ASAP : a Multi Path Payment Channel Network Protocol

Advanced Blockchain Technology Course Project Report

by

Akash Kumar

(Roll No. 213050020)

Hrishikesh Saloi

(Roll No. 213050057)

Manoj Kumar Maurya

(Roll No. 213050067)

Guide:

Prof. Vinay Joseph Ribeiro

Computer Science and Engineering

INDIAN INSTITUTE OF TECHNOLOGY BOMBAY

2022

Abstract

In a payment channel network like the Lightning network Alice can transfer a payment to Bob by splitting the payment into partial payments and transferring these partial payments through multiple paths. The transfer, however, delays if any of the partial payments fails or delays. To handle this, one can add redundant payment paths. The challenge in doing so is that Bob may now overdraw funds from the redundant paths. Due to the stochastic nature of Payment networks, i.e., random delays and failures of payment paths, multipath routing often idles while waiting for a few straggling paths. This leads to a longtime-to-completion (TTC) of transfers. Furthermore, after success of the payment, the other redundant payments are again informed by the source to free up the liquidity. This leads to additional time for the completion of the payment steps. The redundancy increases the congestion of payments in the payment channel network. ASAP, a new PCN protocol is discussed which stands for Adaptive Split of a Payment. The proposed solution works to reduce the unnecessary division of payments into partial payments. Approach is designed for splitting the payment only when required i.e, the payment channel has insufficient balance. Splitting criteria algorithms are discussed to reduce the congestion by implementing an adaptive splitting protocol. The experimentation showed ASAP congestion is ten times less than the Boomerang [1] protocol and the success ratio being better than the LND protocol and nearly same as Boomerang.

Contents

Chapter 1

Introduction

Payment channel network (PCN) enable fast and scalable payments by moving transactions offchain thereby overcoming the scalability drawbacks of Blockchain. However if we allow payments through a single path like lightning network does, chances of failure increases due to unavailability of required amounts in the channel.

In lightning network, a user Alice can join the network by creating a connection with another user Bob, who is already in the network which is called a payment channel. Both the users need to lock up some fund to create the payment channel between them, after which they can transfer money between them. Now Alice can use the channel with Bob to send funds to other user say Chand. Alice will search for a path through Bob to send the fund to Chand. However the channels should have sufficient money to successfully run the transaction. For security reasons the balances of the channels are not disclosed to any user. So a payment may fail if there are not enough money in the channel which carry the funds. In that case, Alice has to search for a new path, which again is subjected to failure for the same reason. This process adds a non-negligible delay to the payment process.

To tackle the above, Alice supposedly splits the payment into smaller paths and transfer the partial payments to Chand through different paths. This method reduces the chances of failure. As the amount gets smaller, the chances of failure reduces. However as we increase number of path in the transaction, the transaction is pushed to depend upon more number of channels. If any of the channel of any of the path fails, the whole payment will fail. Splitting the payment into smaller payments before starting the transfer may lead to formation of unnecessary payments such that even the path could have been possible to accommodate higher amount, the protocol divides the payment into smaller values. This leads to congestion in the payment channel network. To address this issue Neu, Tse and Bagaria devised Boomerang. Boomerang extends the multi-path payment approach by adding redundant payments to handle the situation where failure of single partial payment may lead to fail the entire payment. Formation of redundant

payments are done by sending more amount than the original payment amount in such a way that destination cannot overdraw. If the destination tries to overdraw then it leads to inform the source node regarding a secret of destination such that source can then cancel the entire payment to stop the overdraw. The approach leads to congestion in the payment channel network due to the use of redundant payments. Also, the source needs to have more money in its balance to be able to suffice the redundant payments.

This paper discusses about a novel approach, ASAP (which stands for adaptive split of a payment) to adaptively and dynamically split payments only when required. The payment is designated to follow a path to transfer funds. Whenever a node receives the payment contract and tries to setup contract to the next neighbouring node in the designated channel and finds the channel to be having insufficient balance, then the node splits the payment into smaller values according to the splitting criteria discusses later in the paper. The split payments are setup with the neighbouring on the shortest path from that neighbour to the destination. The ASAP protocol reduces the congestion in Boomerang by ten times. The success ratio is nearly same as Boomerang and better than LND protocol.

Chapter 2 discusses about the Background literatures followed by our analysis on the papers been studied in the chapter 3. The novel ideas are introduced in the chapter 4. Chapter 5 explains the algorithms for the splitting criteria of payments. The experiments carried on the novel protocol alongwith the findings are discussed in chapter 6. The protocol is concluded in chapter 7 where the future work is been discussed.

Chapter 2

Background Study

2.1 Boomerang

To address the problems with the multi-path payments, Bagaria, Neu, and Tse introduced Boomerang, a mechanism based on secret sharing and homomorphic one-way functions, which allows Alice to revert the transfer if Bob overdraws. Boomerang uses a homomorphic one-way function to intertwine the preimage challenges used for HTLC-type payment forwarding, such that Alice learns a secret of Bob iff Bob overdraws funds from the redundant paths. Funds are forwarded using Boomerang contracts, which allow Alice to revert the transfer iff she has learned Bob's secret.The author justified its theory by simulating it by implementing a prototype of its concept where it outperforms other multipath algorithms.

Boomerang [1] is generic technique used to increase the throughput of multipath routing scheme by using the idea of sending redundant payment. Figure 2.1 [1] shows the transfer of v amount from Alice to Bob happens with redundant payment of u amount leading to transfer total amount w=v+u. It uses a secret sharing mechanism where the sender learns a secret whenever receiver tries to overdraw and the sender can revert and cancels all the partial payments. The Boomerang contract works on two components. Forward component is responsible to make payment reach the receiver but this liquidity is locked until sender free up the reverse component after all successful payment is done without overdraw attempt from receiver.

Backward component is responsible for the reverse payment where sender reverse all payment made to receiver when sender learns a secret of receiver.The protocol divides the original payment into partial payments of equal amount. The introduction of redundant payments increases the probability of successful payment even when some of partial payment gets failed.

Example: After dividing payment amount 'A' into v payments of amount 'A/v', we send 'u' more payments of size 'A/v'. We may call the total number of partial payments sent as $w = v + u$

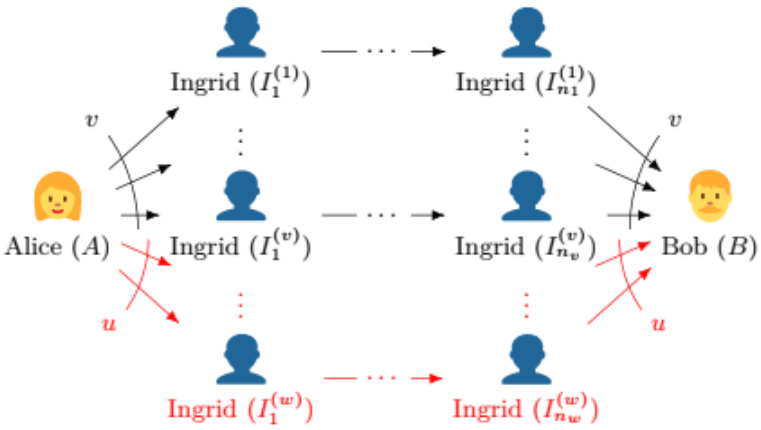


Figure 2.1: Transfer of amount v from Alice to Bob

where u is the number of redundant payments. The protocol implements a reversible HTLC-type forwarding where the payment gets reversed when the sender learns a secret of receiver. The receiver chooses a polynomial function defined as

$$P(x) = \sum_{j=0}^{v} \alpha_j * x^j$$

and informs the sender about $H(\alpha_0), H(\alpha_1), \ldots, H(\alpha_v)$ where H is an one way homomorphic function. For each payment $i = [1, 2, \ldots, v, \ldots, w]$ using the homomorphic property:

$$H(P(i)) = H(\sum_{j=0}^{v} \alpha_j * i^j) = \prod_{j=0}^{v} H(\alpha_j))^{(i^j)}$$

Each of the partial payment gives one polynomial:$\alpha_0 + \alpha_1 * i + \alpha_2 * i^2 + .. + \alpha_v * i^v$

For each payment 'i', receiver has to provide the sender with $H(p(i))$ value. For this the receiver gives the 'i' value out of band and as the sender already has $H(\alpha_j)$ for $j = 1$ to v, the sender can directly compute the value of $H(p(i))$ by substituting these hash values. According to the boomerang contract, to redeem the payment transaction receiver has to reveal the $P(i)$ corresponding to $H(p(i))$. When the receiver tries to overdraw amount, he has to reveal $P(i)$ for more than v transaction. Using the system of linear equation rule, the sender can now compute to get the v+1 unknown coefficients using more than v equations. Hence the boomerang contract uses the α_0 value to reverse the payments made in the forward components.

The payment procedure completes in four steps:

Promise: Sender attempts w = v + u Transactions.

Deliver: Receiver reveals the preimages P(i) to the corresponding preimage challenges $H(p(i))$ to claim the payment which activates the forward components of the Boomerang contracts along the respective paths.

Cancel: Receiver after completing v transactions, requests to cancel all outstanding transactions.

Finish: As after cancel step, overdraw is not possible so sender renounces all reverse components of Boomerang contract to free up liquidity.

Boomerang Evaluation: Boomerang is evaluated in a low-fidelity prototype implementation of the routing components inspired by the testbed of Flash. The experiment is conducted on N=100 nodes. For each payment channel endpoint, its initial liquidity is drawn log-uniformly in $[log(100), log(1000)]$. Source and destination are sampled uniformly from the N nodes. Experimentation is performs on v= 25 partial payments. Three multi-path routing protocols are compared: 'Retry', which initially attempts 'v' number of partial payments and reattempts up to 'u' of them. 'Redundancy', which attempts $v+u$ TXs from the start. 'Redundant-Retry' is a combination of the two, which starts out with $v+min(u,10)$ partial payments and reattempts up to u min(u, 10) of them. Throughput, i.e., total amount of successfully transferred funds per runtime is measured. 'Redundancy' and 'Redundant-Retry' show a 2x increase in throughput compared to 'Retry'.

Average time-to-completion (TTC) of a successful transfers which determines the latency experienced by the user and for how long liquidity is tied up in pending multipayment transfer. The result showed a 40% reduction in TTC for 'Redundancy' over 'Retry'. For u $\leq$ 10, 'Redundant-Retry' is identical with 'Redundancy' and for u>10, 'Redundant-Retry' is identical with 'Retry'.

2.2 Spear

Spear is based on the same analogy of boomerang but with exclusive improvements in various aspects. Spear can be implemented with minor change in Lightning Network. The minor change gives the control over the release of partial payments to both Sender and Reciever of the payments. Boomerang was suffering from problems like higher computation and heavy amount of locktime etc., Spear almost reduced the computation by a significant amount, and half of the maximum locktime of Boomerang. Unlike Boomerang it gives control on partial payments to sender and receiver due to this change overdrawing by receiver won't be possible. Another important feature in Spear method is robustness to the malicious intermediate node, it handles the problem of not sending payments by intermediate node or unnecessary delaying of the payments. Spear supports the unequal division of partial payments, it gives more flexibility on partial payments over equal partial payments.

2.2.1 Spear fund transfer procedure

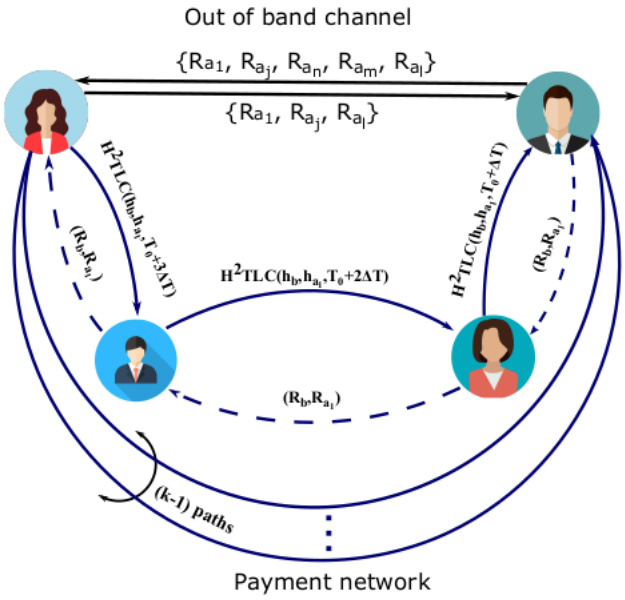
To make payment in Spear takes following steps:

Step1: Alice will receive an invoice from Bob stating the amount F that has to be paid and a hash digest h_b using an Out of Bound channel. In the Spear method partial payments can be divided unequally as mentioned earlier. Moreover there is no agreement needed to division of partial payments between Alice and Bob.

Step2: Alice selects k paths in payments channel network to send the payments to Bob. For each path, she sets an amount of partial payment, and a unique hash digest by applying a cryptographic hash function to a secret unique to the path. She then initiates the transfer of the partial payments all together.

Step3: Using the out of band channel, Bob informs Alice of the partial payments he has received and request preimages to claim the payment. In response, Alice will reveal a subset of her preimages to Bob. To prevent Bob from overdraw- ing, Alice makes sure that the sum of the partial payments corresponding to the released preimages is equal to F.

Step4: Let P be the set of partial payments whose preimages have been released by Alice in Step 3. In Step 4, Bob claims all the partial payments in P if (1) The sum of payments in P is at least F, and (2) Bob has enough time to claim all the payments in P. While claiming the partial payments, Bob cancels any received redundant payment.



Making a payment in Spear

Implementing Spear is relatively simple. In Spear, each partial payment is handled similar to the conventional single-payment scheme with only one exception: the transfer of payment on channels must be conditioned on the release of two preimages instead of one. We emphasize that, in Spear, all other parameters such as timeouts remain the same as the conventional single-payment scheme.

Chapter 3

Our Analysis on the literatures

3.1 Problems With Multipath Payments

Payment failures: Although the payment amount is been reduced due to division of payment into partial payments, the chances of failure of payments are still high. The partial payment's value are decided apriori which may still lead to cases of insufficient balance of channels.
Congestion due to unnecessary payment division: The partial payment value is decided apriori which may lead to unnecessary division of payments when the channels in the shortest path could have satisfied the original payment amount.

3.2 Problems in Boomerang:

Payment latency and Lock Time[2]: The time latency needed to complete the payment in Lightning network (LN) is let's suppose t then For Boomerang it is twice or 2*t that is higher than LN so it needs more locktime Therefore, in the worst case (e.g., as a result of a griefing attack) funds can be locked for a longer period of time in Boomerang.
Computational overhead: In Boomerang, Alice requires to compute at least v^2 field multiplications per partial payment. Bob has to compute a finite field exponentiation per partial payment, with each exponential requiring at least q field multiplications, where q is the order of the finite field in bits.
Network Congestion due to redundant payments: Although we use redundant payments in order to overcome the situation of re sending the payment in case of payment failure but for higher payment amount, introducing large number of redundant partial payments will lead to congestion in the payment channel network.

Increase in number of hash locks: While sending partial and redundant payment, a large set of channels are dedicated to hash lock the payment amount for the corresponding payment paths. Not all the locked amounts will be used to send the intended amount but we still lock it.

Node minimum balance: Assuming the sender needs to send δ amount as redundant payments to transfer a payment of amount V. To suffice this condition, the sender must have atleast V+δ balance in the payment channel network for the transfer of payment.

3.3 Analyzing the possibility of Timing Attacks[3] on Privacy in Boomerang:

Timing attack is been tried by the intermediate node in a payment channel network to infer the likeliest payment endpoints. The adversary intermediate node may try to deanonymize the sender and receiver based on the payment amount, timestamp values and the HTLC's time-lock delta value. By sending set of payments with destination in incremental hop count, the adversary nodes computes the path latency and edge latency measurements. These measurements helps to give likelihood estimation of possible destination. But for source estimation, the adversary drop the payment causing payment failure. As per the general used protocol, the source will instantly retry to send the payment giving one more chance to the same path which failed. The adversary measures the difference between the timestamp of payment failure and the time when the retrial of payment again reaches the adversary node. And comparing the values with the precomputed path latency and edge latency measurements, the likelihood estimation of source is computed.

After analyzing the timing attack on Boomerang, we found that deanonymizing destination is still possible as the adversary after precomputing the latency measurements can still find the likelihood estimation for destination. But deanonymizing source is not possible due to the use of redundant payments. When, a payment fails, the source does not retry the payment as the redundant payment are already present to suffice and overcome the payment failure. So as the adversary cant compare the timestamps to find the round trip time, hence the source cant be deanonymized.

Chapter 4

Our Novel Ideas

In order to tackle all those above problem, we proposed the idea of split if and only if necessary. After analyzing the timing attack on privacy in Boomerang, we concluded that deanonymizing Destination is possible. So in our designed protocol, the Destination information will be shared to the intermediate nodes to allow them finding paths while splitting.

Let say A need to transfer fund to C, A will send money to its immediate neighbour B. If there is enough money in channel B-C, B wont split the money. But if there is not enough money, B can split the money for its neghbouring nodes, find shortest path and then send the split payment through different paths to destination. Two different ideas are approached to do this;

1. Let say 'A-B-C-D-E-Dest' is a path , The only duty of B is to send the desired money to C. Now B will decide whether it is necessary to split the fund. After C gets the fund, C will accumulate the partial payments and send to D. The intermediate nodes will always be updated about the destination node

2. A will transfer the money to B. Now its B's duty to send the fund to Dest, accordingly it will find its path and decide whether to split or not. The partial payments gets accumulated in the destination.

This approach will save the network from unnecessary congestion due to redundant payments. we expect this approach will reduce payment failure as payments gets splitted if there is not enough fund in any channel. This approach will save us from griefing attack as taking place in BOOMERANG. However the affect in locktime need to be studied.

In the first approach, suppose the intermediate node needs to send the payment to neighbouring node 'C' for the transfer of payment to the destination node 'D'. When the channel between the intermediate node and C is insufficient to transfer the amount, the intermediate node may send the payment to the designated neighbouring node by splitting payments through various paths. It may be possible that the payment is splitted to transfer from the intermediate node to C passing through D leading to unnecessary transfer of funds.

In the second approach, the destination is deanonymized as the destination can be deanonymized through the timing attacks. The channel fee is not considered in analysis of the approach. Suppose, the source 'S' wants to transfer funds of value 'U' to the destination node 'D'. Consider an intermediate node B who receives the split payment 'P' with amount 'V' and is designated to transfer V to the destination. B is designated to transfer to the neighbouring node 'C' but the channel B-C has insufficient balance in its channel. The protocol will split the payment value 'V' using the splitting criteria algorithms to generate split payments for its channels with the neighbouring nodes. B computes the shortest path from the neighbouring nodes to the destination where the channels already used in the particular channel or its parent payment(payment split at least one time to form this payment P). The payment is setup on the channels of those neighbouring nodes for the shortest path computed from that neighbouring node to the destination for the amount of split payment value computed using splitting criteria.

The split is done adaptively and dynamically whenever required to avoid unnecessary congestion and redundant payment hash-locks in the payment channel network. The source doesn't need to have balance accommodating the redundant value to transfer the fund. After the successful completion of payment, Boomerang needed to send cancel request to all channels who are hash-locked being a redundant payment value. Our protocol doesn't require that extra time to finish the transfer. The computational overhead is reduced in our protocol as compared to Boomerang which works extensively with computational overhead.

4.1 Implementation of the new PCN protocol: ASAP "Adaptive Split of a payment"

Step 1: Alice receives an invoice from Bob through an out-of-band channel. The invoice includes the amount that Alice has to pay and a hash digest h_k We remark that, unlike Boomerang, Alice and Bob are not required to agree on how will be divided into partial payments;
Step 2: Alice finds the shortest path to Bob and sends the amount (V) to immediate neighbour 'A' with the path details.
Step3: 'A' tries to follow the path sent by the Alice,
If: Sufficient money is present, proceed.
Else: Split the money for the neighbouring nodes according to splitting criteria and find the shortest path from that neighbouring node to the destination where the channels involved in this payment or its parent payment(payment split at least one time to form this payment) are ignored to avoid any case of looping in the transfer of payment. The split payment are designated to transfer the payment on that computed shortest path.
Step 4: Bob may receive money from different paths, when the amount received by Bob gets summed to required amount then Bob releases the secret key(k) to all split payments.

4.1.1 Loop Avoidance

There may be situations where a new path traversed during the split of payment may use the incoming channel to that current node leading to the formation of loop in the path. In order to avoid this situation we modify the HTLC where we introduce a new field **Path_used** that will contain the id's of all channels that has been used by a payment till the point of reaching a node.

4.1.2 Split Stopping Criteria (β)

Splitting payment to smaller amount in case of repeated splittings may lead to unnecessary congestion. There has to be some point after which a payment does not split further and result

in payment failure. We use β as a hyper-parameter to define stopping criteria.

4.1.3 Payments received at destination

The destination node have to wait to receive the entire amount sent by the sender before releasing the secret key to the contracts. Suppose a malicious node 'x' receive a payment with amount V and the designated channel is not having enough amount to transfer. Assuming that x needs to split amount u and V-u in two payments. Node x can setup one payment and not setup the other split payment. The If the destination reveals the secret key before getting the complete payments, node x may still receive the secret key through the one contract and will be able to steal the remaining money.

Chapter 5

Splitting Criteria:

The payment needs to be split when a channel does not have the required balance to setup the payment contract. We have proposed three algorithms of splitting . Assuming a payment 'P' with amount 'V' been received by an intermediate node 'A' from node 'S' and node 'A' has payment channel with nodes $S, B_1, B_2, \ldots B_k$.

5.1 Symbolic notations:

P = Payment / Contract

V = Amount in the payment

A= Current node

S= Node which transferred P to A

B_i= Neighbouring node of A except S

k= Number of Neighbouring nodes of A except S

a_i= balance of A in channel A to B_i β = Split Stopping criteria

Alpha = fraction of V unused by all the neighbouring nodes for splitting of payment during Criteria 1

L_i = The set of paths computed from a $node_i$ to the destination

l_i = Number of paths in L_i

γ_i = Minimum balance to be kept for channel from current node to its neighbouring $node_i$

AmtLeft = Amount left from V to be splitted by the current node

$Split_i$ = Splitting value for channel from current node to its neighbouring $node_i$

$Path\_used$= that will contain the id's of all channels that has been used by a payment till the point of reaching a node.

$\Theta = 1 - \alpha$ = fraction of money taking from a channel

5.2 Criteria 1:

The node 'A' computes the sum of (balance-γ) at each payment channel to the k nodes except 'S' (incoming edge), that can be split satisfying the split stopping condition. If 'V' is greater than the sum computed then the payment cannot be proceeded leading to payment failure. The node 'A' which lacks the required balance 'V' in the designated channel computes one shortest path from each of the neighbouring nodes except the node 'S'(incoming edge). While computing the shortest path, the payment channels been used in the current payment and its ancestor payments (payments suffering zero or more splits to form 'P') are ignored to avoid looping. This means the channels on the path which lead the payment 'P' to reach 'A' are not considered while calculating the shortest paths. For each of the k neighbouring nodes, equal percentage of balance is left unused. After splitting the 'V' for each of the k neighbouring nodes, the HTLC contract is setup with the split amount for the shortest path that is computed by A from B_i ($i = 1$ to k). Consider the unused percentage of balance to be α and the balance of the channels of 'A' to the node B_i (for $i = 1$ to k) to be a_i,

$V = \sum_{i=1}^{k} (a_i * \frac{\alpha}{100})$

$\alpha = \frac{\sum_{i=1}^{k} a_i * V}{\sum_{i=1}^{k} a_i}$

Split money for the channel to node $B_i = a_i * \alpha$. The splitting has been done on an equal proportion for all paths and due to splitting the money on all the paths, the a_i becomes smaller which makes less possibility of payment failure than the received 'V'.

Algorithm 1 Algorithm for Criteria 1

Initializing **AmtSum= 0**

for *each neighbouring node* B_i *except* S **do**
  AmtSum+=(balance-γ_i)

end

if $V > AmtSum$ **then**
  payment failure

  Return;

end

$\alpha = \frac{\sum_{i=1}^{k} a_i * V}{\sum_{i=1}^{k} a_i}$

for *each neighbouring node* B_i *except* S **do**
  SP_i= Computed shortest path from A to B_i such that the payment channels been used in the current payment and its ancestor payments (payments suffering zero or more splits to form 'P') are ignored to avoid looping

  if SP_i==*null* **then**
    payment failure

    Return;

  end

  $Split_i = a_i * \alpha$

  Transfer $Split_i$ amount in path SP_I

  Add SP_i in $Path_used$

end

Disadvantage of Criteria 1:

1. Splitting the payment on all the neighboring nodes may lead to network congestion.

2. Since number of paths is increased, chances of single point of failure increases.

5.3 Criteria 2:

Consider the same payment scenario discussed in criteria 1, One shortest path is computed for each of the neighbouring node B_i to the destination. While computing the paths, the payment channels that has been used in the current payment and its ancestor payments (payments suffering one or more splits to form 'P') are ignored. The split payment if required will be setup in the computed shortest path. Consider the set of paths computed from node B_i to destination to be L_i. The number of channels used in each path from B_i to be denoted by $|L_i| = l_i$. Assume $l_1 > l_2 > l_3 > \cdots > l_k$.

Algorithm 2 Algorithm for Criteria 2

Initializing **AmtLeft= 0, i=1**

for *each k neighbouring node* **do**

$Split_i = min[\mathrm{a}_i - \gamma_i, (\frac{l_i}{\sum_{j=1}^{k} l_j}) * V]$

AmtLeft += $[\frac{l_i}{\sum_{j=1}^{k} l_j} * V] - Split_i$

end

while $i \leq k$ & *AmtLeft* > *0* **do**

if $(a_i - \gamma_i - Split_i)$ > *AmtLeft* **then**

$Split_i +=$ AmtLeft

AmtLeft = 0

Break;

end

else

AmtLeft $-= a_i - \gamma_i - Split_i$

$Split_i = a_i - \gamma_i$

$i = i + 1$

end

end

if *AmtLeft* > *0* **then**

payment fails

end

A computes one shortest path from each of the neighbouring nodes $B - i$ except the node 'S'(incoming edge). While computing the shortest path, the payment channels been used in the current payment and its ancestor payments (payments suffering zero or more splits to form 'P') are ignored to avoid looping.
The used channels for the payment are added in $Path\_used$.
The amount is transferred in the shortest path from that neighbouring node B_i to the destination.

The split criteria is based on the total number of paths that can lead to destination. The criteria works by more paths leading to more ratio of amount to be used in the $Split_i$ for the i'th neighbouring node. Having more number of paths is a reliable approach in terms of reducing payment failures.

Disadvantage of Criteria 2:

1. Splitting the payment on all the neighboring nodes may lead to network congestion

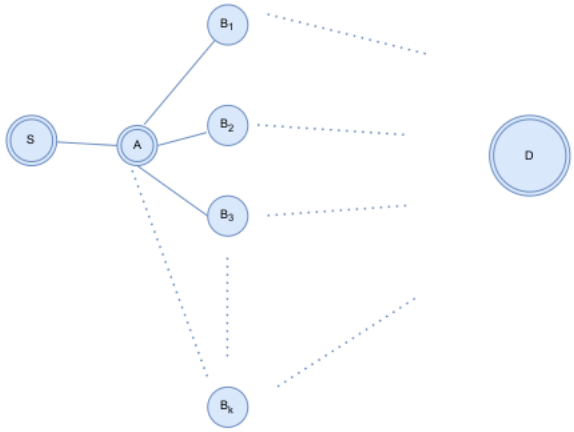
2. Finding all possible path on all the k neighboring nodes lead to higher time complexity.

3. Adaptive nature of splitting is not quite satisfied.

5.4 Criteria 3

The first and second algorithm may eat up a lot of time of a node to repeatedly finding out all possible paths in case of payment failure. Also they may lead to the high congestion in the network. To decrease the time complexity of calculating all shortest path, network congestion and also to bring in the adaptive nature of splitting, we propose third and final algorithm

Figure 5.1: Splitting payments in different paths



V= money to be split,

A= node at which split is initiated, A-B1, A-B2, A-B3 .... are multiple paths from A to Destination(D) and a1, a2, a3 ...are funds available in the channels respectively.

Idea: there will always be some amount left in the channel after taking the money (minimum balance of channel) and if split satisfies the stopping criteria, it stops the split and payment fails.

Algorithm 3 Algorithm for Criteria 3

$X = V$; // X is remaining balance after splitting which is equal to original amount initially

while *remaining amount to be split(X)* $> \beta * V$ **do**

Find the next shortest path;

if *no path available* **then**

payment fails

Break;

end

if $(a_i == 0)$ **then**

continue; //if there is no money in the current path skip this path

end

if $(X < \theta * a_i)$ **then**

transfer "X" amount from this path

//this means this path contains the required amount and no need to split further

Add the used channel in $Path_{used}$

Break;

end

else

Take $\theta * a_i$ from channel A-B_i ;

Transfer $\theta * a_i$ amount with this path;

$X = X - \theta * a_i$; // update the remaining amount

update the connection matrix by removing the used edge;

Add the used channel in $Path_{used}$

end

end

Where α, β are the hyper-parameters

$\theta = 1 - \alpha$ = fraction of money taking from a channel

β = fraction of original money 'V' after which it cannot be split further

Note:

1. Splitting may take place in the source too

2. Once destination gets the required amount, it reveals the secret key to the nodes through which it received the money. Then those nodes will reveal the secret key to their incoming nodes.

3. Splitting stops when amount after splitting(X) $< \beta * V$ (original amount) Or if some amount is left after using all the path.

4. When calculating the next shortest path , the channel previously used is not considered.

5. When shortest path is computed from a node, the channels that are included in the used path field in HTLC are not used. This is done to avoid formation of loop.

6. If a channel has sufficient amount to send money, the payment will not get splitted. This makes our algorithm adaptive and not unnecessarily split the money if not required.This feature will also reduce congestion.

Chapter 6

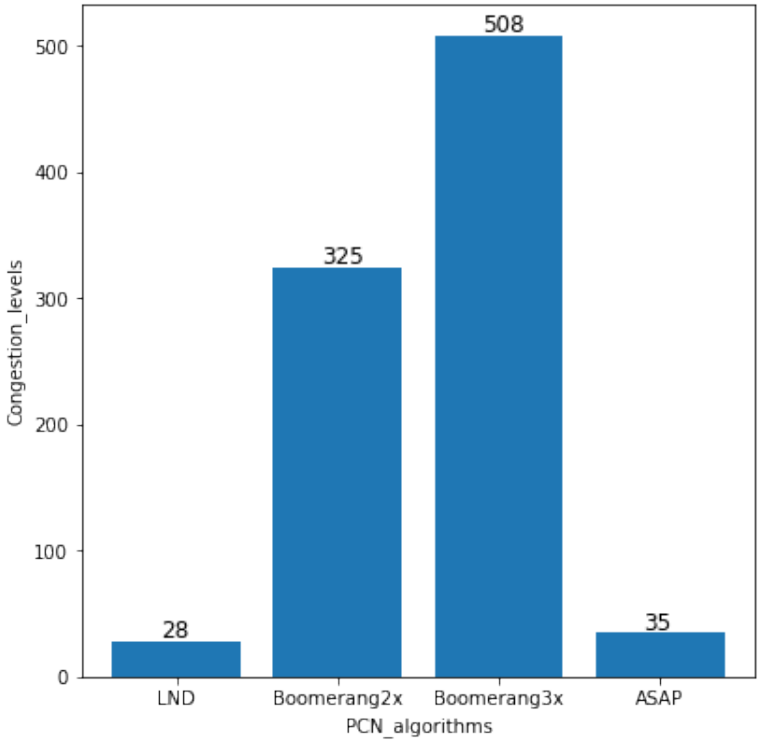
Experiment and Findings

We are comparing LND, Boomerang and ASAP PCN algorithms.

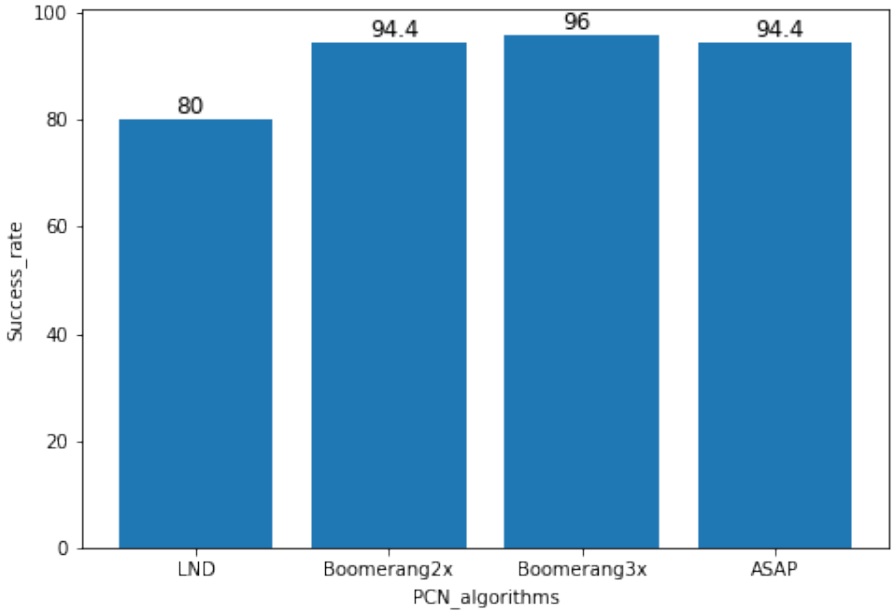
- The simulation is done in python as a toy implementation of above mentioned algorithms.
- Random graph is generated using networkx module in which each graph node denotes a node in payment channel network and edges denotes payment channel between two nodes.
- The cost of payment transfer through a channel is a function of variables like lock-time,latency,transaction fee etc. ; but for simplicity of simulation in our model ,we have considered the cost of a payment channel to be integer.
- Each channel is given a cost based on a randomly chosen a integer from a uniform distribution of range(0,10).
- We assumed that initially fund in each channel randomly chosen integer from a uniform distribution of range(10,20).

We compared the congestion level in network and success ratio of the algorithms result after simulating 1000 payment transactions. In the worst condition all the 1000 transaction may take place simultaneously.The congestion level denotes the number of times apayment channel is used in average during the 1000 transactions.

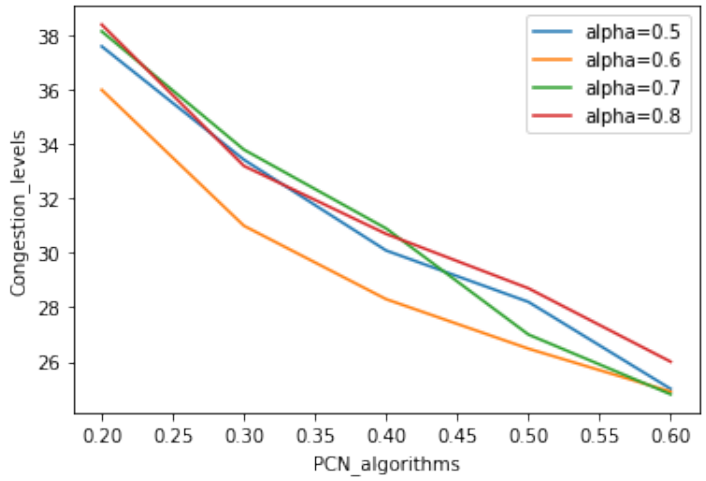
Note: Here Boomerang2K means twice the payment amount is send as redundant and similarly Boomerang3K mean thrice of the original amount is send. λ is rate parameter in exponential distribution.



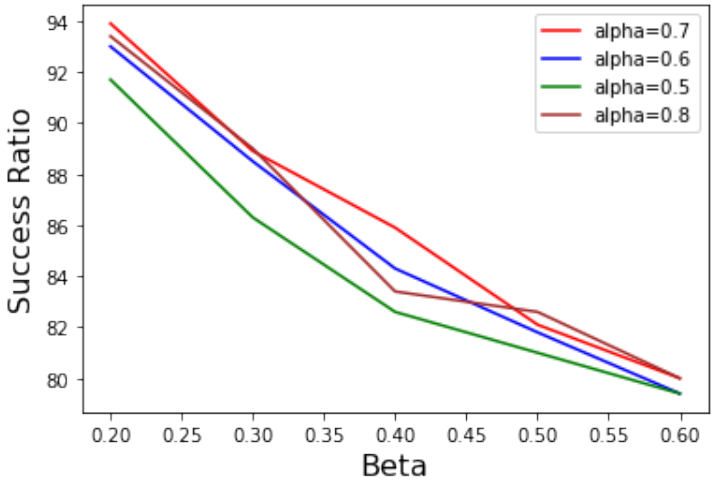
Congestion Level of different PCN algorithms



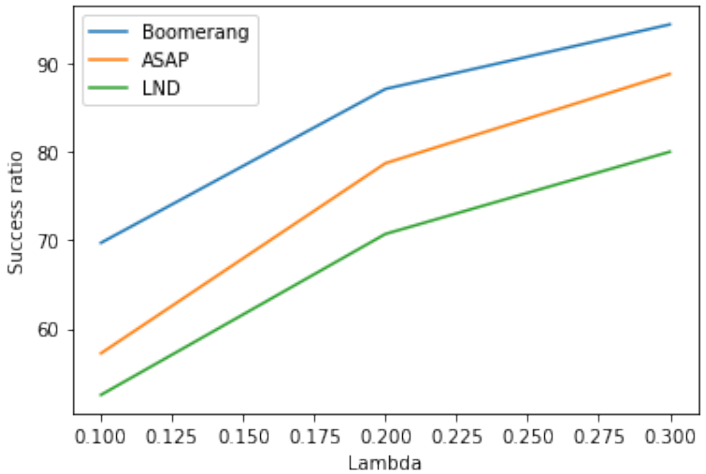
Success ratio of different PCN algorithms



Congestion Level of ASAP on different values of α and β



Success ratio of ASAP on different values of α and β



Success ratio of different algorithms on different values of λ

The different level of congestion and success ratio in ASAP PCN algorithm while tuning the hyper parameter α and β

For the value of $\alpha = 0.5$					
β	0.2	0.3	0.4	0.5	0.6
Congestion Level	35.39	30.81	28.11	26.13	23.81
Success Ratio	93.9	88.9	85.9	82.1	80.0

For the value of $\alpha = 0.6$					
β	0.2	0.3	0.4	0.5	0.6
Congestion Level	36.0	31.0	28.3	26.48	24.9
Success Ratio	93.0	88.5	84.3	81.8	79.4

For the value of $\alpha = 0.7$					
β	0.2	0.3	0.4	0.5	0.6
Congestion Level	38.15	33.8	30.9	27.0	24.8
Success Ratio	93.9	88.9	85.9	82.1	80.0

For the value of $\alpha = 0.8$					
β	0.2	0.3	0.4	0.5	0.6
Congestion Level	38.4	33.2	30.7	28.7	26.0]
Success Ratio	93.9	88.9	85.9	82.1	80.0

In our current controlled environment setup, ASAP outperforms the LND algorithm in terms of success ratio and congestion level being larger by a small fraction. Though Boomerang2K has a higher success ratio than ASAP, its congestion level is extremely high. Overall we can say that ASAP is able to match the success ratio of a boomerang with congestion levels lowered exponentially.

Chapter 7

Conclusion and Future Work

Multi path payments serves to send the total amount in small payments where a path with insufficient balance leads to the failure of entire payment. Boomerang [1] was introduced to improve the performance of multi path payment by adding redundant payments. The high computation requirements and congestion due to redundant payments were the key drawback of Boomerang. We introduced ASAP, a new PCN protocol which adaptively split the payment only when required. Three splitting criteria were proposed to perform the splitting task when the channel lacks enough balance to transfer amount. Experiments were performed to compare the performance of LND, Boomerang and ASAP on various parameters. Congestion level of the three protocols were compared by simulating 1000 payments. ASAP outperforms the LND and Boomerang3K protocols in terms of success ratio, congestion level. ASAP takes ten times less congestion level than Boomerang. ASAP success ratio is nearly same as Boomerang and way better than LND protocol. Our protocol matches the success ratio of Boomerang without adding congestion in the network in form of redundancy. Transaction fee was ignored in our protocol during the analysis. For future work, we will consider the transaction fee in the analysis of our protocol and extend our protocol by adding adaptive payment of the transaction fee.

References

[1] 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.

[2] Sonbol Rahimpour and Majid Khabbazian. Spear: fast multi-path payment with redundancy. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*, pages 183–191, 2021.

[3] Elias Rohrer and Florian Tschorsch. Counting down thunder: Timing attacks on privacy in payment channel networks. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 214–227, 2020.

7.1 Code and Presentation's Recording links

Project Presentation Video:

`https://drive.google.com/file/d/1E5B6sM2BcbEkAdDJ_vNeplW8IglK1eJo/view?usp=sharing`

Code on Github:

`https://github.com/Hrishi0000/Blockchain_projects.git`

Code Demo:

`https://drive.google.com/file/d/1D1uHBgkTtSedelHiY2NNqf6io9HHpke1/view?usp=sharing`