the path that can carry the maximum flow to the target channel) is used. As shown, the performance of single-path based discovery methods is significantly lower than the performance of Torrent. This is



(a) Type A



(b) Type B

Figure 4: The percentage of discoverable channels by Torrents Type A and B when they use the greedy algorithm.

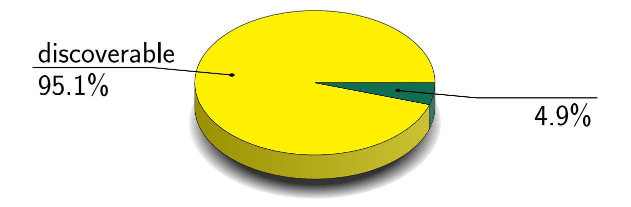
somewhat expected because the flow that any single-path can push through the target channel can be significantly lower than the flow that multiple-path payment methods can push.

B. Greedy Algorithm

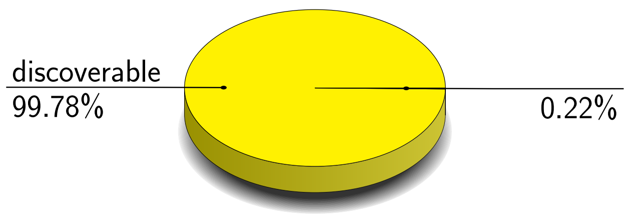
To evaluate the performance of Torrent powered by the greedy algorithm, we used the Random Balance method. Recall that in this method, balances of channels (except the source and destination channels) are selected uniformly at random. In this set of simulations, a target channel was marked as *discoverable* if Torrent could generate a flow that is at least equal to the balance of the target channel. Figure 4 shows the percentage of channels that could be discovered by Torrents Type A and B, respectively. This percentage is about 97.8% in the case of Torrent Type B. This is remarkable as, in Type B, Torrent only uses one channel, yet it can discover balances of nearly all the channels in the network.

As stated earlier, Torrent Type A is expected to discover a smaller percentage of balances that Type B. This is because Torrent Type A has to not only deliver a large flow of payments to the target channel, but also must deliver this large flow to its own receiver. To improve the performance of Torrent Type A, we can use multiple receivers (in our simulations, we used only a single receiver).

An interesting question here is how well the greedy algorithm performs compared to the optimal solution. To answer this question, we used a linear program, with the same balance function used in the greedy algorithm, to find the maximum number of discoverable channels. Recall that, given the balance function, linear programming can find the optimal solution to the max flow problem. The results of this simulation are shown in Figure 5. A parallel comparison between Figures 4 and 5 shows the greedy algorithm can discover balances of the vast majority of the channels that are possibly discoverable. This is particularly the case for Torrent Type B.



(a) Type A



(b) Type B

Figure 5: The optimal percentage of discoverable channels.

VI. Related Work

Table I compares Torrent with the existing balance discovery methods. As reflected in Table I, the methods proposed in [2], [16], [17] require opening a channel directly connected to the target channel. This may not be possible in practice as the owners of the target channel may not accept new channels. In addition, these methods are not scalable because they have to open a new channel for every target channel.

The remaining balance discovery methods [3]–[5] do not require direct connection to the target channel. However, they all use single-path payments, and sequentially probe the target channel. As mentioned earlier, single-path payments can generate a limited flow compared to multi-path payments, hence have limited capability to discover balances of remote channels. Moreover, these methods are slower than Torrent as they do not transmits payments in parallel.

The major cost of balance discovery is due to opening channels for probing purposes. The methods in [2] and [16], [17] require to open up to two payment channels per each single target channel. Other methods such as NIBT [5] open two channels irrespective of the number of target channels. This significantly reduces the cost of channel discovery. As in NIBT, our method requires at most two channels. Unlike NIBT, our methods cancels payments, and this way avoids paying transaction fees.

VII. Conclusion and Future work

We proposed Torrent, a powerful balance discovery approach. Unlike the existing balance discovery methods that use single-path payments, Torrent uses multi-path payments. This allows a single node to push a large flow of payments through any target channel. Torrent can work even when the Lightning Network does not transmit error messages to indicate insufficient balances. It is because Torrent can estimate a balance by calculating the maximum amount of fund that it can deliver to its receiver(s) through the target channel. For many channels in the Lightning Network, this maximum is equal to the balance

Method	**Need a direct channel?**	**Relies on control messages?**	**Success rate**
Balance discovery attack [2]	Yes	No	89.1%
Improved balance discovery attack [16]	Yes	No	98.37%
Generic attack [17]	Yes	Yes	No Data
Probing channel balances [3]	No.	No	65%
Probing attack [4]	No	No	No Data
Probing of Parallel Channels [18]	No	No	80%
NIBT [5]	No.	Yes	92%
Torrent	No.	No	87%–97%

Table I: Existing balance discovery methods.

of the target channel thanks to the large flow that Torrent can generate. Another advantage of Torrent is that, unlike the existing methods, it transmits payments in parallel, which can significantly speed up the balance discovery process.

A future research work is to use the technique presented in this paper to learn about the balances all over the network; that is to take a snapshot of the network's balances. Another interesting future work is to find an efficient algorithm to solve the max flow problem in the Lightning Network. The algorithm can receive the network topology and channel capacities (but not balances) as input. It also has access to an oracle that answers to questions on whether or not a path p can transfer an amount m, and if so, can transfer the payment by updating balances of the channels on path p. Unlike the existing maximum flow solution, the efficiency of the algorithm will be determined by the number of times it accesses the oracle (rather than the algorithm's time complexity).

References

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

[2] J. Herrera-Joancomartí, G. Navarro-Arribas, A. Ranchal-Pedrosa, C. Pérez-Solà, and J. García-Alfaro, "On the difficulty of hiding the balance of lightning network channels," in *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security, AsiaCCS*. ACM, 2019, pp. 602–612. [Online]. Available: https://doi.org/10.1145/3321705.3329812

[3] S. Tikhomirov, R. Pickhardt, A. Biryukov, and M. Nowostawski, "Probing channel balances in the lightning network," *CoRR*, vol. abs/2004.00333, 2020. [Online]. Available: https://arxiv.org/abs/2004.00333

[4] U. Nisslmueller, K. Foerster, S. Schmid, and C. Decker, "Toward active and passive confidentiality attacks on cryptocurrency off-chain networks," in *Proceedings of the 6th International Conference on Information Systems Security and Privacy, ICISSP 2020, Valletta, Malta, February 25-27, 2020*, S. Furnell, P. Mori, E. R. Weippl, and O. Camp, Eds. SCITEPRESS, 2020, pp. 7–14. [Online]. Available: https://doi.org/10.5220/0009429200070014

[5] Y. Qiao, K. Wu, and M. Khabbazian, "Non-intrusive and high-efficient balance tomography in the lightning network," in *ASIA CCS '21: ACM Asia Conference on Computer and Communications Security, Virtual Event, Hong Kong, June 7-11, 2021*, J. Cao, M. H. Au, Z. Lin, and M. Yung, Eds. ACM, 2021, pp. 832–843. [Online]. Available: https://doi.org/10.1145/3433210.3453089

[6] A. Schrijver, "On the history of the transportation and maximum flow problems," *Mathematical programming*, vol. 91, no. 3, pp. 437–445, 2002.

[7] L. R. Ford and D. R. Fulkerson, "Maximal flow through a network," *Canadian journal of Mathematics*, vol. 8, pp. 399–404, 1956.

[8] A. Paithankar and S. Chatterjee, "Open pit mine production schedule optimization using a hybrid of maximum-flow and genetic algorithms," *Applied Soft Computing*, vol. 81, 2019. [Online]. Available: https://doi.org/10.1016/j.asoc.2019.105507

[9] Q. Zhu, M. Chen, B. Feng, Y. Zhou, M. Li, Z. Xu, Y. Ding, M. Liu, W. Wang, and X. Xie, "Optimized spatiotemporal data scheduling based on maximum flow for multilevel visualization tasks," *ISPRS International Journal of Geo-Information*, vol. 9, no. 9, p. 518, 2020. [Online]. Available: https://doi.org/10.3390/ijgi9090518

[10] J. I. Orlicki, "Generalized minimum cost flow and arbitrage in bitcoin debit and custodian networks," in *I Simposio Argentino de Informática Industrial e Investigación Operativa (SIIIO 2018)-JAIIO 47 (CABA, 2018)*, 2018.

[11] A. Armbruster, M. R. Gosnell, B. M. McMillin, and M. L. Crow, "Power transmission control using distributed max flow," in *29th Annual International Computer Software and Applications Conference, COMPSAC*, 2005, pp. 256–263. [Online]. Available: https://doi.org/10.1109/COMPSAC.2005.121

[12] K. Weare, "MIT Researchers Test Oracles and Smart Contracts on Bitcoin Lightning Network," https://www.infoq.com/news/2018/06/Bitcoin-Lightning-Oracles/, 2018.

[13] O. Osuntokun, "AMP: Atomic Multi-Path Payments over Lightning," https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-February/000993.html, 2018.

[14] A. Mizrahi and A. Zohar, "Congestion attacks in payment channel networks," *CoRR*, vol. abs/2002.06564, 2020. [Online]. Available: https://arxiv.org/abs/2002.06564

[15] Lightning-Network-Daemon, "lnd v0.13.0-beta," https://github.com/lightningnetwork/lnd/releases/tag/v0.13.0-beta, 2021.

[16] G. van Dam, R. A. Kadir, P. N. E. Nohuddin, and H. B. Zaman, "Improvements of the balance discovery attack on lightning network payment channels," in *ICT Systems Security and Privacy Protection - 35th IFIP TC 11 International Conference, SEC*, ser. IFIP Advances in Information and Communication Technology, vol. 580. Springer, 2020, pp. 313–323. [Online]. Available: https://doi.org/10.1007/978-3-030-58201-2_21

[17] H. Yousaf, G. Kappos, A. Piotrowska, S. Kanjalkar, S. Delgado-Segura, A. Miller, and S. Meiklejohn, "An empirical analysis of privacy in the lightning network." Financial Cryptography and Data Security, 2021.

[18] A. Biryukov, G. Naumenko, and S. Tikhomirov, "Analysis and probing of parallel channels in the lightning network," *IACR Cryptol. ePrint Arch.*, vol. 2021, p. 384, 2021. [Online]. Available: https://eprint.iacr.org/2021/384