Routing payments on the Lightning Network

Giovanni Di Stasi, Stefano Avallone, Roberto Canonico, Giorgio Ventre
University Federico II of Napoli (Italy)
Email: {name.surname}@unina.it

***Abstract*—Bitcoin is a new digital currency created with the aim of being decentralized, peer-to-peer, censorship-resistant, borderless, scarce, fast and cheap to use. Since Bitcoin main way of operating is based on broadcast communications, e.g. all transactions of the currency reach all network peers, how to achieve scalability is one of the most important concerns for it to achieve wide usage. One of the proposals to improve its scalability takes the form of the Lightning Network (LN) which consists of an overlay network on top of the base Bitcoin layer. Such an overlay allows to perform a new kind of off-blockchain transactions, i.e. transactions not broadcast to the entire network, that, differently from previous off-blockchain solutions, do not require to put trust in any third entity.**

Important open problems of the LN are i) defining how payments should be routed and ii) establishing which fee policies intermediate nodes should apply for forwarding payments. The contributions of this work are twofold: we first propose a new, more general, way for nodes to apply fees for forwarding payments which allows to keep the network balanced and improve its performance in the long term. Secondly, we propose a new multipath routing payment scheme, based on the atomic multipath payment method, which is able to significantly reduce the fees paid by users, while being fast and also able to keep the network balanced.

I. Introduction

Bitcoin is a new digital currency which is considered to be decentralized, peer-to-peer, censorship-resistant, borderless and scarce. The digital currency also aims at being fast and cheap, but to fully reach such goals important scalability issues must be addressed first. Bitcoin was invented by an unknown programmer, or a group of programmers, under the name Satoshi Nakamoto and released as open-source software in 2009 [8]. While Bitcoin with the capital B generally denotes the overall system (composed by the peer-to-peer network and the underlying protocol), *bitcoins* indicates the currency units that are created and transferred on the Bitcoin network itself.

During the course of years, several different currencies derived from Bitcoin have been devised and are now being referred to as cryptocurrencies. Such currencies differ on some parameters, e.g. the rate and amount of currency units issued, or offer some additional features, e.g. privacy for transactions. The ideas discussed in this work apply as well on most of the released cryptocurrencies so far.

The Bitcoin architecture is based on a peer-to-peer network that distributedly manages a public ledger called the *blockchain*. The blockchain is where transactions are recorded and new currency units are issued.

A problem of Bitcoin and, in general, of all cryptocurrencies that are based on the blockchain, is scalability. Indeed, only a limited number of transactions per second is achievable. The limitation stems from the broadcast nature of communications of the protocol, since every transaction has to reach every node of the network. For such a reason, the higher the number of transactions per second, the higher the bandwidth requirement for nodes and, therefore, the more difficult it becomes to be part of the network, as a *full node*, i.e. a node that downloads and verifies all transactions. Other ways to participate are possible, e.g. light client, but to run a full node is the only way to being able to fully verify all the currency history and get the greatest amount of security against attacks.

The maximum rate at which the Bitcoin blockchain can process transactions, given the current network rules, is only a few transactions per second, i.e. 3-7 (a precise number cannot be stated since the transactions' sizes vary). Such an amount of transactions per second is orders of magnitude smaller of that achievable by centralized payment processors such as VISA or Paypal.

To increase such a figure, several proposals have been made. The first and simplest is to simply remove the limit on the blockchain growth. Bitcoin developers [1] and other important Bitcoin stakeholders have, however, refused to perform such a change in the protocol, for the reasons previously anticipated. Given such a refusal to change the limit by the developers, other currencies have been created with the precise aim of creating a Bitcoin clone where the size of the blockchain could grow at a higher rate, e.g. Bitcoin cash.

Another proposal, which follows a different approach, is to deploy the Lightning Network (LN). The LN allows to significantly increase the throughput of Bitcoin in term of transactions, while keeping the bandwidth requirements low and therefore the network more decentralized. The LN is an overlay network on top of the Bitcoin network that allows to perform a new form of off-chain [2] transactions that, being off-chain, do not need to be broadcast on the network and recorded in the blockchain. Differently from previous forms of off-chain transactions, LN transactions do not require to put trust in any third entity, so intermediate forwarding nodes cannot steal funds. Indeed, payments on the LN follow a unicast path from the sender to the destination traversing a series of LN nodes which forward them.

The LN can be deployed on top of the Bitcoin network, as it is today, without requiring any modifications to the base Bitcoin layer. With respect to the LN, the blockchain acts as a fund allocator and as an arbitrator, in case of dishonest

[1] Bitcoin developers are the ones that have commit access to the bitcoin github repository.

[2] Off-chain and off-blockchain are synonyms.

behavior or malfunctions of LN nodes. Additional advantages of off-chain transactions, and therefore of the LN, as opposed to merely increasing the blockchain growth rate, are their almost immediate confirmation and very low fee, because they are not included in the blockchain which takes several minutes. While the LN approach seems promising, there still are, though, important issues to be addressed before it can be deployed and exploited to the the full extent. The first one is a routing problem, i.e. defining how payments should be routed. The routing problem is in turn dependent on how the LN nodes apply the fees for processing payments. Currently, as defined by the BOLT specifications [1], intermediate nodes apply a fixed charge plus a proportional fee for forwarding a payment. As a first contribution of this work, we show that such an approach is not optimal in terms of guaranteeing the best network performance and propose a new, more general way of specifying fees. Such a new way allows to keep network links more *balanced* and increase the network performance in the long term. To specify what a balanced link is we must first say that each link in the LN has funds assigned to its endpoints whose amount at each endpoint can be changed to perform payments. A balanced link is defined as a link where the overall funds allocated to it are equally distributed to its endpoints. Indeed, when a link is balanced, it can perform payments in both directions with equal probability and, potentially, can stay open for longer times (without requiring the opening of a new channel); moreover, in case that no information on the allocation of funds to the endpoints of channels is disseminated in the network, it is safer to assume for each channel that the funds are equally split between its endpoints. As a second contribution, we propose a new multipath payment routing scheme, based on the Atomic Multipath Payments feature [2]. Such multipath payment routing scheme allows to significantly decrease the network fees for users, while still keeping the network balanced.

II. Bitcoin basics

As previously said, the Bitcoin technology is based on a peer-to-peer network that allows users to exchange digital tokens called bitcoins without the intervention of any financial institution. Such tokens are exchanged through the broadcasting of signed transactions which are distributedly verified and recorded by the network in a public ledger called the *blockchain*. The history of all transactions ever happened in the network is stored on such ledger, in a transparent way.

The blockchain, as the name suggests, is composed by a set of data structures called *blocks* organized in a chain structure. The first block of the chain is called *genesis block* and is hard-coded in the Bitcoin software. Each of the subsequent blocks is linked to the previous one (the parent) by means of an *hash pointer*, an integer value in the block header which corresponds to the SHA-256 hash of the header of the previous block. The blockchain structure assures that no block can be deleted or modified in the middle of the chain without requiring all the subsequent blocks to be modified as well. Indeed, if a block were to be modified, the block pointing at it would be invalid because of the invalid hash pointer and, for the same reason, all the following blocks would be invalid as well.

The process by which the blocks are created is called *mining*. The name derives from the fact that such process mimics the process of mining gold, as it is both cost-intensive and allows the issuance of new currency (gold in a case and bitcoins in the other). Indeed, each block of the blockchain contains, by convention, a special transaction called *coinbase* which assigns previously non existent bitcoins to the node that created such block. The mining process will be described in greater details in sec. II-B.

The blockchain is constructed according to a set of rules referred to as *consensus rules*. Should a node try to issue a block or a transaction not in conformity to such consensus rules, the block or the transaction would be rejected by the rest of the network. As long as the majority of nodes act honestly, the mining process assures that the blockchain is formed according to the consensus rules, as will be explained in sec. II-B.

A. Block and transaction formats

Each block contains a set of transactions that has been broadcast in the network by nodes. It is composed by an header and a payload. The header contains the following fields:

- *version* version of the block format
- *prev_block* the hash pointer to the previous block, i.e. SHA256 hash of the previous block header
- *merkle root* The merkle root of the block, i.e. a SHA256 hash representing all the transactions included in this block
- *timestamp* Timestamp of when the block was created
- *nonce* The nonce value used to generate this block (used in the mining process)
- *txn_coint* Number of transactions in this block set always to zero

The payload of the block, instead, includes a set of (well-formed) transactions. Transactions, with the exception of the coinbase transaction, have the following format:

- *Version no* - the version of the transaction format
- *In-counter* - number of inputs of the transaction
- *list-of-inputs* - the inputs of the transaction
- *Number of output* - the number of *outputs* of the transaction
- *list-of-outputs* - the outputs of the transaction
- *lock_time* - block height or timestamp after which the transaction is final

An *output* is a (logical) container of bitcoins and is a structure consisting of two fields: i) *value*, the number of contained *satoshis*, which is the smallest unit of the currency and is $1/10^9$ of a bitcoin; ii) *scriptPubKey*, a script in the Bitcoin scripting language which defines how *value* can be spent, i.e. moved to a different output.

An *input* is a reference to an output. It consists of three fields: *Previous tx*, the transaction id of the transaction containing the referred output; *index*: the index of the output in the

referred transaction; *scriptSig*: a script in the Bitcoin scripting language which is used, in conjunction with the scriptPubKey of the referred output, to authorize the spending of the output. In particular, the transaction is valid if the script obtained by the concatenation of the scriptPubKey and the scriptSig evalutes without errors.

As previously anticipated, the coinbase transaction has a different format. In particular, such transaction, used to create new currency tokens, includes a single fictitious input and has one output which contain a number of satoshis as defined by the protocol (more details in the following section).

B. Mining process

As previously stated, the mining process has the purpose of constructing the blockchain according to the consensus rules. The difficulty of the process has significantly increased over the course of years and the mining process nowadays can be performed efficiently only by means of highly specialized ASIC (Application Specific Integrated Circuits) equipment.

The mining process works as follows. Any time a miner receives a new transaction, it verifies and stores it in a local memory area called *mempool*. The transaction is verified in the following way. Apart from the obvious checks, e.g. In-counter actually equal to the number of inputs in list-of-inputs, each input is checked by constructing a script composed of the scriptSig of the input and the scriptPubKey of the referenced output. Such resulting script must evaluate to true for the transaction to be valid. As a basic example, such script could verify that the output associated to particular public key is spent by the user that demonstrates the possession of the corresponding private key.

While the miner collects and verifies new transactions, it also tries to forge a new block by picking a set of transactions from the mempool that maximize its revenues. A component of the revenues is the transactions fees. Because of the limited block size, transactions that pay the higher fee per unit of size are chosen. Apart from the transaction fees, another component of the revenues is the block prize that is given to the block creator. Indeed, each new block contains, by consensus rules, a coinbase transaction that assigns to the miner newly created bitcoins. The amount of bitcoins to be assigned was 50 at the genesis block and has been halving every 210.000 blocks (around 4 years). The halving is meant to continue at the same rate until year 2140, when no new bitcoins are meant to be created. At that point, 21 million of bitcoins would exist.

To forge a valid block, the miner has to find a value for the block nonce field such that the SHA-256 hash of the block header is below a certain threshold. The threshold is computed such that, in average, a new block is found by a miner every 10 minutes. The only known way to find the nonce is to perform a brute-force search by trying different nonce values until one that meets the condition is found. As the nonce field has become too little for the purpose, i.e. all nonce values were tried and no solution could be found, miners also started to change the timestamp of the block, to try to come up with a different SHA-256 block hash header. The process of brute-force searching entails the consumption of energy for which the miner is compensated by the fees and the new created bitcoins.

When a miner is able to forge a new block, it has to broadcasts it in the network in order for all the other nodes to verify and store it in their local blockchains. All nodes participate in the broadcasting of new blocks, not only miners. While miners validate and perform the mining process to extend the blockchain, full nodes just verify and forward valid blocks, while they reject them if the consensus rules are not met.

It may happen that more than a miner forge a new block at approximately the same time. In such a case, all of such miners would send their blocks to have them accepted by the network. Other nodes and miners, though, would consider only the first new block they receive. At that point, different miners will start working on extending a different blockchain. Eventually, though, one of competing blockchains will become longer than the other, in terms of computing power required to construct it, and, as for the consensus rules, will become the only valid blockchain. Indeed, honest nodes working on the shorter blockchains will have to discard such blockchains and accept the longer one, on which, miners will work to extend.

C. Scalability issues of the blockchain

The Bitcoin network at the moment is able to process only a few transactions per second, because of the limit imposed on the rate of blockchain growth. Indeed, each new block added to the blockchain cannot have a *weight* greater than 4 million, as for the consensus rules [3]. Such a limit has been introduced to ensure the decentralization of the network, defined in terms of how easy it is for nodes with low bandwidth and computational capacities to join the network and autonomously verify all the transactions, i.e. join as a full-node.

As current transactional capability is not enough for the Bitcoin currency to be adopted at large scale, proposals have been made to overcome such a limit. One the proposals consists in changing or eliminating altogether the limit on block size. Such a proposal, though, is not encountering a broad support from the Bitcoin stakeholders, such as the main Bitcoin developers, because of fear of losing the decentralization property of the network. To make such change even more difficult, there is the fact that it would require a so-called *hard fork* of the protocol, i.e. a fundamental change of the Bitcoin protocol that is not backward compatible and requires all the nodes to upgrade. Hard-forks are considered very risk because if not all the nodes upgrade, the network splits in two parts and two different currencies are created.

Another proposal consists in constructing another layer on top of the Bitcoin protocol, following the general architectural approach adopted in computer networks designs.

The Lightning network, described in the following section, follows such an approach.

III. Lightning Network

The Lightning Network [9] is a layer-2 protocol constructed on top of the current Bitcoin protocol. Such layer-2 protocol allows the deployment of an overlay network where off-chain payments can be made from node to node on a path of such an overlay, without the need to put any trust on any of the nodes of the path itself.

Such a protocol makes use of *payment channels*. A payment channel is a logical connection between two Bitcoin peers where multiple off-chain bitcoin transactions can be made. Such transactions involves only such two entities and can happen as fast as the network between them allows.

As far as the blockchain is concerned, only two transactions need to be registered on it, i.e. the one that opens the channel and another one that closes it. The transaction that opens the channel is called *Funding Transaction* and causes the locking of an amount of satoshis in it. Each of the two peers contributes to part of such amount and such part constitutes the initial balance of the peer that committed it.

Before actually broadcasting the Funding Transaction for inclusion in the blockchain, each peer signs a *Revokable Committment Transactions* (RCT) which allows the other peer to unilaterally close the channel and get the amount it committed back (to avoid the perennial lock of the funds in the channel should the other peer become unresponsive).

When the Funding Transaction is actually broadcast and confirmed by the network, the payment channel becomes usable. At that point, the peers can modify the initial balances of the channel by creating new Revocable Commitment Transactions which can be used to close the channel with a different allocations of funds than the initial for the peers.

The creation of new RCTs, though, is not sufficient, as the peers could still broadcast one of the old RCTs that close the channel assigning the old balances. For that reason, the old RCTs have to be *revoked*. Without going into all the details, the old RCTs are revoked by exchanging temporary private keys (or hash images, depending on the implementation) that allow to have one of the peer immediately get all the bitcoins of the channel should the other peer broadcast an old (revoked) RCT.

To recap, peers can create as many RCTs they want, thus changing the balances of the channels and performing payments. Many new Revocable Commitment Transactions can be made, as long as the channel remains open.

When the peers decide to close the channel, they have to agree on a final (non revocable) Commitment Transaction that allocates the final balances of the channel to the two peers. When the channel is closed in this way, both peers can immediately spend the received bitcoins.

It is worth noting that all the transactions described in this section are normal Bitcoin transaction, i.e. they are valid and can be broadcast in the network in any moment. The difference is that the peers do not need to do so, apart from the Funding and the Commitment transactions; many Revocable Commitment Transactions can be created, which represent enforceable obligations among peers, but do not need to be broadcast, if both peers are cooperative and honest.

As an example, suppose that a payment channel is created where peers commit $0.5BTC$ each. Such committed amounts are locked into the channel (so they cannot be spent elsewhere) and represent the initial balances of the peers. Suppose, then, that the first peer needed to transfer $0.1BTC$ to the second. That can be done by creating RCTs that basically change the balances in the channel by granting to the second node $0.6BTC$ and the first one that made the payment $0.4BTC$. As previously said, the Bitcoin network as a whole is not aware of such change, but the RCTs represent an agreement between the peers that can be enforced on the blockchain if needed.

Suppose now that the second node wanted to transfer $0.05BTC$ to the first node. That could be done by the two nodes agreeing on making new RCTs that assign as balances $0.45BTC$ to the first node and 0.055 to the second.

As the bitcoins locked in the channel can be in fact moved in both directions, such channels are sometimes called *bi-directional payment channels*.

The advantage of the payment channel is that many transfers can be made between the two participants, without any cost for the network as a whole. The disadvantage is that the channel needs to be created and then closed (which requires two Bitcoin transactions), which makes the use of the payment channel feasible only when the involved entities plan to perform many transactions among themselves. [7]

A. A network of payment channels

Payment channels can be used to perform many transfers of bitcoins between two peers that have a channel in place between them.

Suppose now that a node, denoted as A, wanted to transfer $0.1BTC$ to another node, denoted as C, but no channel existed between them.

Suppose, moreover, that both nodes had a payment channel open with node B. What they could do in that scenario is the following: node A could transfer $0.1BTC$ to node B through the payment channel it has with it, and, then, node B could transfer $0.1BTC$ to node C (supposing, of course, that the balance of A and B is enough, respectively, in the channels to B and to C).

Things in theory work well if all nodes are honest and cooperative, but in general that cannot be assumed to always be the case. In the previous example, for instance, node B could receive the value but not deliver it to C, therefore stealing it.

The Lightning Network protocol, as previously said, allows not to have to put any trust on the intermediate nodes. Such an important result has been achieved thanks to the definition of a new type of transaction called *Hashed Timelock Contract* (HTLC). An HTLC transaction is considered final only if the recipient is able to demonstrate the knowledge of a password in a certain timeframe, after which the transaction is canceled. A multi-hop payment is made on the LN by performing HTLCs transaction on the links of the path, all of them subject

to the knowing of the same password set by the sender, from the first to the last. Every intermediate node waits to receive an HTLC payment from the previous hop, before doing an HTLC payment to the next hop.

Once the final recipient is reached, the sender reveals the password to it. The final recipient, at that point, to finalize the HTLC is obliged to disclose the password to the peer on the last link, which, in turn, can use the password to make the HTLC it received on the penultimate link final. In a similar way, the HTLC can be made final on the last but two link, and so on until the HTLC on the first link is made final.

To clarify things, let us make an example, in the same scenario of the previous example. Node A transfers $0.1BTC$ to node B in a HTLC, which, as previously said, require the knowledge of the password to actually finalize the transfer. Node B, then, can transfer $0.1BTC$ to node C in a HTLC as well, by subordinating the finalization of the transfer to the knowledge of the same password. Node B can do that as the HTLC it received contains all the information necessary to create a new HTLC subordinated to the same password. When C receives the HTLC it can finalize it as it knows the password (it is the one that created it in the first place and communicated it privately to the sender A). For doing so, C is obliged to disclose the password, so also B gets to know it and therefore can finalize the transfer of $0.1BTC$ from A.

The aforementioned example supposes that the intermediate nodes transfer exactly the same amount of bitcoins received to the next node. In the general case, they will keep for themselves a part as a fee for the gateway service they provide. The original sender would have then to make a payment that also include all the fees to be paid to intermediate nodes.

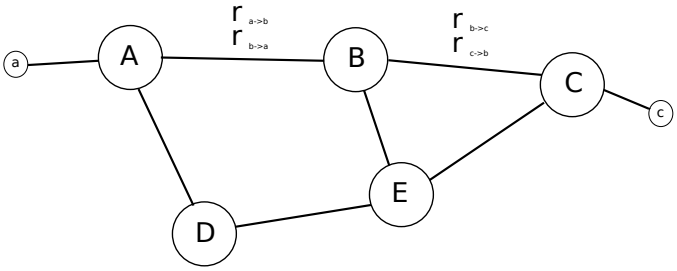

Fig. 1. Network of payment channels

To make a more complete example, let us describe how a payment of $0.1BTC$ from a to c could be made in the network of Fig. 1. Such nodes are not directly connected through a payment channel, but a path traversing gateways A, B and C can be used (a can pay to A,A to B, and so on). Let us suppose that all the nodes require a fixed fee of $0.001BTC$ to relay the payment. In such a scenario, a, that we suppose aware of both the path and the fee costs required by the nodes, sends $0.103BTC$ to gateway A. Gateway A, after having received the payment, sends $1.002BTC$ to B, equal to the received amount less the fee. B in turn sends $0.101BTC$ to C, which finally, sendls $1BTC$ to the destination c, fulfilling the payment.

IV. Problem formulation

We consider a network G made of nodes connected through Poon-Dryja payment channels (just channels in the following) [9]. We denote as N the set of LN nodes, the channel between nodes $u \in N$ and $v \in N$ as (u, v) (or simply as u, v when it is clear from the context) and the set of all channels as E.

We suppose all the channels to have been already created and to have funds (bitcoins) allocated to their endpoints. We denote the funds (balance) of node u in the channel u, v as $r_{u,v}$ and as $r_{v,u}$ the funds of y on the same channel.

Such amounts vary, of course, as previously stated, when payments are made. In particular, when a payment is made from u to v, the funds of the first decrease while the funds of the second increase (of the same amount). A payment of value P can be made from u to v only if $r_{u,v} \geq P$ and from v to u only if $r_{v,u} \geq P$.

We define as the imbalance of the channel $u \leftrightarrow v$ the value $|r_{u,v} - r_{v,u}|$. Low values of the imbalance imply that payments can be made in general in both directions, while, at the opposite, a high imbalance implies that payments can be generally made only in one direction.

We recall that payments can be made from a payer to a payee that are not directly connected through a channel. In the following we denote the intermediate nodes as gateways, to distinguish them from the payer and the payee.

In such a case, the payment flows through a series of nodes, much like an IP packet, if a feasible path with enough funds on each link is present; each of such nodes sends to the next gateway the amount of bitcoins received, apart from a small part kept as a fee. For such a reason, the initial payment made by the payer must be greater than the amount meant to be received by the payee, since it has to take into account all the fees to be paid during the trip. When a payment is performed on a payment channel, both the sender and the receiver can request a fee on the payment.

We denote as $f^s_{u,v}(P)$ the fee charged by gateway u for a payment of amount P on the payment channel between u and v and as $f^r_{u,v}(P)$ the fee charged by gateway v for the same payment. The first can be seen as a sending fee, while the second as a receiving fee. The presence of a receiving fee can be motivated by the necessity to declare it as negative so as to foster the reception of payments on that channel and reduce its imbalance.

We denote as $f_{u,v}(P)$ the sum of the sending and receiving fees. In order to reduce the amount of messages exchanged on the network, only the total fee can be advertised by node u, after it requested the information on the receiving fee to its peer.

Suppose, in the network of the figure, that a payment of value P reaches c through the path $a - A - B - C - c$. The following four fees (f_0, f_1, f_2 and f_3) are, therefore, due on the four traversed channels: $f_0 = f_{a \to A}(S)$, $f_1 = f_{A,B}(S - f_0)$ and $f_2 = f_{B,C}(S - f_1 - f_0)$ and $f_3 = f_{C,c}(S - f_1 - f_2 - f_0)$ where S is the initial payment made by a that has to be equal to $P + f_1 + f_2 + f_3$.

While in this example only one path is used to deliver the payment, in general more paths can be used by splitting the amount on them, e.g. also the A, D, E and C path could be used. Such a strategy can be convenient in the case there is no single path that alone can transfer the entire amount or when, by splitting the payment on more paths, it is possible to reduce the total amount of fees paid.

V. Fee functions

Currently, according to the BOLT [1], LN nodes can announce the fees they apply in terms of a base (always applied and independent on the payment size) plus a proportional part (to the amount to be transferred). As previously anticipated, we consider such a choice not flexible and general enough to reach certain goals such as the keeping of network links balanced (as the tests will show). Therefore, we propose a new way of specifying the fees which consists on making them continuous piecewise linear functions of the amount to be transferred. As an example, consider the function of Fig. 2.

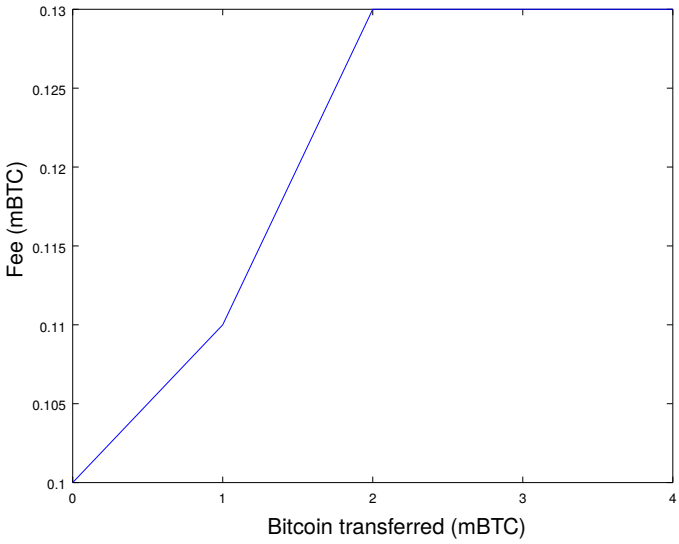

Fig. 2. Example of fee function

The applied fee, according to the figure, consists of $0.1mBTC$, plus an amount that depends on the payment to be made and increases at different slopes (0.01 and 0.02) and until a maximum fee of $0.13mBTC$.

Such fee function can be easily announced on the network by sending only a (relative) small number of values. Such values are the base fee b, a series of optional couples $m(i), p(i)$, where the first specifies a slope and the second up to which payment value that slope holds; and, finally, the last slope m.

Such a way of specifying fees allows, as particular cases, to specify the following fee policies:
i) fixed fee: by declaring as b the wanted fixed fee, no couples, and m as zero;
ii) proportional fee: by declaring $b = 0$, no couples, and m as the wanted proportional slope;
iii) proportional fee plus a base fee: as for ii), by specifying a b different from zero;
iv) proportional with cap: by declaring $b = 0$, $m(0)$ as the proportional fee, $p(0)$ the value of the payment that corresponds to the maximum fee and m as zero.

In the following section, we show how a particular fee function, among the ones possible with this scheme, is able to reach certain goals, e.g. the balancing of channels.

A. A fee function that fosters the balancing of channels

Currently, according to BOLT specifications, the path (or paths) taken by each payment is (are) decided by the node that initiate it, i.e. the payer. It is reasonable to assume that such a choice is made with the aim of minimizing, as much as possible, the cumulative amount of fees paid.

LN nodes can influence such a decision by means of the fees they announce (a payment channel that requires a lower fee is going to be selected more than a payment channel that requires a higher fee).

An important network wide objective for the LN could be the maximization of the acceptance ratio of payments. On that regard, we suppose that payments arrive dynamically and, depending on the state of the LN, can be either successfully performed or fail. In general, we can assume the probability to find one or more paths from a payer to the payee to be higher the more the links of the LN network are balanced. Indeed, when links are balance is more easy to find a path that has enough funds to reach the intended destination.

Therefore, in order to foster the balancing of channels, we propose the use of a particular fee function, which we denote as *OptimizedFees* and described as follows. Such fee function consists in applying a fixed charge, as in the normal case, to account for the communication cost on the channel, plus a variable part which depends on the size of the payment to be made. Such variable part has two slopes, a *low* slope and a *high*, i.e. steeper, slope. The low slope is applied for the part of the payment that decreases the imbalance of the channel, i.e. half of the amount of the payment channel imbalance. The high slope for the remaining part of the payment.

In details, if we denote as T a payment to be made on the channel between a and b the fee $f_{a,b}(T)$ is calculated as in Alg. 1.

$Imb = |r_{a,b} - r_{b,a}|$;
if $r_{a,b} > r_{b,a}$ **then**
  if $T > Imb/2$ **then**
    $f_{a,b}(T) = b + (Imb/2) * s_{low} + (T - Imb/2) * s_{high}$;
  else
    $f_{a,b}(T) = b + T * s_{low}$;
  end
else
  $f_{a,b}(T) = b + T * s_{high}$;
end

Algorithm 1: OptimizedFees: algorithm for fee calculation that fosters channel balancing.

where b is the base fee, s_{low} and s_{high} are two coefficients with $s_{high} > s_{low}$.

The rationale behind this fee function is to make it convenient to perform the payment (or just a part of it, in a multipath approach) on channels while at the same time balancing them, while making it less convenient to perform the payment on imbalanced channels while at the same time increasing such an imbalance.

Such a fee function could result by an agreement between the sending and the receiving nodes of the channel with the aim of allowing the incorporation of a negative fee by the receiver which is not seen by the whole network. Indeed, in such a case, the low slope is just the difference between the high slope, i.e. the fee applied by the sending node, and the aforementioned negative fee applied by the receiving node, up to a certain payment value. The result is a fee function similar to OptimizedFees which can be exactly OptimizedFees if the negative fee is applied up to the payment value that would perfectly balance the channel.

As seen in the simulation section, OptimizedFees allows to significantly increase the network balance in the long term.

VI. Routing of payments on the LN

Payments on the LN are performed, as previously stated, on a path (or paths) chosen by the payer with the primary objective of minimizing the paid fees and a secondary objective to keep the path (or paths) as short as possible[3].

In this section we discuss different algorithms for calculating such paths. The approaches we discuss are both singlepath and multipath and both exact and approximate.

As far as singlepath approaches are concerned, an exact algorithm exists that finds the path that globally minimizes the fees paid. Such an algorithm, a variation of the Dijkstra algorithm, is described in [4]. It can be applied when the fees are specified as a base plus a proportional part, as of now, but also in the case where the fee function has the general form described in the previous section, i.e. continuous piecewise linear.

The smallest amount of fees can be obtained with multipath approaches, though. Indeed, multipath approaches have the ability to split the payment in smaller payments and perform each of such smaller payments on a path that has the capacity and requires the lowest amount of fees. To the best of our knowledge, multipath algorithms for the LN have not been studied or proposed yet.

Therefore, we set as an objective to study the multipath case. To that purpose, we first define an optimization problem where the payment is treated as a flow that can be freely allocated on all the LN links between the payer and the payee, given the capacity constraints of links. The problem has the following objective function, which aims at minimizing the amount paid by the payer (and therefore the fees):

$$g = \sum_{x \in N | (s,x) \in E} T_{s,x} \tag{1}$$

where s is the payer and $T_{x,y}$ is, in general, the amount (flow) of payment going from x to y.

The problem has the following constraints:

$$\sum_{x \in N | (x,y) \in E} T_{x,y} = \sum_{x \in N | (y,x) \in E} (T_{y,x} + f_{y,x}(T_{y,x})), \forall y \in N \tag{2}$$

$$T_{x,y} \leq r_{x,y}, \forall (x,y) \in E \tag{3}$$

$$\sum_{x \in N | (s,x) \in E} T_{s,x} - \sum_{(x,y) \in E} f_{x,y}(T_{x,y}) = \sum_{x \in N | (x,d) \in E} T_{x,d} = P \tag{4}$$

where N is the set of nodes, d is the payee and P is the amount to be paid; $f_{x,y}(T))$ is the fee function which expresses the fee to be paid to make a payment on the link x, y for the payment T. The constraints have the following meaning: cos. 2 is a flow constraint that expresses the fact that the amount of payments entering a node y is equal to the amount of payments originating from that node plus the fees required in ingress to make them;
cos. 3 is a capacity constraint that expresses the fact that the sum of payments flowing through a channel must not exceed the residual funds of the channel in the payment direction.;
cos. 4 is a couple of constraints that express the fact that a payment of cumulative value P must arrive at the destination d after has started from a source s.

We suppose, for generality, that the fee function f has the general form described in Sec. V. The fact that such fee function contains a fixed fee for each link makes the optimization problem NP-hard [5]. Such a problem can be solved, however, by means of the branch and bound technique [4]. This way of solving the problem, however, cannot be employed on the live network because of its computational intractability. Nonetheless, its resolution is useful to provide lower bounds (theoretical optimum solutions) to evaluate the solutions given by the multipath heuristic that we describe in the following section.

A. A multipath heuristic for payments on the LN

Given the computational intractability of the exact resolution of the multipath routing problem, in this section we describe a multipath heuristic to payments in the LN that is both fast and able to provide good solutions in terms of the paid fees.

The heuristic consists of k steps, with k being a parameter which represents the maximum amount of paths that can be used simultaneously for the payment.

The algorithm works as follows. In the first of the k steps, the best singlepath for the entire payment is evaluated by means of the available exact singlepath algorithm. The path

[3]The more the links on a path, the greater the probability of the payment failing at a certain hop, which causes the locking of the funds for a certain time. We do not focus on the problem of path length on this work.

[4]After it has easily been restated as a mixed-integer linear problem.

and the amount of fees are recorded, to be later compared with other solutions. In the second step, the best single path is evaluated for only half of the payment ($P/2$, where P is the original payment). If such best path is found, it is recorded along with the required fees and its links are removed from the graph. Then, a new best singlepath is looked for, again for a $P/2$ payment. If found, the resulting path and required fees are recorded as well. At this point, we have two link-disjoint paths, each carrying a $P/2$ payment and we proceed with an optimization step. Such step aims at modifying the amount allocated to each of the two paths, while keeping the paths fixed and the sum of the two payments equal to P. Such optimization problem can be made linear, given the following modifications (the second of which consists on relaxing the original problem): i) the base fees is not considered, as they are fixed given that the paths are fixed (they can be added at the end); ii) constraints on the payment values on the links of the used paths are added so that they remain in the linear range of their fee functions (that, we recall, are continuous piecewise linear). The solution of such optimization step, which is at least as best as the original solution, is recorded, together with the required fees.

The third and following steps are as the previous. Indeed, if we are at step i, the original payment P is split in i payments each of P/i value. i best single paths are then looked for (removing the links of each of the new paths when each of them is found). Once the i paths have been found, the optimization step is performed, as previously described, to change the allocation on each of the paths from P/i to different values that improve the overall solution.

When all the k steps are performed, the solutions obtained at the different step (for i there going from 1 to k) are compared and the best one is chosen (in terms of fees).

We note that at each step the procedure can fail, because it is not possible to find one of the needed best singlepaths. In such a case, the algorithm goes to the next step. Indeed, we note that while step i can fail, step $i+1$ can succeed, e.g. there are no two link-disjoint singlepath solutions each carrying a $P/2$ payment but there is a three link-disjoint singlepaths solution each carrying a $P/3$ payment.

The computational complexity of the algorithm is as follow. For each step i, i best single paths are searched, which has a complexity of a $O(i * |N|^2)$, where N is the number of LN nodes. As i is goes from 1 to k, the complexity becomes $O((k+1)/k|N|^2)$. We need, however, to also consider the k optimization steps. At each optimization step i, at worst 2^i solution points need to be verified, because for each of the i paths, at most two possible values for the payment can give the optimal solution. For the k steps, we have that the amount of solutions to be verified is less than $k * 2^k$. Overall, we then have a complexity which is $O((k+1)/k|N|^2 + k * 2^k)$, which considering the fact that k does not depend on the problem size (and is moreover a small number), gives us a complexity, in the worst case, of a $O(|N|^2)$.

VII. Simulation studies

A. Evaluation of the OptimizedFees policy

The first simulation study has the objective of demonstrating the advantage of the proposped fee policy (OptimizedFees) for LN nodes, which leverages the new way of specifying the fees that we propose.

To evaluate such an advantage, we use as metric, as previously anticipated, the *network imbalance* of the LN. The network imbalance is defined as the ratio between the sum of the imbalances of all the channels to the sum of all the LN funds. Such a measure represents the percentage of network funds that are allocated on just one endpoint of channels. As previously introduced, a balanced channel offers several advantages, such as the ability to perform payments with equal probability and sizes in both directions on each link and the possibility to keep the channel open for longer times.

We realized a simulator for the LN that we make available at [10]. Such a simulator is able to simulate the network at the LN protocol level. The simulated LN topology can be given as input or generated on the fly according to specific parameters, such as, e.g., the number of nodes, the number of sources and destinations, the probability of existence of a link between to nodes, etc. For our simulations, we used a LN topology with the following characteristics. The number of nodes were 200. Channels were created between each couple of nodes with a probability of about 3%. Funds were initially allocated to channels according to an uniform distribution having as limits 0.01 and 0.05 btc. Payments submitted to the network were generated according to a Poisson process with average interval between payments of 1s and a value obtained from a Gaussian distribution having different averages (and a variance of 0.0012). Sources and destinations of such payments were chosen randomly between all network nodes. We note here that such a choice (of having all nodes as potential sources or destinations) makes it less important for the routing algorithm to try to balance the network as payments naturally flow on all links on both directions and therefore tend to balance out the links. As far as the routing algorithm is concerned, we used the (exact) singlepath algorithm (at this point we just want to evaluate the effect of the OptimizedFees).

In Fig. 3 we show the overall imbalance during time (as payments were processed) for different configurations. Such configurations differ for i) the average value of payments, which is expressed as percentage of the average funds of channels (reported as number in the labels); ii) the fee policy applied, which can be the LN default, fixed fee plus a proportional part (1 satoshi as base plus 30 millisatoshis for each 1000 millisatoshis transfered), and the OptimizedFees, identified with the OF in the label (with 1 satoshi as base, 10 and 30 for each 1000 satoshis transferred for, respectively, the $slow$ and $shigh$ regions).

The results show that, when the average values of payments increases, the network imbalance increases as well, as expected. In particular, for an average of 50%, the imbalance can get as high as 30% (Exp50_sp). The results show, moreover,

that the proposed fee policy is able to significantly decrease the tendency of the network to become imbalanced. In particular, for payment of average size of 20% of channels' average fund sizes (Exp20 and Exp20_OF), the reduction of the imbalance is of about 49% (0.126 versus 0.249), while for payments of 10% (Exp10 and Exp10_OF) of channels' average size the reduction is of about 22% (0.229 vs 0.295).

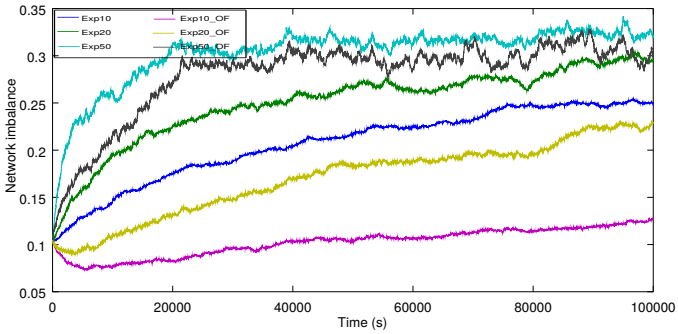

Fig. 3. Network imbalance for different fee policies and different average values of payments.

B. Evaluation of the multipath routing heuristic

The second set of simulations has the aim of evaluating the multipath routing scheme we propose. The simulations were performed with the same parameters as the previous simulations. In Fig. 5, the average fee for both the proposed multipath heuristic and the singlepath exact algorithm are shown. As can be seen from the figure, by adopting the proposed multipath heuristic a significant reduction in fees can be achieved.

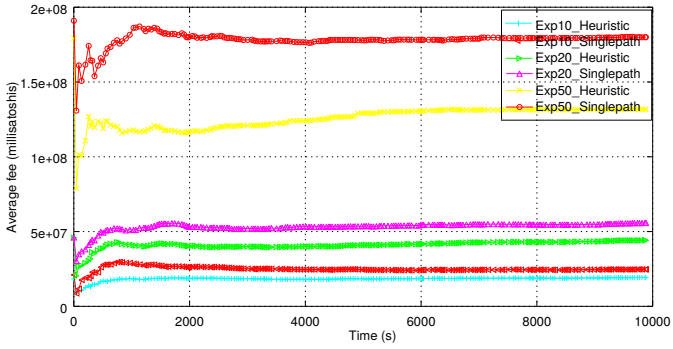

Fig. 4. Comparison among the proposed multipath approach and the singlepath exact algorithm.

C. Evaluation of the multipath routing heuristic against the exact algorithm

The third set of simulations has the aim of evaluating how good the results given by the multipath routing heuristic is in comparison with an algorithm that gives the optimal solutions (as described in Sec. VI). The simulations were done with LNSim simulator which, in case of the exact resolution, leverages the open source tool *gplk* [6] able to solve the problem applying the interior point method.

We must note here that differently from the multipath heuristic, which splits the original payment on several completely distinct smaller payments (because of the way the Atomic Multipath Payment method works), the multipath exact algorithm treats the payment as an end-to-end flow (which can then be split and reassembled at each node), giving even better solutions. Such a choice is meant to give the absolute best theoretical results in terms of fee that can be achieved on the network to use them to evaluate the proposed multipath heuristic.

The simulations were performed on the same scenarios of the previous simulations, but with different average values for the payments (from 10% to 50% of the payment channels average funds, in increments of 5%). The results, averaged over the different simulations, show that in the long term the fees obtained with the multipath heuristic are only $1/3$ greater than the ones obtained with the exact multipath algorithm. Such a difference can be partly explained by the fact that the multipath heuristic, as previously stated, cannot treat the payment as an end-to-end splittable flow, but only as different separate payments.

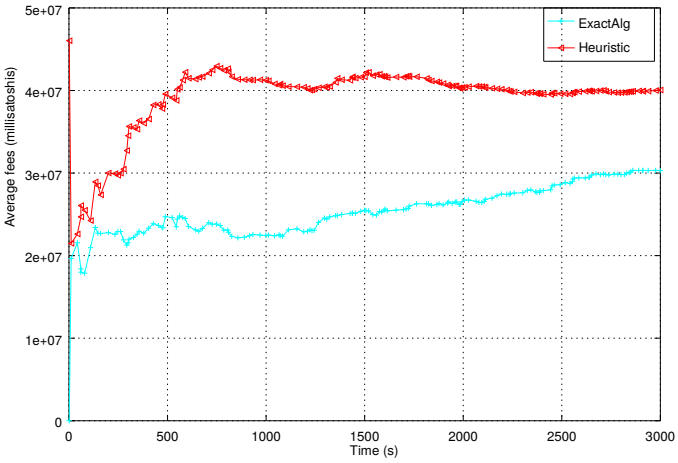

Fig. 5. Comparison among the proposed multipath heuristic and the exact algorithm.

VIII. Conclusions and Future Works

The lightning network approach to scalability is promising but important aspects of the technology need to improve before it can be widely and proficiently adopted. Indeed, the routing problem is a particular tricky one. On that regard, in this work we propose a new multipath routing heuristic for the LN which is able to significantly reduce the fees paid by the user. As the fees applied by nodes influences the routing decisions, we also addressed such aspect. In particular we propose a new fee policy that allows to keep network links balanced, which is beneficial for the functioning of the LN network in the long

term as, e.g. allows to keep payment channels open for longer times. As future work, we plan to study the scenario where the knowledge of the network is only partial, i.e. no complete view of the current balances or no complete knowledge of the fees applied on the channels. We, moreover, plan to extend the proposed multipath routing scheme to account for the possibility of automatically opening new channels between nodes not directly connected, in case such a choice improved some network wide or node operating variable.

REFERENCES

[1] Basis of Lightning Technology. https://github.com/lightningnetwork/lightning-rfc/blob/master/00-introduction.md, 2017. [Online; accessed 23-March-2018].

[2] Atomic Multipath Payments. https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2018-February/015708.html, 2018. [Online; accessed 23-March-2018].

[3] Block weight. https://en.bitcoinwiki.org/wiki/Block_weight, 2018. [Online; accessed 23-March-2018].

[4] F. Engelmann, H. Kopp, F. Kargl, F. Glaser, and C. Weinhardt. Towards an economic analysis of routing in payment channel networks. In *Proceedings of the 1st Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*, page 2. ACM, 2017.

[5] D. S. Hochbaum and A. Segev. Analysis of a flow problem with fixed charges. *Networks*, 19(3):291–312, 1989.

[6] A. Makhorin. Glpk (gnu linear programming kit). *http://www. gnu. org/software/glpk/*, 2008.

[7] P. McCorry, M. Möser, S. F. Shahandashti, and F. Hao. Towards bitcoin payment networks.

[8] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[9] J. Poon and T. Dryja. The bitcoin lightning network: Scalable off-chain instant payments. Technical report, Technical Report (draft). https://lightning. network, 2015.

[10] G. D. Stasi. Lightning network simulator. https://github.com/gdistasi/LNSim, 2017.