# Subnetworks in BlockDAG

OLEKSANDR ANTONENKO, Odesa I. I. Mechnikov National University, Ukraine
SERGII GRYBNIAK, Odesa Polytechnic State University, Ukraine
DENIS GUZEY, Odesa I. I. Mechnikov National University, Ukraine
OLEKSANDR NASHYVAN, Odesa Polytechnic State University, Ukraine
RUSLAN SHANIN, Odesa I. I. Mechnikov National University, Ukraine

In the article, we study the horizontal scaling of the Waterfall or similar blockDAG networks by partitioning them into subnetworks by applying hierarchical and graph-based clustering algorithms. It leads to the reducing the network load and, in addition, to the increasing of the potential performance parameters of the underlying protocol. We consider methods of topology construction, propose clustering algorithms, and perform a simulation of a network partitioning into subnetworks.

CCS Concepts: • **Computing methodologies** → **Distributed algorithms**; *Modeling and simulation*; • **Information systems** → **Clustering and classification**; • **Networks** → *Peer-to-peer networks.*

Additional Key Words and Phrases: blockchain, blockDAG, subnetwork, agglomerative hierarchical clustering, divisive hierarchical clustering

**ACM Reference Format:**
Oleksandr Antonenko, Sergii Grybniak, Denis Guzey, Oleksandr Nashyvan, and Ruslan Shanin. 2022. Subnetworks in BlockDAG. 1, 1 (December 2022), 20 pages. https://doi.org/XXXXXXX.XXXXXXX

## 1 INTRODUCTION

In modern world, blockchain technologies and their generalizations have become widespread and play an important role in multiple industries. The use of blockchain to store and distribute information in decentralized distributed networks is limited by poor scalability and low transaction speed [2]. One solution is to use a directed acyclic graph (DAG) structure instead of a chain one. This leads to the emergence of blockDAG networks. The data structure in the form of a DAG allows the simultaneous creation of several blocks and has been studied in many works [5, 11, 16, 21, 22, 25]. Other approaches to solving these problems have also been proposed: sharding [4, 6, 12, 15, 26], Layers [1, 14, 20], etc.

In the paper, we consider the problem of scalability the blockDAG Waterfall network [7, 8]. We will focus on a concept of subnetworks which suggests that some information is not distributed throughout the network, but only among some set of nodes, which we will call a subnetwork. Since several blocks are created in blockDAG networks in one round of work and, thus, there is no need to send all transactions throughout the network, we can use subnetworks to separate the set of transactions, which does not make sense in conventional blockchain systems. Also, partitioning

Authors' addresses: Oleksandr Antonenko, Odesa I. I. Mechnikov National University, Odesa, Ukraine, antonenko@onu.edu.ua; Sergii Grybniak, Odesa Polytechnic State University, Odesa, Ukraine, s.s.grybniak@op.edu.ua; Denis Guzey, Odesa I. I. Mechnikov National University, Odesa, Ukraine, d1onat1d@gmail.com; Oleksandr Nashyvan, Odesa Polytechnic State University, Odesa, Ukraine, o.nashyvan@op.edu.ua; Ruslan Shanin, Odesa I. I. Mechnikov National University, Odesa, Ukraine, ruslanshanin@onu.edu.ua.

into subnetworks, we build methods which reduce the network latency within the subnetworks while keeping the subnetworks roughly the same size. This innovative approach allows us to reduce network load and gives us a partial solution of the problem of the simultaneous creation of blocks with some identical transactions. Note that the concept of subnetworks has been used in other way in some blockchain protocols. For example, in Ethereum 2.0 [23], Kaspa [24] and TON [4], subnetworks are used for division of nodes into different subgroups to perform additional tasks.

Thus, in the paper, we consider the division of the Waterfall network into subnetworks and appropriately dividing the transaction pool into disjoint subpools for reducing the network load.

## 2 SCIENTIFIC NOVELTY

In this paper, the following approaches are formulated for the first time:

(1) automatic (algorithmic) subnetting (partitioning into subnetworks) of the blockchain / blockDAG network into subnetworks in order to reduce network traffic using data on network delays between nodes (to reduce delays within subnetworks), and not in a pseudo-random way (thus, the principle of division into subnetworks differs significantly from the principles of division into subnetworks in previous works);
(2) introduction of the concept of virtual nodes as an element of the separation of the basic Waterfall protocols into the network level (at which there is data on the physical nodes of the blockchain / block DAG, their addresses and their network interaction) and the block DAG / shard level at which there are the concepts of validators, virtual nodes (as groups of validators), published blocks and links between them and delays in the distribution of blocks between validators.

Also in this paper, we study a very specific clustering problem (which was not considered earlier in such a combination of requirements):

(1) the clustering algorithm uses a graph structure of connections between clustered objects, and the graph is sparse;
(2) additional restrictions are imposed on the fact that the total weight of clusters (the total number of validators in it) should be approximately equal in all clusters;
(3) very fast operation of the algorithm: the algorithm should run in no more than $O(m \log m)$, where $m$ is the number of nodes.

Based on the above requirements, the paper presents new modifications of hierarchical division clustering algorithms, as well as a modified distance function for hierarchical agglomerative clustering algorithms. The operation of these algorithms has been tested on the interaction model of network nodes of the blockDAG.

## 3 PROBLEM STATEMENT

First, let us review the necessary points related to the operation of the Waterfall network. The time of work of the Waterfall network is divided into rounds, called slots, and epochs, which are used to summarize the work of the network. At the beginning of each epoch, a list of committees for the next epoch is determined, and at the beginning of each slot, the workers who will create a block in that slot are determined.

A BlockDAG network node contains a pool of transactions that are waiting to be added to a block or finalized and also a local copy of the BlockDAG ledger. On single blockDAG node, multiple validators can work at once, the main function of which is to sign the blocks generated by the node. Each validator is characterized by a pair of keys — private and public ones, with the public key acting as the validator's identifier.

Currently, in Waterfall, each transaction is distributed throughout the network, and thus potentially hits every node. When the queue for publishing a block by some validator comes, the transaction falls into a correctly created block, the blockDAG nodes propagate this block over the network. Having received a block with transactions, the node marks the transaction as added to the block so that it will not be republished. After some time, this block is finalized, and at the moment when the blockDAG node receives information about it, all transactions of this block are deleted from the node's transaction pool. Note that nodes may not contain validators, in which case they will never publish blocks, but will receive, store and distribute transactions and blocks. An example of such nodes would be "wallet nodes", which generate transactions (for example, to transfer money) and distribute them over the network, but never generate blocks themselves.

The basic idea of subnetting (partitioning into subnetworks) is that each node will belong to some single subnetwork and will only store and distribute transactions of that subnetwork. At the same time, it will receive, store and distribute all blocks of blockDAG and, thus, will have a complete copy of the entire blockDAG repository.

### 3.1 Requirements to partitioning into subnetworks

Let us begin our consideration of partitioning into subnetworks by discussing the challenges and problems that arise in the process. First, it is obvious that

- all nodes containing validators must belong to some subnetwork.

Also, since nodes from different subnetworks will have different sets of transactions and all correct transactions must be published to blockDAG, then

- each subnetwork must contain nodes that have validators on them.

Since the validator does not create the block often, and the transactions must be processed as quickly as possible, we have another requirement

- subnetwork validators must "frequently" create blocks.

This requirement can be interpreted in a number of ways. First, it can be understood in the probabilistic sense that, on average, at least one block per slot is created in each subnetwork. Second, it can be understood in the deterministic sense that in each subnetwork it is created at least one block per slot.

In general, three different approaches can be proposed for splitting blockchain/blockDAG nodes into non-overlapping subnetworks:

(1) in the first approach, each node determines which subnetwork it belongs to;
(2) in the second approach, pseudo-random partitioning occurs;
(3) in the third approach, the partitioning takes into account the characteristics of the connection between the nodes, to minimize the delay within the subnetworks.

The first approach is more appropriate for systems based on Proof of Work consensus rather than Proof of Stake consensus. The second approach is tempting for its simplicity of implementation. In this paper, we propose the use of a third approach, for which it is necessary to develop:

(1) an information collection method;
(2) an algorithm of clustering (division) the network to subnetworks based on collected data.

Partitioning the network into subnetworks requires a method for obtaining information about the links in the network. Immediately note that, since each subnetwork must have validators, the information required for clustering

must contain, in particular, information about the presence and number of validators in the network nodes, which, generally speaking, can generate new types of threats and attacks. To prevent this, we introduce the notion of "virtual node", and we will work with virtual nodes when partitioning the network into subnetworks.

Finally, another important requirement for a clustering method is speed. Unfortunately, most methods of clustering data are quite slow, and when handling large volumes of data, they require greater computing power. Thus, we focus on algorithms that work at speed $O(m \log m)$, where $m$ is the number of virtual nodes.

### 3.2 Subnetting approaches

We will refer to a virtual node as a set of validators that we assume share a common transaction pool and are located on the same physical node. By default, each validator is considered as a virtual node. To combine validators into a virtual node or to change the composition of a virtual node, validators must publish the appropriate transactions to the network. This procedure can be implemented in different ways, so we will not dwell on it, we will only point out that in order to join a virtual node, both the desire of the attached validator and confirmation from existing validators are needed.

Note that it is impossible to check the correspondence of one blockDAG node to one virtual node, since there is no information about physical nodes in the blockDAG. If the validators of one physical node end up in different virtual ones, then this can lead to the fact that they end up in different subnets and the blockDAG node will not be able to function in normal mode. On the other hand, if the validators of several blockDAG nodes are combined into one virtual node, then their functioning will not be disrupted (this situation should be considered as normal).

Prior to the introduction of subnets in Waterfall, block creators in a particular slot were pseudo-randomly selected from all validators currently running in the blockDAG using a "swap-or-not shuffle" algorithm, similar to the Etherium 2.0 network [9].

Let us consider several approaches to subnetting with all of the above in mind:

(1) The block creator selection system does not change.
   (a) The network is divided into subnetworks with the sole restriction that subsets cannot be too small. The advantage of this solution is the simplicity and the undemanding nature of the clustering algorithm. The disadvantage of this solution is the fact that, on smaller subnetworks, blocks will be generated much less frequently, and the frequency of block generation is not guaranteed. Thus, transactions could potentially hang for long periods of time.
   (b) The network is divided into subnetworks which contain approximately equal numbers of validators. In this case, the number of blocks created in the slot should not be less than the number of subnetworks. Assuming an even distribution of block creators, on average at least one block per slot it will be created in each subnetwork.

(2) The block creator selection system is tied to subnetworks, namely, block creators in each slot are selected from members of the subnetwork:
   (a) The network is divided into subnetworks that contain approximately equal numbers of validators. In this case, the number of blocks, created in a slot, is a multiple of the number of subnetworks. This method works most predictably, but requires special properties from the clustering algorithm.
   (b) The network is divided into subnetworks, and the number of validators in each subnetwork may differ significantly (for example, by no more than two or three times). In contrast to the previous point, in order

to maintain approximately equal probability of block creation by all validators, additional mechanisms must be introduced to ensure that the probability of block creation by participants in different subnetworks is the same.

Note that in case 2) a necessary requirement is that all validators know the composition of subnetworks (the distribution of validators across subnetworks), which limits the choice of clustering methods. In this paper, we study the second approach when partitioning into subnetworks. Also, since clustering must be identical at every node, we suppose that all nodes will execute the clustering algorithm based on the finalized part of the ledger before some fixed slot.

## 4 BUILDING A NETWORK TOPOLOGY

### 4.1 Notations and preliminary remarks

Further we assume that the set $\mathcal{V}$ of all validators of the network is given, $|\mathcal{V}| = n$ is the number of validators, and some class $\mathcal{N}$ of subsets of $\mathcal{V}$ is given, $\mathcal{N} \subseteq 2^{\mathcal{V}}$, whose elements are virtual nodes, $|\mathcal{N}| = m$ is the number of virtual nodes. We assume that any two elements of the class $\mathcal{N}$ either coincide or do not intersect, and each of the validators belongs to some virtual node. We additionally impose a restriction that a virtual node cannot contain many validators, namely, there is such natural number $L$, that for any virtual node $N \in \mathcal{N}$, the inequality $|N| \leqslant L$ holds, where, by $|N|$, we denote the number of validators in the virtual node $N$. This condition is important, because otherwise there could be substantial problems with the uniform distribution of validators across subnetworks.

Let us formulate our problem in the described terms: it needs to partition the set of nodes $\mathcal{N}$ on $k$ subsets (clusters, subnetworks) $Cl_1, Cl_2, \ldots, Cl_k$ so that $Cl_1 \cup Cl_2 \cup \ldots \cup Cl_k = \mathcal{N}$ and $Cl_i \cap Cl_j = \varnothing$ if $i \neq j$, which means that each virtual node is included in exactly one cluster. In what follows we will denote the set of clusters as $\mathcal{C}$, $\mathcal{C} = \{Cl_1, \ldots, Cl_k\}$.

### 4.2 Restrictions on clustering options

(1) The number of clusters is much less than the number of nodes, i.e.

$$k \ll m.$$

Moreover, we can assume that $k = O(1)$ or $k = O(\log m)$.

(2) The maximum number of validators per node is much less than the weight of the cluster

$$\max |N| \leq L \ll \frac{n}{k}.$$

(3) The maximum number of validators that belong to one user or one coordinated group of nodes is much less than the weight of the cluster (no cluster collusion):

$$\sum_{\text{by some group}} |N| \ll \frac{n}{k}.$$

### 4.3 Information collection methods

In what follows by era we mean some fixed period of time such that during this period of time all network participants will create at least one block. Information about the connections between virtual nodes is collected and published within the era. Let us describe this procedure.

Let $N_1$ and $N_2$ be two arbitrary nodes, and let validator $v \in N_1$ fix $q$ times the distance between the virtual nodes $N_1$ and $N_2$. When one of validator of node $N_2$ publishes a block in blockDAG, some time later this block gets to validator $v$. At that moment, it fixes the delay $d_{q+1}$ according to the slot beginning and calculates a new distance between virtual nodes $N_1$ and $N_2$ by the following formula:

$$d_{v,q+1}(N_1, N_2) = \begin{cases} \frac{q}{q+1} d_{v,q}(N_1, N_2) + \frac{1}{q+1} d_{q+1} & \text{if } q \leqslant Q, \\ \frac{Q}{Q+1} d_{v,q}(N_1, N_2) + \frac{1}{Q+1} d_{q+1} & \text{if } q > Q. \end{cases}$$

Also we set $d_{v,0}(N_1, N_2) = d_0$. Hence, all validators of the node $N_1$ (or the physical nodes on which they reside) store information about the average delay for the blocks of each other virtual node (except for those that have not yet published blocks, or have not published blocks for a very long time, i.e. have been offline for a long time).

The procedure for collecting information described above can be supplemented depending on the clustering algorithm. Also, the procedure can be supplemented by a decrease in the indicators of not validly working or unstable virtual nodes.

### 4.4 Requirements for published information

Each node contains information about the distances from it to all other virtual nodes. But we cannot do clustering by this information, since all nodes must store consistent information about links. There are several options for solving this problem:

(1) The nodes agree on the accumulated information using some BFT consensus and publish or distribute the agreed results.
(2) Each node publishes its information to the ledger at the moment when it comes time to generate a block.

In this paper, we consider only the second approach.

Let us formulate the basic requirements for storing and publishing information about connections:

- Each virtual node has some communication characteristic with the majority of other virtual nodes, we will consider this as the average block delay.
- All nodes publish no more than $O(m \log m)$ blocks of information to BlockDAG.
- According to the information published in BlokDAG, the clustering process is unambiguously carried out by all nodes.

### 4.5 Two models of published information

(1) In the first approach, each node publishes some number of connections to other nodes:
    (a) some fixed number of closest nodes;
    (b) some fixed number of closest nodes and some number of random connections to other nodes.
(2) In the second approach, clustering is carried out in two steps:
    (a) at the first step, some representatives of clusters are selected, which we will call organizers or centers of clusters, i.e. $k$ nodes, one per cluster;
    (b) in the second step, these cluster organizers publish the distances to all other nodes (at least those that they have).

With both models, we get the amount of published data no more than $O(m \log m)$.

In this paper, we will consider in detail only approach 1(a), namely that each node publishes some fixed number of closest nodes. At the moment when it is the turn of the validator $v$ of node $N_1$ to publish its block, it selects from all virtual nodes (except its own) $s$ nodes with the least delays $d_v(N_1, N_2)$, and publishes the list of $s$ identifiers of virtual nodes and delays to them.

### 4.6 Topology of network of virtual nodes

All clustering algorithms are divided into algorithms that work with a metric representation of objects (each object is represented by a vector in a metric space) or a relational representation of objects. With the chosen model of published information, we get a relational representation of objects, which is a sparse matrix of distances between objects or a weighted graph. Let us describe the construction of this matrix or graph in more detail. At the same time, from the initially non-symmetric distance matrix, we will strive to obtain a symmetric matrix for the convenience of the algorithms.

For any two virtual nodes $N_1, N_2 \in \mathcal{N}$ we denote by $V(N_1, N_2) = \{v \in N_1 \cup N_2 \colon \exists d_v(N_1, N_2)\}$ the set (possibly empty) of all validators, that have published distances between nodes $N_1$ and $N_2$, where we denote by $d_v(N_1, N_2)$ the published validator $v$ distance between nodes $N_1$ and $N_2$, $d_v(N_1, N_2) = d_v(N_2, N_1)$, and write

$$D(v) = \{d \colon \exists N_1, N_2 \in \mathcal{N} \quad d_v(N_1, N_2) = d\}$$

for the set of all distances published by the validator $v$. Thus, if some validator $v \in N_1 \cup N_2$ publishes a distance between nodes $N_1$ and $N_2$, then it is counted as the distance from $N_1$ to $N_2$ as well as the distance from $N_2$ to $N_1$.

If $V(N_1, N_2) \neq \varnothing$, then we define $d_1(N_1, N_2)$ as the average distance between nodes $N_1$ and $N_2$ by the following rule

$$d_1(N_1, N_2) = \frac{1}{|V(N_1, N_2)|} \sum_{v \in V(N_1, N_2)} d_v(N_1, N_2),$$

where the sum is taken over all published distances between nodes $N_1$ and $N_2$.

The problem of the $d_1$ metric is the discrepancy between the distances between the nodes $N_1$ and $N_2$. For example, if the distances measured by validators $N_1$ are small, are included in the list of shortest distances and are published, and the distances measured by validators $N_2$ are significantly larger, are not included in the list of shortest distances and are not published, then in $d_1(N_1, N_2)$ only short distances measured by $N_1$ validators are taken into account and distances measured by $N_2$ validators are not taken into account.

To partially correct this problem, for virtual nodes $N_1$ and $N_2$ such that $V(N_1, N_2) \neq \varnothing$ we also define the distance $d_{max}$ as follows

$$d_{max}(N_1, N_2) = \frac{1}{|N_1| + |N_2|} \sum_{v \in N_1 \cup N_2} d_v^{max}(N_1, N_2),$$

where $d_v^{max}(N_1, N_2) = d_v(N_1, N_2)$ if $v \in V(N_1, N_2)$ and $d_v^{max}(N_1, N_2) = \max\{\max D(v), d_1(N_1, N_2)\}$ if $v$ does not belong to $V(N_1, N_2)$. This metric is designed to heuristically take into account the distances missing in the published data, only if they can be large, based on other $N_2$ data published by the node.

In what follows, we denote by $d(N_1, N_2)$ one of the values $d_1(N_1, N_2)$ or $d_{max}(N_1, N_2)$. We will assume that the values $d_1(N_1, N_2)$ and $d_{max}(N_1, N_2)$ are undefined if $V(N_1, N_2) = \varnothing$, that is, if no validator has published distances between nodes $N_1$ and $N_2$. Thus, we can say that the published distances $d(N_1, N_2)$ define an undirected graph $G = (\mathcal{N}, E)$, where the set of edges coincides with those pairs of nodes between which the distance is published, i.e. $E = \{(N_1, N_2) \in \mathcal{N}^2 \colon V(N_1, N_2) \neq \varnothing\}$, weights of nodes are equal to the number of validators in them $w(N) = |N|$,

and weights of edges are equal to $d(N_1, N_2)$. Note that we suppose that the graph $G$ is connected which is confirmed by numerical experiments at $s \geqslant 10$.

Note that a situation is theoretically possible when a graph contains a small number of isolated vertices, i.e. nodes that have not published link data for any reason and for which other nodes have not published link data. Such nodes can interfere with the clustering algorithm, so they need to be treated specially. We will temporarily exclude such nodes from the operation of clustering algorithms and cluster the remaining nodes. Then we will add these isolated nodes in turn to the cluster with the smallest number of validators, continuously recalculating the number of validators in clusters (because after adding a node to a certain cluster, this cluster may cease to be minimal in terms of the number of validators, and the next isolated node will be added to another cluster).

### 4.7 Clustering into roughly equal clusters

As mentioned earlier, we are trying to achieve an approximate equality in the number of validators in all clusters. Of course, exact equality cannot be achieved because the total number of validators $|\mathcal{V}| = n$ may not be a multiple of $k$, and we divide into clusters not validators, but virtual nodes, which may contain up to $|N| \leqslant L$ validators.

Therefore, we formalize the notion of approximate equality as follows: let us call the number of validators in a cluster the cluster weight and denote it by

$$w(Cl) = \sum_{N \in Cl} |N| = \sum_{N \in Cl} w(N).$$

We will say that the clusters $Cl_1, Cl_2, \ldots, Cl_k$ roughly equal in number of validators if $w(Cl_j) \leqslant w(Cl_i) + L$ for every $i, j \in \{1, ..., k\}$, $i \neq j$. It is easy to see that such partitioning into clusters always exists. Indeed, we can build such a partition iteratively, starting with $k$ empty clusters $Cl_1, Cl_2, \ldots, Cl_k$ (the approximate equality condition is still satisfied) and adding nodes in turn to the cluster, which currently has minimum weight. Moreover, after adding the node with the maximum $L$ validators to the smallest cluster, its weight cannot exceed the weight of the largest by more than $L$. Thus, we know that clustering with an approximate equality in the number of validators always exists. Of course, this clustering may not be optimal in terms of various parameters, such as the average distance in the cluster, so in practice we can also be satisfied with the option when the largest and smallest cluster weights differ by no more than two times.

### 4.8 Balancing block generation

Consider the case when the number of validators in subnetworks can differ by a maximum of 2-3 times. In this case, we can compensate for this discrepancy in the following way: in the smallest subnetwork, blocks are always generated once per slot, but in subnetworks with a large number of validators, 2 or even 3 blocks will sometimes be generated in one slot.

Let in the division of the entire network of nodes, we obtain $k$ subnetworks (clusters) of different sizes, namely, the sizes $w(Cl_1) \leqslant \ldots \leqslant w(Cl_k)$. Let an era consist of $R$ slots. Then the smallest subnetwork generates one block per slot, and then all the others in proportion to their size, i.e. $i$-th network will generate $\left\lfloor \frac{w(Cl_i)}{w(Cl_1)} R \right\rfloor$ blocks per era.

Let $S$ blocks be generated by some subnetwork in an era. In the $j$-th slot it is generated $s_j = \left\lfloor \frac{S}{R} j \right\rfloor - \left\lfloor \frac{S}{R} (j-1) \right\rfloor$ blocks. If more than one block needs to be generated in the $i$-th slot, the entire transaction pool is divided into “baskets”, depending on the remainder of the transaction hash divided by $s_j$.

## 5 HIERARCHICAL CLUSTERING

When selecting clustering methods that meet the above requirements, we settled on variations of hierarchical clustering methods. Modifications of divisive and agglomerative hierarchical clustering algorithms were considered (for more details of these algorithms, see, for example, [10, 19]).

### 5.1 Agglomerative hierarchical clustering (AHC)

In this approach, we cannot guarantee that the obtained clusters will have approximately equal sizes. Nevertheless, with the specially introduced distance between clusters, we increase the probability that the weights of the maximum and minimum clusters will not differ by more than a factor of 2. Let us denote by $M = \frac{|\mathcal{V}|}{k}$ and let

$$wc(Cl) = \begin{cases} w(Cl) & \text{if } w(Cl) \leqslant M-1, \\ M-1+\frac{w(Cl)-M}{w(Cl)} & \text{if } w(Cl) > M-1. \end{cases}$$

Also we set $P(Cl_1, Cl_2) = \frac{M}{M-wc(Cl_1)} \frac{M}{M-wc(Cl_2)}$. Then we define the distances between clusters as follows

$$\text{dist}_{max}(Cl_1, Cl_2) = P(Cl_1, Cl_2) \max_{\substack{a\in Cl_1,\ b\in Cl_2 \\ V(a,b)\neq\varnothing}} d(a,b),$$

$$\text{dist}_{min}(Cl_1, Cl_2) = P(Cl_1, Cl_2) \min_{\substack{a\in Cl_1,\ b\in Cl_2 \\ V(a,b)\neq\varnothing}} d(a,b).$$

In some sense, we can think that $d_{max}(Cl_1, Cl_2)$ is a variation of complete linkage hierarchical clustering, and $d_{min}(Cl_1, Cl_2)$ is a variation of single linkage hierarchical clustering, with an additional factor penalizing "heavy" clusters.

### 5.2 Divisive hierarchical clustering (DHC)

Classic divisive hierarchical clustering algorithms [10] work as follows:

(1) Each time, some cluster is divided into two parts.
(2) In the divided cluster the most distant points in some sense should be found, which become the starting points of new clusters.
(3) Using these points, all other nodes in the cluster are added to some subcluster by certain rules.

The general idea of this process is quite simple: if we need to divide the set into $k$ clusters, then we will initially divide the entire set into two clusters, then one of them again into two clusters, etc., until we get $k$ clusters. At the same time, we need the resulting clusters to be approximately equal in the number of validators.

There are two ways to do this:

(1) Let $k = 2^j$ be a power of two. Then we can divide the whole set in half into two (approximately) equal parts in the number of validators, then each of the halves is again equally divided, and so on $j$ times.
(2) We divide the whole set into two clusters in the ratio (approximately) $k-1$ to 1, then we again divide the most part in the ratio $k-2$ to 1 and so on, until we will not get $k$ approximately equal clusters.

Now we propose the following approach for partitioning the network or some cluster into two parts in some proportion:

(1) We select two vertices that are the most distant from each other in a certain sense; we will call them the base vertices of the clusters.
(2) Let's divide the set of nodes into two parts in turn enlisting the nearest vertices to the base vertices in their clusters, observing two conditions each time:
    (a) the cluster size ratio is as close to the desired one as possible;
    (b) both clusters are connected graphs.

### 5.3 Selection of the most distant vertices

The main idea is that if we select vertices distant from each other as base ones, then we will get a good division of nodes into two clusters. For example, if one of the nodes is located in America, and another node is located in Europe (or, for example, in the Far East), then it is logical that the nodes that have a better connection with America, i.e. most likely geographically located close to the first node, will fall into one cluster, while European and Asian ones will fall into another cluster. Note that the requirement of equality of clusters or the need to separate validators in a certain proportion will degrade the quality of clustering, but this requirement is very important for us. For example, if it is necessary to divide the nodes in two clusters of equal weight, but 2/3 of the nodes are located in the USA and 1/3 in Europe, we will get either clusters that differ significantly in size, or, in the case of equal clusters, we will have one cluster of American nodes, and another mixed cluster consisted of American and European ones, so the average distance between nodes in it will be significantly greater.

A similar idea of bisectional clustering was first proposed by Lawrence Hubert in 1973 [10]. The paper [19] provides an overview of various hierarchical agglomerative and divisional clustering methods. The main difference between the methods proposed in this study and the above works is that for us the ratio of weights in clusters is significant, so we want to end up with clusters with (approximately) the same number of validators.

Suppose we have a graph $G$ whose vertices are virtual nodes and whose edge weights are the published distances between nodes, let's consider ways to find "distant" nodes in the graph.

In the simplest and most obvious way, we can require each node to publish the distances to what it thinks is the farthest node, and then in ($O(m)$ operations choose the pair with the largest distance from all given pairs. As tempting as it is, this approach has some very significant drawbacks:

(1) if the node $N_1$ has published the distance to the node $N_2$ as the largest of the metrics in its metrics table, then it is not certain that the node $N_2$ will publish the distance to the node $N_1$, therefore this distance cannot be cross-checked in this way;
(2) after dividing the cluster in half, it is quite logical to assume that for most vertices the most distant vertices from them will be in another cluster. But how then do we divide the already obtained clusters again in half?
(3) But the most important objection is the following: a large value of the distance between the nodes may not mean much in practice. For example, it may mean that one of the nodes is faulty (works intermittently or slowly), has problems with the Internet, or is simply dishonest. In this sense, small distances are much more indicative. Of course, dishonest validators can also indicate small distances, but if we require to get a long chain of small distances to get a large distance, then this partially protects against the arbitrariness of some false node.

The next way is to find the largest of the shortest distances in the graph. Moreover, the graph $G$ can be considered as an ordinary unlabeled graph (forgetting about the indicated distances) and the shortest distances in the graph can be obtained as the number of edges in the minimum path between the vertices, or considered as a weighted undirected

graph and the shortest distances in the graph can be obtained as the minimum sum of weights edges in some path between vertices. Thus, it is required to find the diameter of the graph. The problem with this approach is that finding the diameter of a graph is a very expensive operation that may require ($O(m^3)$ operations (in the case of the Floyd-Warshall algorithm) or ($O(m^2s\ logm)$ operations in case of running Dijkstra's algorithm from each vertex (weighted graph case) or ($O(m^2s)$ in case of running BFS from each vertex (weighted graph case).

One way to "speed up" this search is the following idea: let's first build a minimum spanning tree of $G$ in some way (using Prim's or Kruskal's algorithm for ($O(ms \log m)$ or slightly faster, depending on the implementation). We can then find the outermost vertices in the tree using the well-known tree diameter search algorithm (double tree traversal algorithm) in ($O(m)$ – as in the case of distance in a weighted tree as well as in an unweighted tree. The main idea of finding the most distant vertices in the tree is well known: namely, we take an arbitrary vertex of the tree, let's call it $N_0$, find the vertex in the tree that is the most distant from the vertex $N_0$, let's call it $N_1$, then in a similar way we will find the most the vertex $N_2$ remote from the vertex $N_1$. Let us call this the double traversal algorithm.

The most distant vertex from some vertex $N$ of the graph is found as follows: Start the traversal of the graph starting from vertex $N$, the most distant vertex is the last vertex in this traversal. If we use breadth-first search (BFS) as a traversal, the pair $N_1$, $N_2$ will be the furthest in terms of number of edges between them (i.e. the furthest in an unweighted graph), and if we use Dijkstra's traversal [3], then the pair $N_1$, $N_2$ will be the most distant in terms of the length of shortest path between them (as sum of weights of edges, in the weighted graph). In an arbitrary graph this double traversal algorithm will not find a pair of vertices with maximal distance. But it will still find a pair of sufficiently distant vertices, which can be taken as a base pair.

After conducting the first numerical experiments, we rejected the idea of building a minimum spanning tree and finding the farthest vertices in this tree, as this gave poor results. In this way, we focus on the following methods for finding two most distant vertices for this step:

(1) Find the most distant vertices in a weighted graph ($O(m^2s \log m)$ operations). We call this method global_dijkstra.
(2) Find the most distant vertices in an unweighted graph ($O(m^2s)$ operations). We call this method global_bfs.
(3) Find some distant vertices by double traverse in a weighted graph ($O(ms \log m)$ operations). We call this method double_dijkstra.
(4) Find some distant vertices by double traverse in an unweighted graph ($O(ms)$ operations). We call this method double_bfs.

### 5.4 Graph traversals

To select the most distant vertices and to divide the graph into two clusters, various classical graph traversals are used, we briefly list them. In all traversals, a certain structure is stored, to which the initial vertex in the traversal is initially added, the next vertex that is extracted from the structure is marked as traversed (and we do not return to it in the future). The graph can be effectively represented in software implementation as adjacency lists.

(1) Breadth First Search (BFS) [13] is used a queue for storing vertices, selected a vertex from the queue and added its neighbors that haven't been traversed yet. This traversal finds the shortest paths by the number of edges in an unweighted graph and, accordingly, the vertex farthest by the number of edges, complexity $O(|V| + |E|) = O(ms)$.

(2) Breadth-first traversal with edge sorting (sorted-BFS) is the same as BFS, only we sort the edges from the shortest to the longest. Complexity is $O(ms \log m)$ or $O(ms)$ if the edges are already initially sorted in ascending order of weights.
(3) Dijkstra's algorithm [3] uses a priority queue to store vertices, takes the shortest distance to the vertex from the original one as a priority, selects the vertex with the minimum distance from the queue, adds its neighbors that have not yet been traversed, while calculating or recalculating the distance to them. This algorithm finds the shortest paths by the sum of the lengths of edges in a weighted graph and, accordingly, the most distant vertex by the sum of the lengths of the edges. Complexity is $O(ms \log m)$.

### 5.5 Partitioning a graph into two clusters

Once the two base vertices are chosen, we need to partition the graph into two clusters. The basic idea of partitioning is very simple: one by one, we enroll the nearest vertices to the base vertices into their clusters, observing two important conditions each time:

- cluster size ratio is as close to the desired ratio as possible;
- both clusters are connected graphs.

We use the following approaches to determine the vertex closest to the cluster:

(1) traverse vertices by BFS (aka wave method) as if the graph were unweighted, using edge weights only to determine the order of passing neighbors of the next vertex (we process the nearest neighbors first);
(2) traverse vertices by Dijkstra's algorithm in a weighted graph, passing first the nearest vertices in terms of distance from the base vertex;
(3) traverse vertices by Prim algorithm [17] (as in construction of minimal spanning tree), passing first the nearest vertices in the sense of distance from any vertex of the cluster part already built at that moment.

$$\frac{w\left(Cl_1\right)}{w\left(Cl_2\right)} \approx q.$$

The input of the algorithm is a connected graph $G = (V, E)$, the vertices of which are virtual nodes, and the edges are the published distances $d\left(N_1, N_2\right)$ between data nodes, also two base vertices of clusters $N_1$ and $N_2$ $(N_1 \neq N_2)$ and some positive number $q$. Initially $V = \mathcal{N}$, then $V$ is the cluster chosen for splitting. The algorithm performs the division of all nodes into two non-overlapping subsets (clusters) $Cl_1$ and $Cl_2$ and two graphs into two connected subgraphs, while the number of validators in clusters should approximately correspond in the proportion $q$:

$$\frac{w\left(Cl_1\right)}{w\left(Cl_2\right)} \approx q.$$

For example, in the case of clusters of equal size $q = 1$. This means that the number of validators in the first cluster should be $w\left(Cl_1\right) \approx \frac{q}{q+1} w(V)$, and the number of validators in the second should be $w\left(Cl_2\right) \approx \frac{1}{q+1} w(V)$.

The general idea is that we run some graph traversal algorithm (for example, BFS, Dijkstra or Prim's algorithm) in parallel, starting from both vertices $N_1$ and $N_2$, observing the condition $\frac{w(Cl_1)}{w(Cl_2)} \approx q.$. When traversing the graph, we will paint over the vertices of the graph (virtual nodes), in the first or second color, thus referring them to different clusters. The current vertices (BFS algorithm, Dijkstra) or edges (Prim's algorithm) will be contained in two containers, namely, queues (BFS) or priority queues (Dijkstra's and Prim's algorithms).

Let's write the partitioning algorithm.

(1) We initiate the first traversal of the graph, starting from the vertex $N_1$, adding the vertex $N_1$ to the first container, the cluster $Cl_1$ is empty.
(2) We initiate the second traversal of the graph, starting from the vertex $N_2$, adding the vertex $N_2$ to the second container, the cluster $Cl_2$ is empty.
(3) Until all vertices (nodes) have been traversed (i.e. while at least one of the containers is not empty)
   (a) We choose which cluster we will increase $Cl_1$ or $Cl_2$:
      (i) if there are no reachable vertices to traverse number 1 (container 1 is empty), choose cluster $Cl_2$ ($i = 2$);
      (ii) otherwise if there are no reachable vertices to traverse number 2 (container 2 is empty), choose cluster $Cl_1$ ($i = 1$);
      (iii) otherwise if $w(Cl_1) \leq q \cdot w(Cl_2)$, select the cluster $Cl_1$ ($i = 1$), otherwise select the cluster $Cl_2$ ($i = 2$).
   (b) Select the first vertex $N_C$ in the selected container number $i$.
   (c) Add it to the cluster $Cl_i$, increase its weight by $|N_C|$.
   (d) Delete $N_C$ from container number $i$, as well as from another container, if it is contained there.
   (e) We add to the container number $i$ all adjacent to the vertex $N_C$ still uncolored vertices (BFS, Dijkstra algorithms) or edges leading to uncolored vertices (Prim's algorithm), calculating the distances to them as:
      - distance to $N_C$ plus one — in case of BFS;
      - the sum of the distance to $N_C$ and the corresponding weight of the edge — in the case of Dijkstra's algorithm (update the distance value only if it is less than the currently stored distance to the vertex, initially all vertex distances are infinity, except for the starting vertex, for which the distance equals zero);
      - edge weight in the case of Prim's algorithm.

## 6 ASSESSING THE QUALITY OF CLUSTERING

To assess the quality of clustering we will use the following indicators.

### 6.1 Silhouette

Let $N_i \in Cl_j$ and $a(N_i, Cl_j) = \frac{1}{|Cl_j|} \sum_{N \in Cl_j} d(N_i, N)$ be the average distance from node $N_i$ to other objects of cluster (compactness) and let

$$b(N_i, Cl_j) = \min_{Cl \in C \setminus Cl_j} \frac{1}{|Cl|} \sum_{N \in Cl} d(N_i, N)$$

minimum average distance from node $N_i$ to objects from other clusters (separability). Then the estimation of clustering quality will be the following value (silhouette [18]):

$$Sil(C) = \frac{1}{m} \sum_{Cl \in C} \sum_{N \in Cl} \frac{b(N, Cl) - a)N, Cl)}{\max\{a(N, Cl), b(N, Cl)\}}.$$

It is easy to see that $-1 \leqslant Sil(C) \leqslant 1$. The closer $Sil(C)$ is to 1, the better the clustering.

### 6.2 Diameter index

Let $D(X) = \{d(a, b) \colon a, b \in X\}$ be the diameter of the set $X$ and let

$$I_D = \frac{D(\mathcal{N}) - \max_{Cl \in C} D(Cl)}{D(\mathcal{N})}$$

be the diameter index. This index shows how much the diameter of clusters is smaller than the diameter of the whole network, and varies from 0 to 1. The closer this parameter is to 1, the better the clustering.

### 6.3 Sphericity index

We call a node $N_{Cl}$ the center of a cluster $Cl$ if

$$\sum_{A\in Cl} d(A, N_{Cl}) = \min\left\{\sum_{A\in Cl} d(A, N) \colon N \in Cl\right\}.$$

Let $S(C) = \frac{1}{k}\sum_{Cl\in C}\frac{1}{|Cl|}\sum_{A\in Cl} d(A, N_{Cl})$ be the average distance from cluster centers to cluster elements,

$$T(C) = \frac{1}{k^2-k}\sum_{A,B\in C, A\neq B} d(N_A, N_B)$$

be the average distance between cluster centers and

$$I_S = \frac{T(C) - S(C)}{T(C)}.$$

The parameter $I_S$ shows the sphericity of clustering. The closer this parameter is to 1, the better the clustering.

### 6.4 Standard deviation index

For the set $X$ we denote

$$D_{sq}(X) = \left(\frac{1}{|X|^2-|X|}\sum_{A,B\in X, A\neq B} d^2(A, B)\right)^{\frac{1}{2}}.$$

Then the standard deviation index is

$$I_{sq}(C) = \frac{D_{sq}(\mathcal{N}) - E(D_{sq}, C)}{D_{sq}(\mathcal{N})},$$

where $E(D_{sq}, C) = \frac{1}{k}\sum_{Cl\in C} D_{sq}(Cl)$.

### 6.5 Uniformity of cluster weights

Let

$$I_W = \left(\frac{1}{k}\sum_{Cl\in C}\left(\frac{w(Cl)}{|\mathcal{V}|} - \frac{1}{k}\right)^2\right)^{\frac{1}{2}}$$

be standard deviation of the relative weights of clusters from the best value.

## 7 RELIABILITY AND STABILITY OF THE SUBNETTING ALGORITHMS

When dividing the blockDAG of the Waterfall network into subnetworks, there may be the following problems:

(1) some virtual node may not provide data in time;
(2) some virtual node may provide incorrect or unrepresentative data due to failures in its operation or due to instability of the Internet connection;
(3) some virtual node may provide obviously incorrect data for some reasons of its own benefit, for example, it wants to get into a certain subnetwork or in order to attack the protocol.

Let us call a subnetting algorithm resistant to local changes if changing the data of a small number of virtual nodes (including adding, removing virtual nodes and changing the number of validators on the nodes) does not lead to significant changes in the composition of subnetworks.

If the algorithm is not resistant to local changes, then a small number of nodes can completely change the clustering. For example, a node can perform a “connect two distant nodes” attack: a node declares proximity to two distant nodes, e.g., those two nodes that in the past division into subnetworks were the base ones for division into clusters (that is, the most distant).

Let’s outline some way to deal with such attacks. To begin with, let’s say that the connection of two nodes is one-way, if only one of these nodes declares it and a node is called one-way if all of its links are one-way. In other words, a node is one-way if it is not mentioned in the lists of other nodes.

If a node is one-way, then it means that it is a rather slow node or with slow internet. The node is, in a certain sense, peripheral and does not participate in the best connections of nodes. Obviously, the node should not globally affect the partitioning of other nodes into subnets. Thus, to combat the instability of clustering, it is necessary to give preference to two-way links, and use one-way links according to the residual principle.

We will study resistance and stability issues of different subnetting approaches and algorithms in future works.

## 8 REFINEMENT OF THE WATERFALL TRANSPORT LAYER

Considering the fundamental difference in how nodes work after implementing subnetworks, namely that now nodes must distribute transactions only over their subnetwork, but not globally, it is also necessary to rework the transport layer of the Waterfall protocol. It is initially based on the transport layer of the Etherium, namely devp2p library.

Each node must now report the number of the subnetwork on which it operates in its status message. If a given node does not contain validators, then it can choose a subnetwork number arbitrarily. If the validators are running on this node, then for the normal operation of the system their subnetwork number must match each other (this is achieved by combining them into a virtual node), the blockDAG node just reports this subnetwork number.

Let us describe the additions to the procedure for creating transactions and the features of the operation of subnetwork nodes:

- Each block is sent out throughout the network.
- Each transaction, when created, must have the number of the subnetwork to which it belongs.
- Each node records and processes transactions only on its subnetwork.

The operation of subnetting algorithms with a high probability leads to a result when a node has nodes of its own subnetwork among its neighbors. Nevertheless, if, after subnetting, there are no or few nodes from its subnetwork among the neighbors of a certain node, then it requests tables of their neighbors from its neighbors and from the contents of these tables selects for itself a sufficient number of neighbors belonging to its subnetwork. If after that the number of neighbors from its subnetwork is not enough, then the process is repeated iteratively (which is extremely unlikely in practice).

Another modification of the protocol will be needed at the end of the era. The result of subnetting is fixed by the finalized state of the block DAG two epochs before the end of the era. One epoch is enough for finalization, another one for working out the subnetting algorithm. After executing the clustering algorithm, the node determines a new subnet number and, if it is different from the current one, then the node temporarily saves two subnet numbers - current and future (for the next era). Accordingly, such a node maintains two transaction pools for two subnets and exchanges

transactions with nodes on both subnets (nodes whose current or future subnets match the current or future subnet of this node). In particular, immediately after determining a new subnetwork number that is different from the current one, it is necessary to start the process of synchronizing the transaction pool of the future subnetwork, since at this moment it is empty and there may be transactions in the network that belong to the future subnetwork of the node that will not have time to be published or finalized at this era.

## 9 MODELING

The input data for modeling are the set of nodes of blockDAG and the number of validators on each of them. To simulate distances $d_v(N_1, N_2)$ between nodes, we use a "geographic" model, where each node is defined by a point with coordinates $(x, y)$. The distance between nodes $d_v(N_1, N_2)$ is modeled by the formula

$$d_v(N_1, N_2) = \text{delay}(N_2) + c_v d_e(N_1, N_2),$$

where $\text{delay}(N_2)$ is some random block generation delay by node $N_2$ (depends only on node $N_2$), $c_v \approx 1$ is some random factor symbolizing link instability (each time generated independently) and $d_e(N_1, N_2)$ is the Euclidean distance between points that represent nodes $N_1$ and $N_2$.

In total, 30 datasets were generated, each one consisted of 300 nodes with random coordinates and a random weight (validator count) from 1 to 25.

Table 1. Average clustering indices by 30 data sets, $k = 4$, $m = 300$, $L = 25$.

| **Clustering Method** | $Sil$ | $I_D$ | $I_S$ | $I_{sq}$ | $I_W \cdot 100\%$ |
|---|---|---|---|---|---|
| AHC, $d_1$, $\text{dist}_{max}$ | 0.26 | 0.12 | 0.28 | 0.21 | 5.16% |
| AHC, $d_{max}$, $\text{dist}_{max}$ | 0.25 | 0.10 | 0.24 | 0.21 | 5.16% |
| AHC, $d_1$, $\text{dist}_{min}$ | 0.40 | 0.24 | 0.49 | 0.36 | 5.87% |
| AHC, $d_{max}$, $\text{dist}_{min}$ | 0.40 | 0.23 | 0.50 | 0.35 | 5.51% |
| DHC, $d_{max}$-double_dijkstra-dijkstra | 0.43 | 0.27 | 0.54 | 0.37 | 0.26% |
| DHC, $d_1$-double_dijkstra-dijkstra | 0.43 | 0.27 | 0.54 | 0.37 | 0.28% |
| DHC, $d_{max}$-global_dijkstra-dijkstra | 0.44 | 0.32 | 0.56 | 0.39 | 0.17% |
| DHC, $d_1$-global_dijkstra-dijkstra | 0.44 | 0.31 | 0.52 | 0.39 | 0.20% |
| DHC, $d_{max}$-global_bfs-bfs | 0.44 | 0.32 | 0.56 | 0.39 | 0.19% |
| DHC, $d_1$-global_bfs-bfs | 0.44 | 0.32 | 0.56 | 0.39 | 0.19% |
| DHC, $d_{max}$-global_bfs-sorted_bfs | 0.45 | 0.33 | 0.56 | 0.39 | 0.25% |
| DHC, $d_1$-global_bfs-sorted_bfs | 0.44 | 0.33 | 0.55 | 0.39 | 0.20% |
| DHC, $d_{max}$-double_bfs-bfs | 0.43 | 0.29 | 0.53 | 0.37 | 0.23% |
| DHC, $d_1$-double_bfs-bfs | 0.43 | 0.29 | 0.53 | 0.37 | 0.23% |
| DHC, $d_{max}$-double_bfs-sorted_bfs | 0.43 | 0.28 | 0.53 | 0.37 | 0.15% |
| DHC, $d_1$-double_bfs-sorted_bfs | 0.42 | 0.29 | 0.51 | 0.37 | 0.22% |
| DHC, $d_{max}$-double_bfs-dijkstra | 0.42 | 0.25 | 0.51 | 0.36 | 0.17% |
| DHC, $d_1$-double_bfs-dijkstra | 0.43 | 0.27 | 0.52 | 0.37 | 0.17% |
| DHC, $d_{max}$-double_bfs-prim | 0.41 | 0.24 | 0.48 | 0.35 | 1.77% |
| DHC, $d_1$-double_bfs-prim | 0.41 | 0.26 | 0.51 | 0.35 | 1.66% |

The simulation results are summarized in Table 1. During the simulation process, the algorithms were divided into 2 groups: agglomerative and divisive ones. To specify each particular algorithm, a number of parameters were introduced to describe the internal behavior of a given algorithm. These parameters are separated by a dash or comma.

The first parameter tells the type of algorithm, the second describes the distance function ($d_1$ or $d_{max}$). For agglomerative algorithms, the third parameter describes the distance function between clusters ($\text{dist}_{max}$ or $\text{dist}_{min}$). For divisive algorithms, the third parameter denotes the algorithm that is used to find the farthest nodes. The name double_traverse denotes the double traverse method, where traverse is the graph traversal method (BFS or Dijkstra), global_traverse is that traverse is run in turn from each vertex to find the maximum distance in the entire graph. The last parameter specifies the traversal method for clustering (BFS, sorted_bfs, dijkstra, prim).

Figure 1 shows these results graphically: each of the clustering quality indices corresponds to a strip of a certain color in the image below.

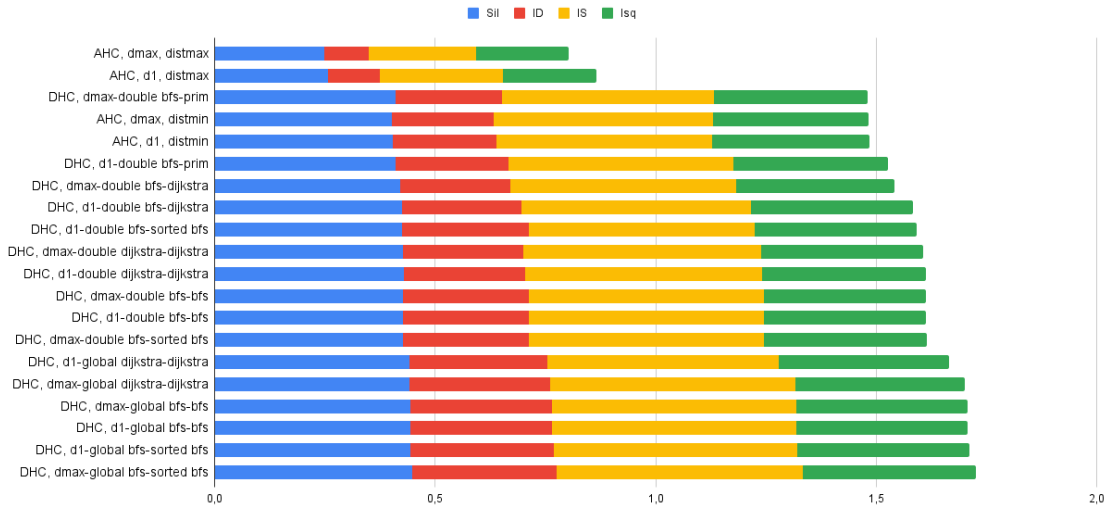


Fig. 1. Comparison diagram of the results of different clustering methods by 30 data sets, $k = 4$, $m = 300$, $L = 25$

Let us illustrate the clustering results of some of the above algorithms. To begin with, on the figures 2 and 3 we will display on the coordinate plane a set of nodes of one of the 30 generated datasets. Each of the 300 nodes is displayed as a small circle with the center coordinates matching the data in the set and the color corresponding to the cluster number, determined by the given algorithm.

We additionally tested the operation of the algorithms on different sets of nodes, with coordinates chosen randomly not from some rectangle, but only at the pixel locations of some (different) images, in order to challenge the operation of the algorithms on data of a more complex structure, rather than uniformly distributed. In general, the simulation results approximately correspond to those given in the table above, we illustrate the obtained clusterings in the figure 4.

## 10 CONCLUSION

In this paper, we consider the problem of partitioning the Waterfall network (or similar blockDAG networks) into subnetworks to optimize network interaction between nodes. The paper considers the main approaches to such partitioning, in particular, the question of the need to change the block generation rule, and the basic requirements of the node clustering algorithm. The main approach is to publish nearest neighbor metric information by each validator in blockDAG at the moment when a new block is published by this validator.

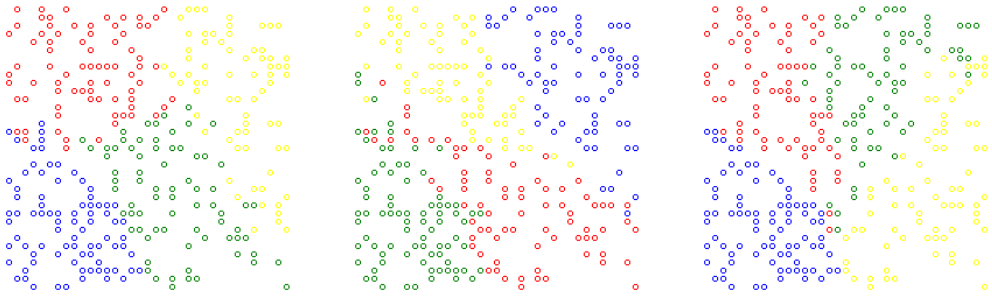

Fig. 2. Visualization of methods which gave the best results, namely DHC, $d_{max}$-global_bfs-sorted_bfs; DHC, $d_{max}$-global_bfs-bfs; DHC, $d_{max}$-global_dijkstra-dijkstra

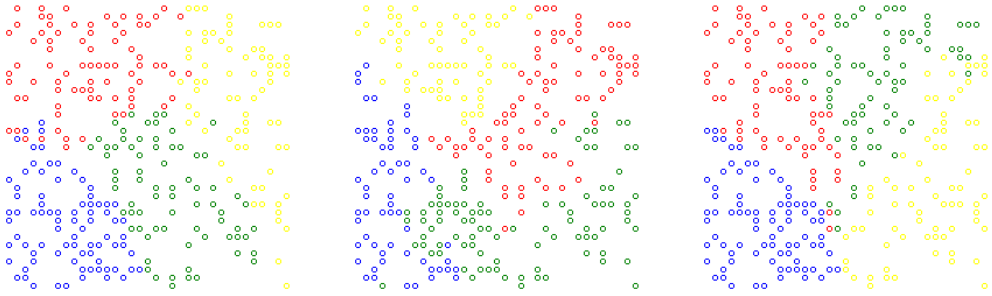

Fig. 3. Visualization of the results of the best methods in the following categories(from left to right): agglomerative (AHC, $d_1$, dist$_{min}$), divisional with the double parameter (DHC, $d_1$-double_bfs-bfs) and divisional with the global parameter (DHC, $d_{max}$-global_bfs-bfs)

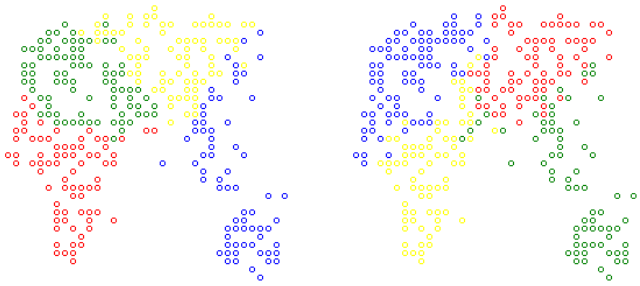

Fig. 4. Visualization of the results of DHC, $d_{max}$-global_bfs-sorted_bfs; DHC, $d_{max}$-global_bfs-bfs on a set of nodes generated by a picture

We introduce a notion of virtual node to separate the storage level of blockDAG from network level and formulate methods to collect and publish information that reflects the quality of communication of virtual nodes. Methods are formulated that allow all nodes of the blockDAG network to retrieve from the published information an undirected weighted sparse graph of links between virtual nodes. Thus, we are reduced to the problem of graph clustering of virtual nodes, with additional conditions on the low computational complexity of the algorithm, and the approximate equality of the obtained clusters by the number of validators in them. This paper presents different variations of hierarchical agglomerative and divisional algorithms, modified specifically to solve this problem. The main clustering quality metrics are also considered. In addition, modeling of these algorithms was conducted, and the results of their work on model sets were obtained.

As a result of numerical simulation, the following results were obtained: agglomerative methods of clustering have much greater non-uniformity of cluster weights (the difference is up to two times). Other than on test-source data, the clusters from one virtual node very rarely appeared, which is absolutely inadmissible. Also, the computational complexity of agglomerative methods is too high. Nevertheless, the distance function version $\text{dist}_{min}$ performs on average better than $\text{dist}_{max}$.

Among the divisional algorithms, different methods of base node extraction and node clustering were considered. Slightly better results are given by methods of global search for the most distant nodes in the graph global_dijkstra and global_bfs. However, these methods require significant computational cost. The fastest methods are those based on double BFS (double_bfs) traversal. For graph partitioning, you can also use the fastest parallel width traversal of vertices (bfs or its version with sorted_bfs). The method based on these traversals works for $O(msk)$, gives basic quality and uniformity indices only slightly inferior to global_dijkstra and global_bfs methods, and works stably on different test data. Therefore, this method can be the basis of the method for partitioning the blockDAG Waterfall network into subnetworks.

At the moment, the process of implementing the concept of subnetworks, the necessary algorithms and changes in the network layer of the Waterfall blockDAG is underway. Also, an important issue to study will be the robustness of the obtained clustering methods and distance metrics to faulty information by some nodes and/or intentional distortion by dishonest blockDAG participants.

## REFERENCES


[1] He Bai, Geming Xia, and Shaojing Fu. 2019. A Two-Layer-Consensus Based Blockchain Architecture for IoT. In *2019 IEEE 9th International Conference on Electronics Information and Emergency Communication (ICEIEC)*. IEEE, New York, 1–6. https://doi.org/10.1109/ICEIEC.2019.8784458

[2] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, Dawn Song, and Roger Wattenhofer. 2016. On Scaling Decentralized Blockchains. In *Financial Cryptography and Data Security*. Springer Berlin Heidelberg, Berlin, Heidelberg, 106–125.

[3] E. W. Dijkstra. 1959. A note on two problems in connexion with graphs. *Numer. Math.* 1, 1 (1959), 269–271. https://doi.org/10.1007/BF01386390

[4] Nikolai Durov. 2019. *Telegram Open Network*. TON. https://ton.org/ton.pdf

[5] Keke Gai, Ziyue Hu, Liehuang Zhu, Ruili Wang, and Zijian Zhang. 2020. Blockchain Meets DAG: A BlockDAG Consensus Mechanism. In *Algorithms and Architectures for Parallel Processing*, Meikang Qiu (Ed.). Springer International Publishing, Cham, 110–125.

[6] Adem Efe Gencer, Robbert van Renesse, and Emin Gün Sirer. 2017. Short Paper: Service-Oriented Sharding for Blockchains. In *Financial Cryptography and Data Security*, Aggelos Kiayias (Ed.). Springer International Publishing, Cham, 393–401.

[7] Sergii Grybniak, Dmytro Dmytryshyn, Yevhen Leonchyk, Igor Mazurok, Oleksandr Nashyvan, and Ruslan Shanin. 2022. Waterfall: A Scalable Distributed LedgerTechnology. (2022). 6 pages.

[8] Sergii Grybniak, Yevhen Leonchyk, Igor Mazurok, Oleksandr Nashyvan, and Ruslan Shanin. 2022. Waterfall: Gozalandia. Distributed protocol with fast finality and proven safety and liveness. (2022). 10 pages.

[9] V. T. Hoang, B. Morris, and P. Rogaway. 2012. An Enciphering Scheme Based on a Card Shuffle. In *Advances in Cryptology – CRYPTO 2012*. Springer Berlin Heidelberg, Berlin, Heidelberg, 1–13.

[10] L. Hubert. 1973. Monotone invariant clustering procedures. *Psychometrika* 38, 1 (1973), 47–62.

[11] Amr M. Khalifa, Ayman M. Bahaa-Eldin, and Mohamed Aly Sobh. 2019. Blockchain and its Alternative Distributed Ledgers - A Survey. In *2019 14th International Conference on Computer Engineering and Systems (ICCES)*. IEEE, New York, 118–125. https://doi.org/10.1109/ICCES48960.2019.9068183

[12] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. 2018. OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, New York, 583–598. https://doi.org/10.1109/SP.2018.000-5

[13] D. C. Kozen. 1992. *The Design and Analysis of Algorithms*. Springer, New York, Chapter Depth-First and Breadth-First Search, 19–24. https://doi.org/10.1007/978-1-4612-4400-4_4

[14] Cheng Li and Liang-Jie Zhang. 2017. A Blockchain Based New Secure Multi-Layer Network Model for Internet of Things. In *2017 IEEE International Congress on Internet of Things (ICIOT)*. IEEE, New York, 33–41. https://doi.org/10.1109/IEEE.ICIOT.2017.34

[15] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. 2016. A Secure Sharding Protocol For Open Blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (Vienna, Austria) *(CCS '16)*. Association for Computing Machinery, New York, NY, USA, 17–30. https://doi.org/10.1145/2976749.2978389

[16] Quan Nguyen, Andre Cronje, Michael Kong, Egor Lysenko, and Alex Guzev. 2021. Lachesis: Scalable Asynchronous BFT on DAG Streams. arXiv:2108.01900v1. , 45 pages. https://arxiv.org/abs/2108.01900.

[17] R. C. Prim. 1957. Shortest connection networks and some generalizations. *The Bell System Technical Journal* 36, 6 (1957), 1389–1401. https://doi.org/10.1002/j.1538-7305.1957.tb01515.x

[18] Peter J. Rousseeuw. 1987. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.* 20 (1987), 53–65. https://doi.org/10.1016/0377-0427(87)90125-7

[19] Maurice Roux. 2018. A comparative study of divisive and agglomerativehierarchical clustering algorithms. *Journal of Classification* 35 (2018), 345–366.

[20] Abhishek Kumar Singh. 2020. A Multi-Layered Network Model for Blockchain Based Security Surveillance system. In *2020 IEEE International Conference for Innovation in Technology (INOCON)*. IEEE, New York, 1–5. https://doi.org/10.1109/INOCON50539.2020.9298422

[21] Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. 2016. SPECTRE: A Fast and Scalable Cryptocurrency Protocol. Cryptology ePrint Archive, Paper 2016/1159. https://eprint.iacr.org/2016/1159

[22] Yonatan Sompolinsky, Shai Wyborski, and Aviv Zohar. 2018. PHANTOM and GHOSTDAG: A Scalable Generalization of Nakamoto Consensus. Cryptology ePrint Archive, Paper 2018/104. https://eprint.iacr.org/2018/104

[23] Ethereum Team. 2022. *Topics and messages*. Ethereum. https://github.com/ethereum/consensus-specs/blob/dev/specs/phase0/p2p-interface.md#topics-and-messages

[24] Kaspa Team. 2020. *Subnetworks*. Kaspa. https://kaspa.gitbook.io/kaspa/archive/archive/components/kaspad-full-node/reference/subnetworks-1

[25] Qin Wang, Jiangshan Yu, Shiping Chen, and Yang Xiang. 2020. SoK: Diving into DAG-based Blockchain Systems. arXiv:2012.06128v2. , 36 pages. https://arxiv.org/abs/2012.06128.

[26] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. 2018. RapidChain: Scaling Blockchain via Full Sharding. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Toronto, Canada) *(CCS '18)*. Association for Computing Machinery, New York, NY, USA, 931–948. https://doi.org/10.1145/3243734.3243853

Received 15 December 2022; revised not yet; accepted not yet