Delft University of Technology, Bachelor Seminar of Computer Science and Engineering

Adding Redundancy to Splitting Protocols for a Better Performance

Ivaylo Georgiev, Dr. Stefanie Roos, Oguzhan Ersoy

TU Delft

June 27, 2021

Abstract

Payment Channel Networks have been developed to deal with the scalability issue in blockchain technologies. Using them, two parties can make multiple payments between themselves relatively fast. However, usually the channels have too small capacities, unable to handle a big payment. Allowing to split a payment into smaller payments and forwarding them through different intermediaries is a way to solve this issue, but a party only knows the capacities of the channels it is connected to. Therefore, it is possible for a payment to be sent to an intermediary which would not have sufficient funds to forward it to another node, closer to the receiver. Making redundant transactions in order to further improve the payment success ratio is a way to handle this drawback. This paper provides 3 algorithms for adding redundancy to the already existing splitting protocol [1]. The evaluation shows that all of them improve the success ratio, but at the price of parties exchanging more messages.

1 Introduction

The biggest problem with cryptocurrencies like Bitcoin [2] and Ethereum [3] is that they lack scalability. Payment channels (PCs) [4] have been developed to tackle this issue. They allow the participating parties to exchange funds off-chain and append only the final agreement to the blockchain when the channel is closed. In order to create a PC, the two participants agree to use a certain amount of their on-chain funds only between themselves. Cryptographic protocols were developed that ensure both parties would not try to cheat the other one out of money. After the PC is established, the participants can make transfers of money by agreeing to a new distribution of the funds on the channel.

Payment-Channel Networks (PCNs) are created by connecting multiple channels. Thus, in case a party A needs to make a payment to another party B and they do not have an established PC among themselves, the payment can still be forwarded from the sender through one or several intermediaries to the receiver, forming a path from the sender to the receiver. One of the most widely used PCNs is Lightning[1] [5].

However, an intermediary on the path from the sender to the receiver might not have sufficient funds to forward the payment on any of the channels it is participating in. The way to overcome this issue is to split the payment over multiple channels [1]. However, this

[1]https://lightning.network/

comes with a number of security issues, such as ensuring that the intermediaries do not steal money during the transaction process. Cryptographic protocols were developed in order to tackle those problems.

When splitting the payments, however, a whole transfer fails if only one partial payment fails. This issue can be tackled by sending redundant payments [6] to the whole transfer. The main problem here is that the two protocols make and work on different assumptions (for example, in the splitting protocol the intermediary nodes decide how to further split the payment, while in the redundancy one the sender decides everything; also, the redundancy protocol splits the whole transfer into v transactions, while the splitting protocol does this as described in Section 2.1).

In this paper a combined protocol's design is described that tackles all the differences in the two protocols. As both protocols have multiple options used for splitting and adding redundant payments, there is a choice between different alternatives for (i) how to choose the next intermediary on the path, (ii) how to split the payment, and (iii) when and how to add the redundant coins in the payment. All of those options are further described and explained in the paper. The aim of the research is to compare the performance of the newly developed combined protocol with the splitting one, to check whether increasing the success ratio of payments comes at the cost of giving up on something else, and finally to make a conclusion whether it is worth it to combine redundancy with splitting in the real world or not.

In order to evaluate the newly developed protocol, its success ratio, as well as the average number of messages exchanged by two parties are measured using a simulation-based evaluation. It is then compared with the already existing splitting protocol using those metrics. It is observed that all implementations of the new protocol have a higher success ratio compared to the splitting protocol. The increase varies between 1% and around 22%, when different splitting algorithms, as well as different combined algorithm implementations are used (those are explained further in Section 2 and Section 3 respectively). However, the number of exchanged messages also increases, with the increase varying from 0.5 to around 150 on average (again, those results come from using different splitting and combined protocol algorithms). Only the static case scenario (where the topology of the network does not change in time, and the channel capacities are reset after every transfer) was simulated, so further simulations should be held in a dynamic environment.

The structure of the paper follows. In Section 2 the splitting and redundancy protocols are discussed more in-depth. Section 3 explains the design of the combined protocol. Section 4 gives the evaluation environment and explains the obtained results. Section 5 explains how the obtained results can be reproduced and in Section 6 a conclusion is drawn and future work is discussed.

2 Existing Protocols

In this section the already existing protocols that need to be combined are discussed in more detail.

2.1 Splitting Protocol

A lot of single- and multi-path routing protocols for PCNs exist [7, 8, 9], but most of them assume that the sender chooses the whole path, through which the payment should be routed. However, the sending party knows only the topology of the network and the capacities of the channels it participates in. Therefore, the chosen route might contain intermediaries, whose channel capacities are insufficient to further forward the payment, and thus result in a failed transfer. One of the most important innovations of the splitting protocol [1] is that the sender of the payment does not specify the whole path of the payment, but only the first hop. After that, all the intermediary nodes decide for themselves how to forward and split the partial payment they receive. Leaving the nodes on the path to decide where to forward the payment to will prevent the case in which the sender decides to send it through a channel with insufficient capacity. In order to do that, the sender and all intermediaries should be able to (i) determine the potential next nodes for the payment and (ii) split the payment over them.

There are 2 ways described in the paper for finding the set of potential next nodes on which the payment can be forwarded/split:

1. **Hop Distance** ($Next_{Hop}$) - it gives the length of the shortest path between two nodes. Payments are forwarded only to the nodes that are the closest to the destination (thus, making it possible to split only when there are 2 or more nodes that have the same distance to the target and this distance is minimal).

2. **Interdimensional SpeedyMurmurs** ($Next_{INTSM}$) - this is a modification of the SpeedyMurmurs [10] algorithm, made more suitable for splitting. It establishes Breadth-First Search spanning trees and uses them to decide on next hops. Contrary to the original SpeedyMurmurs, the Interdimensional version considers the spanning trees concurrently - if a node X wants to transfer a certain amount to node Y, it will consider a node Z as a possible intermediary if Z is closer to Y than X in at least 1 of the spanning trees (here, Z being *closer* to Y than X means that in the spanning tree there are less edges on the path from Z to Y than on the path from X to Y). Note that while constructing the spanning trees only a subset of the edges is used. However, when routing a payment every channel in the graph can be used.

After the set of next hops is created, a splitting strategy is used to split the whole transfer into partial payments according to a certain strategy. A total of 3 strategies are described in the paper:

1. **No Split** ($Split_{No}$) - the payment is not split and is forwarded to a single node.

2. **Split By Distance** ($Split_{Dist}$) - the candidate nodes are sorted in order from smallest to largest distance to the receiver with ties being broken randomly. Then, each node receives an amount equal to the minimum of the remaining channel capacity and the remaining partial payment.

3. **Split If Necessary** ($Split_{IfN}$) - the candidate nodes are sorted in order from biggest to smallest channel capacity. Then, just as in the *Split By Distance* case, each node receives an amount equal to the minimum between the channel capacity and the remaining payment to be made.

Then, for the whole splitting protocol, every distance function can be combined with every splitting strategy to make a complete algorithm.

2.2 Boomerang (Redundancy) Protocol

In the Boomerang protocol [6], contrary to the splitting one, the sender decides the whole path (meaning that intermediaries only forward the payment, they are not allowed to further split or reroute it). It also splits the whole payment into v equal transactions, using another u redundant transactions of the same amount, making it more likely for a whole transfer to succeed. Every time the receiver redeems a transaction, it reveals a key *p(i)* (i being the number of the transaction) to the sender. In case the sender gets to know more than v keys it can revert the whole transfer. Thus, it is most optimal for the receiver to redeem v transactions and no more.

In the paper, 3 different multi-path routing protocols are described:

1. **Retry** - the original v transactions are attempted, and at most u are reattempted in case any of the first ones fail.
2. **Redundancy** - $v + u$ transactions are attempted from the beginning. The transfer is successful if v of them arrive to the destination. According to the cryptographic primitives established in the paper, the destination node will not be interested in redeeming more than v transactions (since in this case the sender will have enough information to revert the whole transfer back).
3. **Redundant-Retry(10)** - a combination of the 2 previous protocols. It starts out with v + *min(u,10)* transactions in the beginning and reattempts at most u - *min(u,10)* of them.

3 Combined Protocol

In this section, four possible realizations of combined protocols are explained:

1. **No Redundancy** (*W/O*) - no redundant payments are made (same as the splitting protocol).
2. **Atomic Transactions without Splitting** ($Tran_{No}$) - the payment is divided into v equal atomic transactions. Then, they are sent through the network one after another, without the possibility of splitting them further (could be also interpreted as using the *No Split* strategy), using one of the two possible algorithms to find the next hop, described in Section 2.1. Failed transactions are rollbacked and can be resent at most u times.
3. **Atomic Transactions with Splitting** ($Tran_{Split}$) - the same as the second one, but the atomic payments could be further split. The payment succeeds only if all partial atomic payments are redeemed by the receiver, and the whole transfer is successful in case v payments succeed.
4. **Retry Amount** (*Amount*) - this strategy uses the *Retry* multi-path routing protocol described in Section 2.2. Firstly, the sender starts by forwarding the whole payment. In case a partial payment ends up in a node, in which the sum of the capacities of all outgoing edges is insufficient, then the node forwards as much as possible and the remainder is retried again from the beginning. The total amount of redundant payments u is decided in the beginning.

Figure 1 gives an example of how all 3 of the newly developed combined protocol realizations would work on a small network.

Note that $Tran_{No}$ and $Tran_{Split}$ do not require any new cryptographic proofs except the ones developed in the splitting and redundancy papers [1, 6] in order to be integrated.

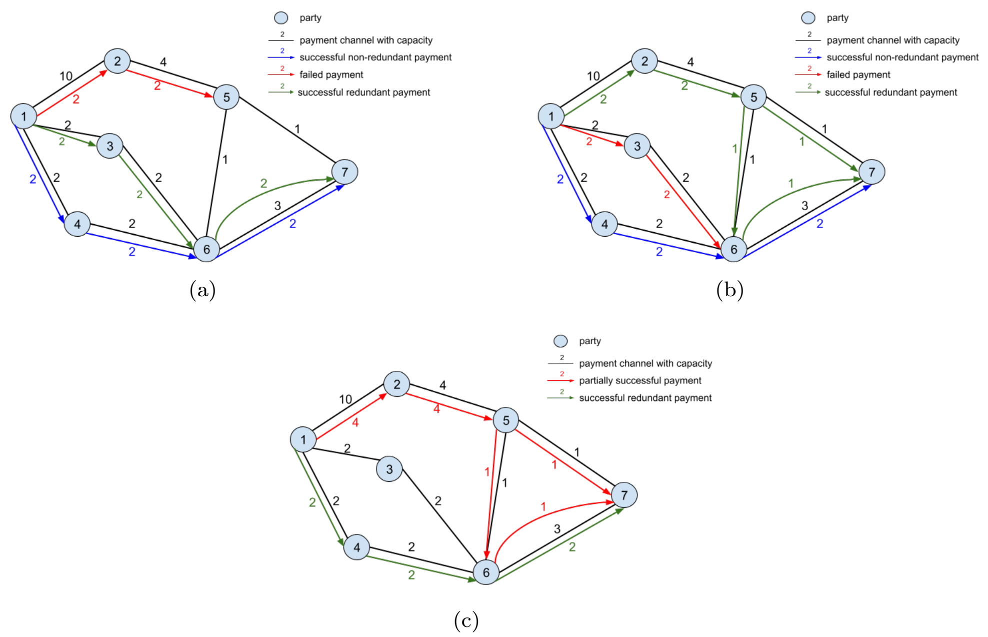

party
payment channel with capacity
successful non-redundant payment
failed payment
successful redundant payment
partially successful payment
(a)
(b)
(c)

Figure 1: An example of a payment of 4 from node 1 to node 7, using the a) $Tran_{No}$, b) $Tran_{Split}$ (in both cases, $v = 2$ and $u = 1$) and c) *Amount* (using $u = 2$) combined algorithm. In all 3 cases the successful payment (if existing) was first, then the unsuccessful and finally the successful redundant one.

Figure 2 shows the communication between the two parties. First of all, Boomerang's cryptographic preliminaries are used - the sender and the receiver agree on v and u, with the sender dividing the whole transfer amount into v equal transactions. The receiver also chooses a polynomial P of degree v with coefficients α_0, α_1, ..., α_v to be used for claiming a transaction:

$$P(x) = \sum_{i=0}^{v} \alpha_i x^i$$

Then, the receiving party commits to this polynomial by sending hashed values of the coefficients α_0, α_1, ..., α_v (the hash function has the property that by knowing H(α_i), obtaining α_i will be difficult, but if α_i is known, then $H(\alpha_i)$ can be easily computed). Learning the value of α_0 would mean that the sender can revert the whole transfer. After that, the splitting protocol's security properties are applied - the receiver sends its hashed random preimage to the sender. Then, the sender appends routing information (which includes the receiver's identity, as well as the number of the transaction) to the payment and forwards it to the next hop. When the receiver gets sufficiently many partial payments (which have the same transaction number and add up to the transaction value), it uses its preimage to unlock them and broadcasts $P(i)$ (i being the transfer number) to the sender to redeem them. The whole transfer is successful when the receiver claims a total of v transactions. Note that the hash function H is chosen in such a way, that retrieving more than v transactions (and thus, broadcasting more than v $P(i)$ values) will give the sender

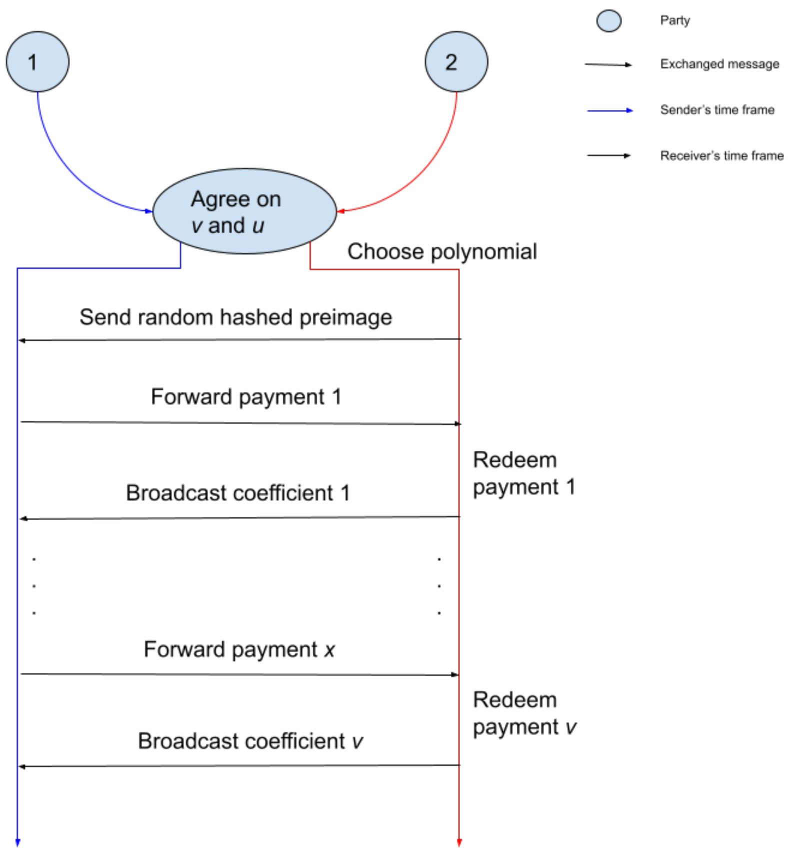


Figure 2: Message exchange between 2 parties throughout a transfer, with $v \leq x \leq v + u$.

the opportunity to calculate all the coefficients α_i for $0 \leq i \leq v$, thus making it possible for the sender to revert the transfer. Therefore, these protocol implementations are safe against cheating behavior from the receiver.

However, the already known cryptographic preliminaries and security models are insufficient for the *Amount* realization. In this algorithm, the payment is not split into smaller atomic transactions, thus making the redundant payments different in value (it is known that the sum of all redundant payments does not exceed a certain number, but different transactions can be of a different amount). This means that if this combined protocol implementation proves to be a better option than the $Tran$ implementations, then new security guarantees should be developed which prevent the receiver from claiming more money than it should.

4 Evaluation

Evaluation on the performance of the combined protocol in comparison with the existing splitting protocol is given in this section. In order to do that, a simulation study was conducted, in which the impact of the 3 combined algorithm realizations on the success ratio, as well as the communication overhead measured by the average number of messages exchanged by two parties, is analyzed.

4.1 Metrics

The most important and representative metrics that will be used are the *success ratio* and the average number of *messages* sent between two parties throughout one payment.

The success ratio is given by the ratio between the successful transfers and all transfers. It is expected that with redundancy more transfers will be successful and the success ratio will increase.

However, redundancy would also increase the number of exchanged messages in one payment. This would increase the overall time required for a transfer to finish in case the transactions are routed one after another (not concurrently). However, if they are forwarded concurrently, messages would be exchanged in a concurrent manner as well, so the time a transfer requires to finish will most probably not be affected as much.

Therefore, redundancy has both its positive and negative sides and they should be compared accordingly in order to make an adequate evaluation of the newly created protocol.

4.2 Simulation Model and Network

The current implementation works only in the static scenario, meaning that both the network topology and channel capacities remain constant throughout the simulation, being reset to their initial value after every transfer. This, of course, is not the case in the real world, therefore a simulation in a dynamic environment is strongly suggested in the future. Also, considering concurrency when routing the payments is vital (since when payments are made concurrently, adding more and more redundant payments does not necessarily increase the success ratio [6]).

The network topology[2] used for the simulation is taken from the splitting protocol paper [1]. It is a real-world Lightning snapshot from March 1, 2020, snapshot 04_00. The total number of nodes is 6329, while the average number of channels per node is 10.31. The channel capacity distribution, as well as the mean capacity value, are also taken from the paper, corresponding to the Lightning channel capacity in March, 2020[3]. The average capacity value back then was 200, with the distribution being highly skewed (most channels had really small capacities). Thus, the exponential distribution seems to be most suitable for this case, and is therefore used in the simulation.

4.3 Simulation Parameters

The parameters that could influence the performance include the amount to be transferred, the number of transactions v to divide the whole payment in and the number of redundant transactions u to retry in the first 2 combined algorithm realizations, and the total redundancy amount u in the last one, all of which are described in Section 3. For the Interdimensional SpeedyMurmurs distance function a total of 5 spanning trees were used in order to identify next potential hops, with the root being randomly chosen.

All experiments are averaged over 20 runs, where each run consists of 10000 transfers. The transfer amount follows an exponential distribution with expectation $1/\lambda = \{1, 5, 20,$

[2]https://git.tu-berlin.de/rohrer/discharged-pc-data

[3]https://1ml.com/statistics

50, 100, 200, 300}. The impact of this parameter was studied with performing a simulation using constant values for the redundancy parameters of the combined algorithms (and also for the initial number of transactions in the *Tran* implementations), namely $v = 25$ and $u = 10$ for the *Tran* algorithms, and $u = 100$ for the *Amount* implementation.

For the *Tran* algorithms, the influence of the number of initial transactions v, as well as the number of redundant transactions u was studied. In the performed simulation, the values that those two parameters took were $v = \{1, 5, 10, 25, 50, 100\}$ and $u = \{1, 5, 10, 20, 30, 50\}$, with u taking a constant value of 10 when v was altered, and $v = 25$ when modifying u. The transfer amount used is exponentially distributed with expectation $1/\lambda = 200$.

The *Amount* combined algorithm is influenced by its u parameter, showing the total maximum redundant amount. In the simulation, the value range for this parameter was $u = \{10, 20, 50, 100, 200, 300, 500\}$. As in the *Tran* simulation, the amount to be transferred in this case is also exponentially distributed with expectation $1/\lambda = 200$.

4.4 Results

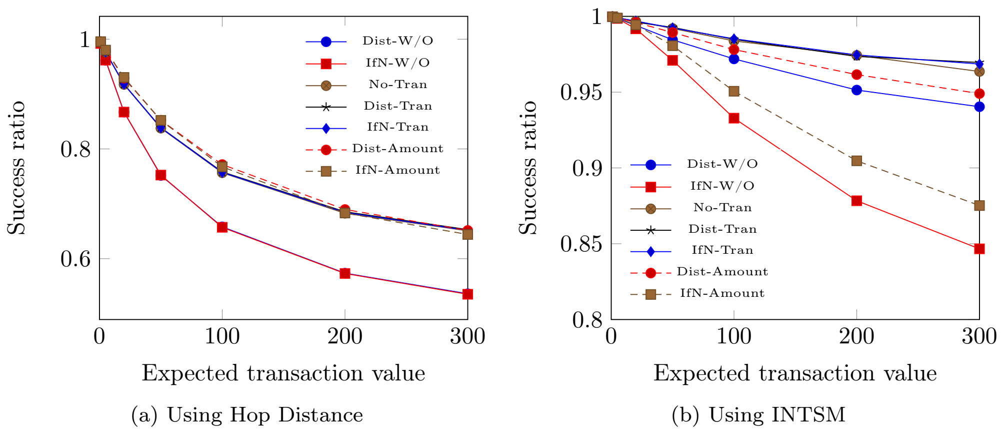
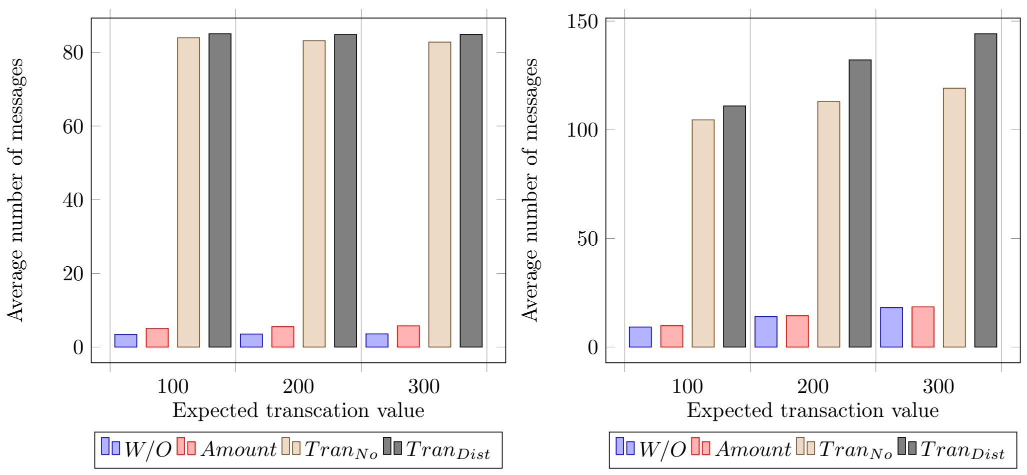


Figure 3: Success ratio for transfers with exponentially distributed transfer amounts.

Figure 3 gives the success ratio of the different combined algorithms using both the Hop distance function and the Interdimensional SpeedyMurmurs to identify possible next hops. The abbreviations indicated in Section 2 and Section 3 are used in the legend for readability. It is clear that in all cases adding redundancy increases the success ratio - about a 19% increase when using the *Hop* distance function, and a 2-14% increase when using *Interdimensional SpeedyMurmurs* (using different splitting algorithms and different distance functions has a big impact here). The reason behind that difference is that when using INTSM, the success ratio is pretty large anyway, so adding redundancy cannot make it much larger. It is also observed that actually using the *Tran* realizations give a greater improvement compared to the *Amount* one.

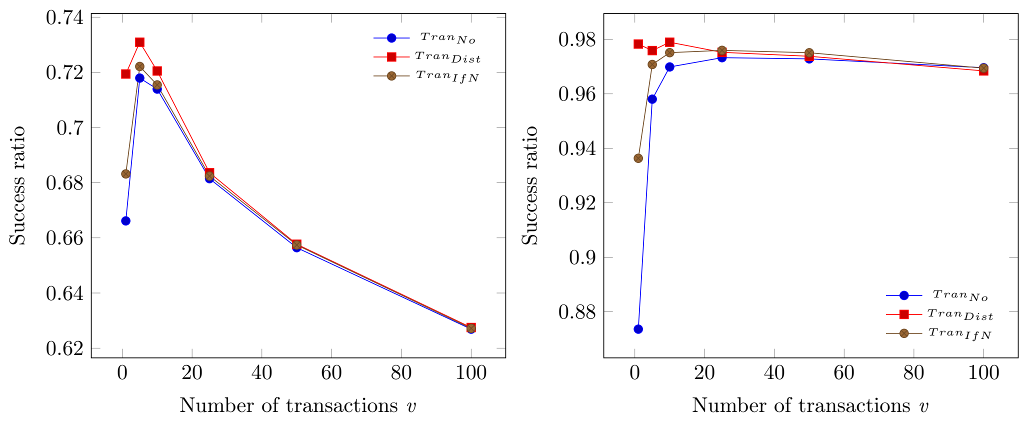
Figure 4 shows the average number of messages exchanged for 1 transfer for the different combined protocol implementations, with the transaction amount being exponentially distributed as mentioned in Section 4.3. As seen, the *Amount* realization makes just slightly more hops than the *W/O* one (about 2 more when using the Hop Distance, and about 0.5



(a) Using Hop Distance (b) Using INTSM

Figure 4: Average amount of messages sent for 1 full transfer using the $Split_{Dist}$ distance function.

more when using Interdimensional SpeedyMurmurs), while in the $Tran$ realizations much more messages are exchanged (for example, when using the $INTSM$ and the expected transfer amount is 300, in $Tran_{No}$ around 119 messages are exchanged and in $Tran_{Dist}$ - around 144, compared to only about 18 in W/O). This is due to the fact that the whole payment is broken into v tiny payments, which are then forwarded one after another, thus messages are exchanged for each of those small payments.

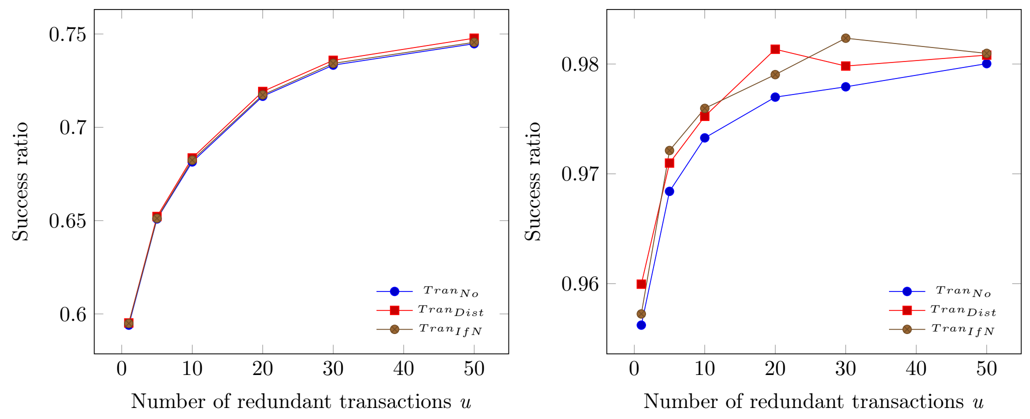


(a) Using Hop Distance (b) Using INTSM

Figure 5: Success ratio of the $Tran$ combined algorithm using different number of transactions v, with number of redundant transactions $u = 10$ and expected transfer amount being 200.

Figure 5 shows how does the success ratio of the $Tran$ algorithms change when the number of initial transactions v is altered. As seen, when using the *Hop* distance function, the most optimal v for all $Tran$ algorithms in this simulation is 5. This could be due to the fact that the total redundant amount decreases when v increases - the bigger the initial number of transactions, the smaller the amount of one transaction, thus the smaller the total amount of redundant transactions. Therefore, the amount of redundant transactions might not be sufficient in order to compensate for the failed ones.

When using the *INTSM* distance function the difference is not that significant when $v \leq 25$ (when $v > 25$ the success ratio drops because of the same reason as in the *Hop* distance function case). The reason behind this is that even without redundancy the total success ratio of the transfers was pretty high, so adding redundancy does not affect it as much as in the *Hop* distance function case. Therefore, making more but smaller atomic payments cannot affect the overall performance that much. Note that when using the *Dist* splitting algorithm the success ratio drops with 0.2% when v increases from 1 to 5. This is due to the randomness included in the splitting algorithm when breaking ties, and can be shown by the confidence intervals obtained for the success ratio in those two occasions. A confidence interval displays the probability that a parameter (in this case, the success ratio) will fall between a pair of values. When $v = 1$, the 95% confidence interval is $(0.976, 0.98)$, and for $v = 5$ - $(0.973, 0.979)$. As seen, they are really close, so it can be deduced that randomness in breaking ties is the reason for this small drop.



(a) Using Hop Distance

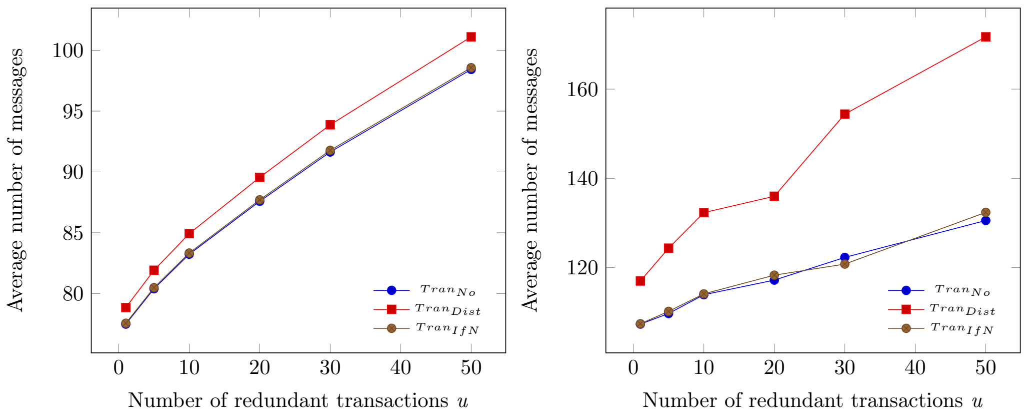
(b) Using INTSM

Figure 6: Success ratio of the $Tran$ combined algorithm using different number of redundant transactions u, with number of original transactions $v = 25$ and expected transfer amount being 200.

Figure 6 shows how the $Tran$ algorithms change their success ratio when the number of redundant transactions u is altered. Note that in the case of having concurrent payments adding more redundancy would not always mean a better success ratio because the payments might block each other. Therefore, conducting a simulation where payments are forwarded concurrently is vital for future work. However, in the static case scenario when only one payment is forwarded through the network at a time, it is expected that the number of successful transfers would increase when the total redundant payments increase - since in this case more partial payments could be reforwarded from the sender in case some others fail. From what can be observed in the figure, this seems to be the case. Only when using the

INTSM distance function with the total redundant payments being between 20 and 50 this is not the case. However, as mentioned before, the success ratio is pretty high there (roughly 98%), so this error is most probably because of the randomness introduced when breaking ties between the possible next hops. For example, when using the *Dist* splitting strategy and u increases from 20 to 30, the success ratio drops with 0.15%. The 95% confidence interval when $u = 20$ is $(0.9797, 0.983)$ and when $u = 30$ - $(0.977, 0.982)$. Again, they are really close to one another, so the success ratio drop would be due to the involved randomness.

However, even though it seems that adding more and more redundant payments is increasing the success ratio (in the static scenario), this also comes at a cost - the number of exchanged messages throughout one whole transfer increases.



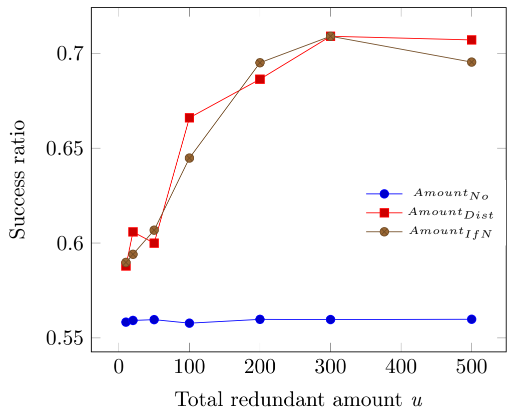
(a) Using Hop Distance

(b) Using INTSM

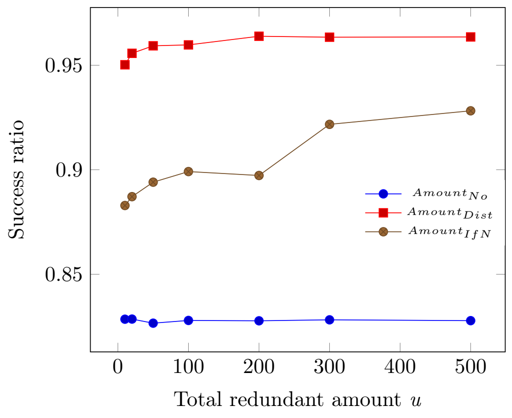
Figure 7: Average number of messages exchanged for the combined algorithm using different number of redundant transactions u, with number of original transactions $v = 25$ and expected transfer amount being 200.

Figure 7 gives information on how much does the average number of messages increase for the $Tran$ algorithms when the number of redundant payments u is altered. When using the *Hop* distance function we see that there is an increase of more than 10 when the redundant payments are 20 and 50 respectively, and when using the $Tran_{Dist}$ algorithm combined with *INTSM* this difference becomes much larger - more than 30 more messages. Therefore, the conclusion which can be drawn here is that with giving up a bit on the success ratio, the number of exchanged messages throughout one transfer would drop. In this particular case, having more than 20 redundant payments would mean unnecessarily increasing the number of messages in order to increase the success ratio with 0.3% or even less.

Figure 8 shows the success ratio of the *Amount* combined algorithm. As seen, adding more redundancy does not help when the whole transfer amount cannot be split. When the transfer fails, the whole amount is reforwarded from the sender again, thus the only hope is that it goes through another path (that is, picking a different neighbouring node when randomly splitting ties between possible next hops). In all other cases it seems that adding more redundancy until $u = 300$ affects the success ratio positively. If the *Hop* distance function is used, the success ratio increases by around 20% when comparing the cases in which $u = 10$ and $u = 300$. When using *INTSM*, it can be observed that the increase also depends on the splitting algorithm used. When the *Dist* algorithm is used the increase is



(a) Using Hop Distance



(b) Using INTSM

Figure 8: Success ratio of the *Amount* combined algorithm, using different total redundant amount u.

about 1.5%, in comparison with a 4.5% increase when using *IfN*. Note that, as in the *Tran* combined algorithm implementations, there are cases in which adding redundancy results in a drop of the success ratio. For example, one of these inverted cases is when u goes from 20 to 50 when using the *Hop* distance function and the *Dist* splitting algorithm. The 95% confidence interval when $u = 20$ is $(0.6, 0.611)$, and when $u = 50$ - $(0.58, 0.618)$. It is seen that the boundaries of the second interval are roughly on the same distance from the boundaries of the first one and that they are pretty close to one another. Thus, it can be concluded that, as in the previous cases, randomness is the main factor causing the success ratio drop.

In summary, the *Tran* realizations have an advantage over the *Amount* one both in the success ratio and in the fact that no new cryptographic protocols should be obtained in order to prove they can be implemented in the real world. However, the average number of messages exchanged between two parties throughout a full transfer is much larger when using the *Tran* implementations than when using the *Amount* one or the normal splitting protocol. This might not be a problem if the transactions are routed concurrently, so implementing and evaluating concurrent payments in the future would be really important here.

5 Results Reproduction

In this section the way to reproduce the results will be discussed. It is of great importance that other people interested in the blockchain technology are able to check the results for themselves, since this can lead them to further improve the presented combined protocol, or even implement it in the real world.

The solid foundation for the whole project was given in the splitting protocol implementation[4]. This is also the code used to obtain the results for the *W/O* realization of the combined protocol (which is exactly the same as the splitting protocol, used only for

[4]https://github.com/stef-roos/PaymentRouting

comparison with the other 3).

Implementation of the $Tran$[5] realizations and the *Amount*[6] algorithm are located in the same repository, but on different branches. However, in both cases the results are obtained in the same way - by running the *main()* method in the *src/paymentrouting/route/Evaluation* class. The *main()* method calls the *evalValTrees()* method. All of the parameter settings are located there. In both algorithms the *vals* array stores all the considered transfer values (or rather, the expected transfer values since they follow an exponential distribution). In the $Tran$ implementations, the *transactions* and the *redundant* arrays store the parameters v (the number of transactions, in which the whole transfer amount is divided) and u (the number of redundant transactions) respectively. The *Amount* implementation contains the *amountToRetry* array, in which the values used for the total redundant amount parameter u are stored. All of the aforementioned arrays currently contain the values used and described in Section 4. After running the evaluation, the results files appear in the *data/lightning-nopadding* folder.

The code was tested and runs under *Windows 10* and *macOS Big Sur* operating systems. No specific file paths were hard-coded and there are not any other machine-specific settings that prevent the implementations from running on any computer.

The results obtained by running the evaluation might slightly differ from those discussed in the paper due to the randomness involved in choosing the next hop when routing a payment. Unfortunately, providing the random seeds used by the random number generator in the evaluation phase discussed in this paper is not possible. However, standard deviation, as well as 95% confidence intervals are present in the results files. These can be used to deduce the possible error between two subsequent simulations.

6 Conclusion and Future Work

This section discusses further the obtained results in Section 4 and brings up ideas on some work that could be done in the future. As mentioned in Section 1, the goal of this paper is to explain how does adding redundancy to the existing splitting protocol [1] affect its performance and whether there are any issues that arise with the newly created combined protocol.

As seen, adding redundancy to the splitting protocol brings the successful payments ratio up. However, this also comes with a price - the nodes in the network will have to exchange more messages (since failed partial payments are reattempted from the beginning).

We have seen that using the *Amount* combined algorithm, the average number of exchanged messages is slightly higher than the splitting protocol's, while the $Tran$ algorithms use much more than it, making the whole transfer process much slower in the static scenario. This, however, also comes with a price - the success ratio of the *Amount* version when using the *INTSM* distance function is smaller than that of the $Tran$ realizations. Also, new cryptographic rules and security models would need to be obtained in order to prove that the whole combined strategy is secure, and that the sender and the receiver's greatest interest would be to respectively pay and claim exactly the amount of the whole transfer.

Therefore, the main goal for the future would be to firstly think on improvements over the *Tran* algorithms. One such improvement could be to send all of the transactions concurrently. In this way, the messages can also be exchanged concurrently, so the total time

[5]https://github.com/ivkontiny/PaymentRouting-1/tree/redundancy-no-split
[6]https://github.com/ivkontiny/PaymentRouting-1/tree/redundancy

one transfer takes would decrease. Also, using heuristics when deciding on the next hops would probably increase the success ratio further.

In case the abovementioned goal proves to be unsuccessful, then cryptographic proofs and security models should be obtained in order to prove that the *Amount* algorithm is feasible for integration in the real world. This seems to be a much harder task, but the benefits will be huge - getting a pretty big increase in the success ratio, while exchanging just slightly more intermediary messages.

References

[1] Lisa Eckey, Sebastian Faust, Kristina Hostáková, and Stefanie Roos. Splitting payments locally while routing interdimensionally. *IACR Cryptol. ePrint Arch.*, 2020:555, 2020.

[2] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot, 2019.

[3] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[4] Andrew Miller, Iddo Bentov, Ranjit Kumaresan, and Patrick McCorry. Sprites: Payment channels that go faster than lightning. *CoRR, abs/1702.05812*, 2017.

[5] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments, 2016.

[6] Vivek Bagaria, Joachim Neu, and David Tse. Boomerang: redundancy improves latency and throughput in payment-channel networks. In *International Conference on Financial Cryptography and Data Security*, pages 304–324. Springer, 2020.

[7] Pavel Prihodko, Slava Zhigulin, Mykola Sahno, Aleksei Ostrovskiy, and Olaoluwa Osuntokun. Flare: An approach to routing in lightning network. *White Paper*, 2016.

[8] Peng Wang, Hong Xu, Xin Jin, and Tao Wang. Flash: efficient dynamic routing for offchain networks. In *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, pages 370–381, 2019.

[9] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. Concurrency and privacy with payment-channel networks. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 455–471, 2017.

[10] Stefanie Roos, Pedro Moreno-Sanchez, Aniket Kate, and Ian Goldberg. Settling payments fast and private: Efficient decentralized routing for path-based transactions. *arXiv preprint arXiv:1709.05748*, 2017.