Tunable Survivability on Off-chain network

Hagay Michaeli and Uri Gadot

Intro

As block-chain usage increased in the last decade, its scalability restrictions are more relevant then ever. The fact that each transaction needs to be written inside a block, which is added to the chain on average time of 10 minutes, does not allow Block-chain based currency users to conduct quick transactions. Off-chain was introduced lately in order to allow block-chain users, to conduct those transactions with low latency waiting time. The basic concept is creating a paying channel between 2 users, which is backed up with some currency on the original block-chain network. These deposits dictate the channel capacity, thus guarantee that no user would have unsettled debts. The created channels can be used as an external network: Users can transfer money on a path of several different existing channels, thus creating a new network, an "Off-chain" network.
Since creating a new channel requires an "On-Chain" transaction, it is expensive and has low-latency. Therefore, we rather create an "Off-chain" topology that requires as least "On-chain" deposits as possible, while allowing as many "Off-chain" transactions as possible. In particular, we are interested in generating a topology that suffices the transaction requirements in a certain probability, which will be denoted as the network survivability.
Our work focuses on those two restrictions in-order to give a good cost-effective solution.
Previous work was done in finding optimal off-chain in terms of minimal deposits and transaction fees.[1] The idea of graph survivability was initially presented as a constraint for routing trees.[2]

Assumptions and Modelling

In order to model the problem easily, we would like to make several assumption:

1. Once a route path was chosen over the "Off-chain" network, all transaction between the users are done on the same path. if there are no sufficient deposit left, the transaction fails.

1. Khamis and Rottenstreich 2021.
2. Yallouz, Rottenstreich, and Orda 2016.

2. We are considering 0 fees with regards to the routing path chosen, thus making the deposit amount the only cost metric involved in our algorithm.
3. We are assuming that the transaction demands are periodical and occur randomly with a given distribution over a specific time interval.

In addition, lets define the following:
The number of users on the "Off-chain" network would be denoted as N.
Off-chain payments demands between users v and u would be denoted as $d(v,u)$.
The amount of payment that user v wants to transfer to user u on a given time period has Gaussian distribution with mean μ and variance σ^2. (e.g. if between users v_1 and v_2 the distribution is: $P_{v_1,v_2} \sim (0,0)$ then the users are not transferring currency, while on the other hand if the distribution is: $P_{v_1,v_2} \sim (1,0)$, then user v_1 is always transferring 1 currency to use v_2).
Lets denote an anti-symmetric matrix M that represent the payments mean values between each 2 users of the "Off-chain".
We would also denote symmetric matrix S that represent the payments variance values between each 2 users of the "Off-chain".
Because each user can never pay for himself we get: $\forall i < N, M[i,i] = S[i,i] = 0$.

Using Gaussian distribution as payments demand over time can be beneficial for several reasons:
1. Gaussian distribution over time can be easily modelled with given data
2. Calculating the probability of a given payment to be above / under a certain value is done more easily with Gaussian distribution
3. Summing several independent Gaussian variables is also easy to calculate.

* Example of the intuition behind this usage appear on Appendix 1.

Given an "Off-chain" channel between users v and u, we can denote this channel: $c(u,v)$. We can also denote the amount of deposit for each user as C_v and C_u.
Finally, lets denote the matrix C as the deposit for each user on all of it's "Off-chain" channels.

As we mentioned before, the demand between user v and user u can have several different routes on the "Off-chain" network. Let's denote a "function" $H : D \to C$
For a given demand $d(u,v)$ the value of the function over this demand, is the channels its uses on it's route over the "Off-chain" network.

$$H(d) = \{c \in C | d \text{ uses payment channel } c\}$$

In addition we can denote $H^{-1} : C \to D$ where for a given payment channel c the function maps all the demands using this channel.

$$H^{-1}(c) = \{d \in D | d \text{ uses payment channel } c\}.$$

We defined the survivability of a channel as the probability of the accumulative demand over this channel to be lower than its deposit:

$$S(c_{u,v}) = P_{success}(c_{u,v}) = P(-C_{u,v} \leq H^{-1}(c_{u,v}) \leq C_{v,u}) =$$
$$\text{P(-}C_{u,v} \leq \Sigma_{d \in H^{-1}(c_{u,v})} d \leq C_{v,u}) =$$
$$\text{P}(\Sigma_{d \in H^{-1}(c_{u,v})} d \leq C_{v,u}) - P(\Sigma_{d \in H^{-1}(c_{u,v})} d \leq -C_{u,v})$$

Because all of the demands are Gaussian variables then:

$$\Sigma_{d_{x,y} \in H^{-1}(c_{u,v})} d_{x,y} \sim N(\Sigma_{d_{x,y} \in H^{-1}(c_{u,v})} M[x,y], \Sigma_{d_{x,y} \in H^{-1}(c_{u,v})} S[x,y])$$

Now we can continue and define the survivability of the entire Graph as the average of survivability of every channel on the graph:

$$S(G) = \frac{\Sigma_{c \in G} S(c)}{\Sigma_{c \in G}}$$

* Adding different weights to every channel can allow us to define importance of one channel against another thus creating weighted average:

$$S(G) = \frac{\Sigma_{c \in G} w(c) \times S(c)}{\Sigma_{c \in G} w(c)}$$

For simplicity we would assume that the weights of each channel are equal: $w(c) = \frac{1}{N}$

Because the survivability of each channel is a Probability the following always holds: $0 \leq S_0 \leq 1$
This allow us to **tune** the survivability level of the transactions over the graph, for example: lets set the desired survivability to $S_0 = 0.9$ thus by definition we want the following to hold:

$$S(G) \geq S_0$$
$$\frac{\Sigma_{c \in G} w(c) \times S(c)}{\Sigma_{c \in G} w(c)} \geq S_0$$

Again, for simplicity we can demand a more strict condition, but easier to enforce:

$$\forall c \in G : S(c) \geq S_0$$
$$\implies S(G) = \frac{\Sigma_{c \in G} w(c) \times S(c)}{\Sigma_{c \in G} w(c)} \geq \frac{\Sigma_{c \in G} w(c) \times S_0}{\Sigma_{c \in G} w(c)} \geq S_0$$

Using this assumption we can define an algorithm to assure a certain amount of survivability while trying to minimize the entire capacity of the graph:

$$\min(C(G'))$$
$$\text{s.t. } S(G') \geq S_0$$

where:
$G' = (V, E)$:
V = all Off-chain users.
$E = \{c | c \in G'\}$
$C(G) = \Sigma_{c \in G} C$

Algorithm 1: Create Off-Chain topology and required deposits

Input:
$D = Sorted\ Demands\ list$;
$S_0 = Survivability$;
Output: $G = Off - ChainNetwork$
$G = (V, \emptyset)$;
for $d = (s, t, \mu, \sigma)$ *in* D **do**
  Find all paths from s to t;
  for *path from s to t* **do**
    calculate required additional deposit to maintain S_0;
  end
  calculate required deposit on direct path;
  choose minimal path;
  update G deposits;
end

Algorithm 2: Calculate additional deposit to maintain S_0

Input:
$P = Path\ from\ s\ to\ i$;
d = new demand to be added;
$S_0 = Survivability$;
Output: $A = additional\ deposit\ for : \forall c\ in\ P$
for $c\ in\ P, c \in G$ **do**
  save previous capacity on this channel: $old\_C = C$
  add d to $H^{-1}(c)$ and calculate optimization problem** -
  $new\_C = optimize(c, S_0)$
  Calculate additional deposits in order to hold the desired survivability -
  $A = max(0, 2 \times (new\_C - old\_C))$*
  remove d from $H^{-1}(c)$ and return A.
end

* A is defined as $max(0, addition)$ in order to avoid minimum additions, for more tighter boundary over the total capacity we can add negative additions to the algorithm.
** The optimization problem is defined as follow:

Optimization Problem: We would like to minimize the deposits (capacity of the edges), subject to survivability constraint:

$$\begin{aligned} &min\ \ x \\ s.t\ \ &\Phi(\tfrac{x-\mu}{\sigma}) - \Phi(\tfrac{-x-\mu}{\sigma}) \geq S_0 \end{aligned}$$

where:

x is the deposits on the edges - **for simplicity we assume same-size deposit on both edges.**
The total demand over the given channel $(c_{u,v})$ is:

$$\mathfrak{D}_{u,v} \sim N(\mu, \sigma)$$

$\mathfrak{D}_{u,v} = \Sigma_{d_{x,y} \in H^{-1}(c_{u,v})} d_{x,y}$
$\mu = \Sigma_{d_{x,y} \in H^{-1}(c_{u,v})} M[x, y]$
$\sigma = \Sigma_{d_{x,y} \in H^{-1}(c_{u,v})} S[x, y]$

This is a convex problem, so the solution for the deposits should satisfy

$$\Phi(\tfrac{x-\mu}{\sigma}) - \Phi(\tfrac{-x-\mu}{\sigma}) = S_0$$

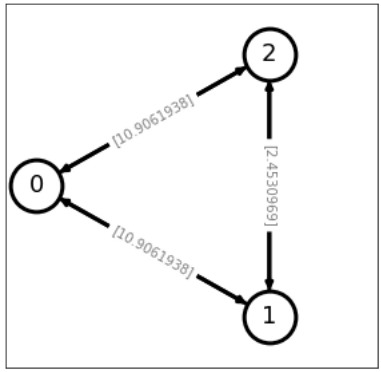
We used numerical algorithm to solve this equation, as there is no closed form solution to this equation.
by defining: $F(x) = \Phi(\frac{x-\mu}{\sigma}) - \Phi(\frac{-x-\mu}{\sigma}) - S_0$
and finding the roots x_i that holds: $F(x_i) = 0$, we can get the minimum needed deposit in order to hold the survivability conditions.

Example: Assume 3 clients with demands:

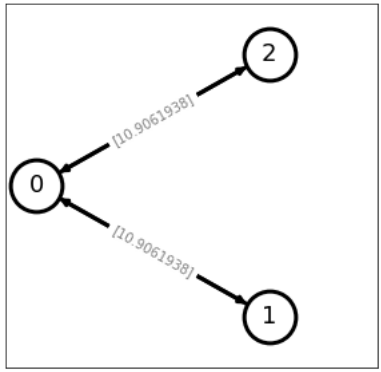
Source	Target	μ	σ
0	1	10	1
0	2	-10	1
2	1	-2	0.5

the required survivability is $S_0 = 0.9$
That means that user 0 is paying user 1 - 10 currencies on average on a given time.
While user 0 is being given 10 currencies from user 2 on average on a given time.

In order to satisfy the requirements with 3 direct channels, we would need the following topology, with total deposit of ~ 49.
(Each edge represents an open channel, and its weight is the required deposit of each of the nodes.



We can see that there is significant difference in the demands between 2 to 0 and 0 to 1. Therefore it is better to satisfy the demand from 1 to 2 through this path, instead of opening a new channel. By using the above method, we get that the requirements are sufficed by only 2 direct channels, in the following topology, and with total deposit of ~ 44/



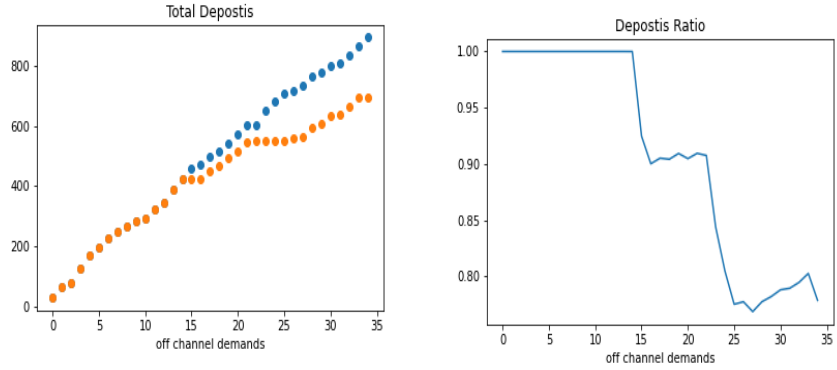
Simulation Results

We ran a simple simulation on our POC algorithm with the following configuration:

1. 10 different users ($N = 10$).
2. Changing amount of off-chain payment demands ($d \in [1, 35]$).
3. We compared our algorithm results to a naive deposit per-demand approach, where we are establishing a payment channel for each demand between 2 users.

In order to reach more conclusive results, we need to run tests that need more effort in terms of compute and time.

The results were encouraging as when the number of total demands increased the saving of our algorithm also increased. as seen in figure 1



(a) *Total deposits, our algorithm VS. naive approach*

(b) *Deposits savings*

FIGURE 1. *Experiments results*

in figure 1a We noticed that at the first iterations the total deposits are equal, while at a certain point, when there are enough channels that can be shared, the deposits in our algorithm grows slower.

Further Research

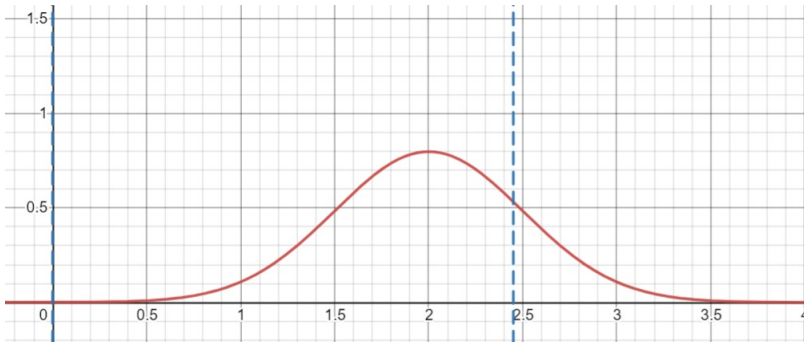
There are several area where we think there is a possibility to further investigate this subject.

1. Removing some of the simplicity assumptions in order to find better generic algorithm such as:
   - The addition A in each iteration of algorithm[2] must be a non-negative.
   - The weights of the demands are equal: $w(c) = \frac{1}{N}$.
   - The capacities on each channel are identical for both sides.
   - Forcing each channel survivability to hold: $S(c) >= S_0$.
2. Collect data from real Transactions, in order to analyze the transactions distribution and model it accordingly to the algorithm (independent Gaussian variable between each 2 users).
3. Exploring other ways to model the payment demand between 2 users.

Appendix

Understanding the intuition behind the usage of Gaussian distribution:
Lets denote a channel c between the users u and v on which there is only one demand: $d_{u,v} \sim \mathcal{N}(2, 0.5)$.
therefore, in order to hold a survivability of $S(c) = 0.9$ we need to add a capacity on the channel between u and v to hold the payments:
$C_u = 2.45$ and $C_v = 0$ and the distribution is displayed in figure 2.

FIGURE 2. *Gaussian distribution and the needed deposits on channel* $c_{u,v}$



Total capacity needed on this channel to satisfy the survivability condition is:

$$C_{total} = C_u + C_v = 2.45$$

but, if there is another demand between users x and y where: $d_{x,y} \sim \mathcal{N}(2, 0.3)$ and we choose to route this demand over channel $c_{u,v}$ from the direction of v to u. then the new distribution over the channel from direction u to v is:

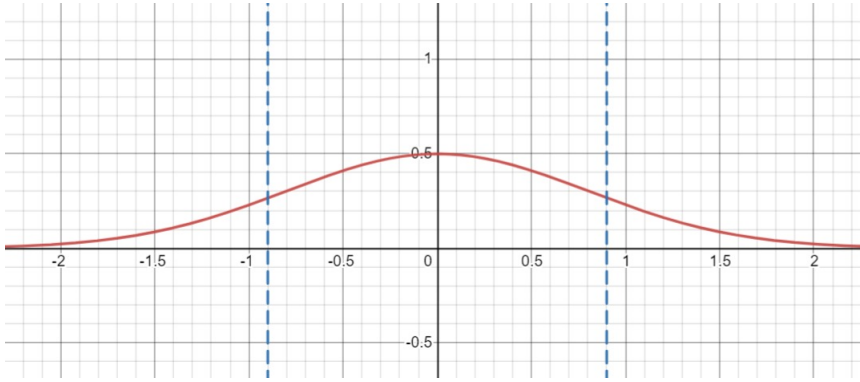
$$\mathfrak{D}_{u,v} \sim N(2 - 2, 0.5 + 0.3) = N(0, 0.8)$$

Therefore the new capacities needed on this channel are: $C_u = 0.9$ and $C_v = 0.9$ and the distribution is displayed in figure 3.
Total capacity needed on this channel to satisfy the survivability condition is:

$$C_{total} = C_u + C_v = 1.8 < 2.45$$

In conclusion, sometimes, its better to equalize the mean value of the money transferred on the channel in order to lower the needed capacity to hold the Survivability conditions.

FIGURE 3. *Gaussian distribution and the needed deposits on channel* $c_{u,v}$



Supplementary Material

Simple python implementation for the algorithm:
https://github.com/Ugadot/Tubable-Survivabilty-Off-chain

References

Khamis, J., and O. Rottenstreich. 2021. Demand-aware Channel Topologies for Off-chain Payments. Paper presented at the 2021 International Conference on COMmunication Systems NETworkS (COMSNETS), 272–280. https://doi.org/10.1109/COMSNETS51098.2021.9352899.

Yallouz, J., O. Rottenstreich, and A. Orda. 2016. Tunable Survivable Spanning Trees. *IEEE/ACM Transactions on Networking* 24 (3):1853–1866. https://doi.org/10.1109/TNET.2015.2438254.