Polystyrene: the Decentralized Data Shape That Never Dies

Simon Bouget, Anne-Marie Kermarrec, Hoel Kervadec, François Taïani

Polystyrene: The Decentralized Data Shape that Never Dies

Simon Bouget
ENS Rennes, 35170 Bruz - France
Email: simon.bouget@ens-rennes.fr

Anne-Marie Kermarrec
Inria, 35042 Rennes, France
Email: anne-marie.kermarrec@inria.fr

Hoel Kervadec
INSA de Rennes
35708 Rennes, France
Email: hoel.kervadec@insa-rennes.fr

François Taïani
Université de Rennes 1 – IRISA – ESIR
35042 Rennes, France
Email: francois.taiani@irisa.fr

***Abstract*—Decentralized topology construction protocols organize nodes along a predefined topology (e.g. a torus, ring, or hypercube). Such topologies have been used in many contexts ranging from routing and storage systems, to publish-subscribe and event dissemination. Since most topologies assume no correlation between the physical location of nodes and their positions in the topology, they do not handle catastrophic failures well, in which a whole region of the topology disappears. When this occurs, the overall *shape* of the system typically gets lost. This is highly problematic in applications in which overlay nodes are used to map a virtual data space, be it for routing, indexing or storage. In this paper, we propose a novel decentralized approach that maintains the initial shape of the topology even if a large (consecutive) portion of the topology fails. Our approach relies on the dynamic decoupling between physical nodes and virtual ones enabling a fast reshaping. For instance, our results show that a 51,200-node torus converges back to a full torus in only 10 rounds after 50% of the nodes have crashed. Our protocol is both simple and flexible and provides a novel form of collective survivability that goes beyond the current state of the art.**

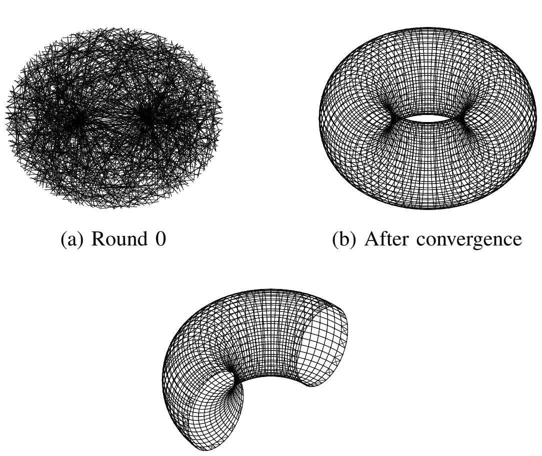
I. Introduction

Decentralized topology construction protocols [1], [2], [3] are natural candidates to support the numerous overlay networks that have been proposed for over a decade now in the context of many P2P and cloud-based applications such as telecommunications (Skype), streaming systems [4], [5], pub-sub systems [6], [7], and key-value stores [3], [8], [9]. Among these, gossip topology construction protocols have received a large amount of attention [1], [2], [10], [11] due to their inherent ability to scale, survive, and adapt. These gossip protocols exploit epidemic interactions to progressively organize nodes along a predefined topology (e.g. torus, ring, hypercube), and have applications in contexts ranging from routing to storage [3], recommendations [12], and event dissemination [13].

Since most of these systems assume no correlation between node failures and their positions in the topology, topology construction protocols (and overlay systems in general) do not handle well *catastrophic correlated failures* in which a whole region of the topology disappears. Unfortunately, if node positions are no longer assigned at random in the topology but depend on some application criteria, such as semantics or geography, this assumption no longer holds. For instance, if a large consecutive portion of an overlay is deployed in one single datacenter, the whole structure might be jeopardized by a power failure of the whole datacenter, even if the surviving part is able to heal and mend its disconnected nodes. When this occurs typically, the overall *shape* of the system gets lost, even when surviving nodes succeed in locally reconfiguring their links (Fig. 1).

As decentralized overlay-based solutions get deployed in datacenters and clouds, the ability to recover gracefully from catastrophic failures appears increasingly important for at least two reasons: (i) datacenters and clouds can greatly benefit from placement decisions that are correlated with the underlying physical infrastructure (e.g. all the virtual machines handling contiguous keys hosted in the same rack) a strategy already proposed for map-reduce clusters [14], [15]; and (ii) this correlation with the physical infrastructure brings very real risks of systemic catastrophic failures, in which a large part of an overlay topology suddenly disappears. This is particularly likely to happen in multi-datacenters or multi-cloud scenarios. Losing the shape of the topology might affect system performance, e.g. routing or load balancing, which often relies on a uniform distribution of nodes along the topology.

Similarly the same challenges apply when considering nano-datacenters and community clouds based on home-hosted servers. To support such applications, we argue in this paper that we need decentralized topology construction protocols that can recover from catastrophic correlated failures and *reform* the system's original shape when this happens. To address this issue, we propose a novel decentralized epidemic protocol, called *Polystyrene*, that can memorize and maintain the general shape of a target topology even in extreme situations. Our approach leverages four simple epidemic processes that are all based on the idea of decoupling physical nodes from the data points defining the shape the overlay should converge to. Our results convey the efficiency of our approach. For instance, we show that, using Polystyrene, an overlay containing 51,200 nodes organized in a torus only requires 10 rounds to reform the torus after one half of the overlay has crashed. This result is obtained in a fully decentralized epidemic approach, and does not assume any global knowledge or central form of coordination.



(a) Round 0

(b) After convergence

(c) Catastrophic failure

Figure 1: Catastrophic correlated failure in a decentralized topology construction protocol (T-Man, 3200 nodes)

The remainder of the paper is organized as follows. We first present the problem we address, and the intuition behind our approach (Sec. II). We then expose our system model and our protocol (Sec. III), before moving on to an experimental evaluation of our solution (Sec. IV). Finally Sec. V presents related work, while Sec. VI concludes with some reflection on the perspectives we think our contribution opens.

II. Background and problem statement

Topology construction protocols seek to organize distributed nodes into a pre-determined *overlay shape* (e.g. a torus, a ring). Typically these protocols aim to distribute nodes uniformly on the shape, and to link up nodes that lay close to each other. They can be fully decentralized, leading to solutions which are particularly scalable and resilient to (uncorrelated) failures: This includes Pastry [16] for a ring overlay, CAN [3] for a Euclidean space, or even RPS for a random graph [17]. Some of these decentralized solutions, such as T-Man [1] and Vicinity [2], can even construct almost any arbitrary topology. The scalability and robustness of these solutions have made them particularly well adapted to large scale self-organizing systems such as decentralized social networks [10], [12], news recommendation engines [13], and peer-to-peer storage systems [9], [18], [19].

A. The problem: Catastrophic correlated failures

Most of these approaches, however, assume that nodes that are neighbors in the overlay topology do not share common failure modes, i.e. they ignore *correlated* failures. This is often justified as these protocols typically seek to maximize the diversity between nodes that are topologically close [20].

Some applications, however, can benefit from synergies between the overlay and the supporting physical infrastructure (the networking layer)—e.g. geographical routing in CAN [3]—or the application space—as in Meghdoot [6] (cross-layer optimization). As decentralized overlay-based solutions get deployed in datacenters and clouds, these synergies become even more compelling to benefit from data-locality (e.g. all the virtual machines handling contiguous keys hosted in the same rack), for instance by exploiting rack-level placement techniques [14], [15]. Unfortunately coupling an overlay and its underlying infrastructure brings very real risks of systemic catastrophic failures, in which a large region of the overlay might suddenly disappear. This is particular likely to happen in multi-datacenters or multi-cloud scenarios.

Fig. 1 provides an example of a correlated catastrophic failure. The first two subfigures (1a and 1b) depict how the decentralized topology construction protocol T-Man [1] converges to a torus-shaped grid. Fig. 1c shows T-Man's behavior when the nodes located in the right-hand side of the torus crash simultaneously. Boundary nodes recreate links with their closest surviving neighbors, but the overall shape of the torus is lost. This is likely to affect the overall performance of the system, as most overlays strive to achieve a uniform distribution of their nodes in the topology. This might impact for instance the system's routing efficiency or create load unbalance. Recovering the shape is precisely the problem we address in this paper.

In the following, we provide a brief reminder of decentralized topology construction protocols, and of their behavior under catastrophic failures. We then sketch the main intuition of our contribution, before detailing the workings of our algorithm in the next section.

B. Topology construction protocols

Topology construction protocols assume each node has a position, which can be used to compute a distance between any pair of nodes. For the sake of clarity, we assume nodes take their positions from a continuous space with a small dimension as in [1] and use the standard Euclidean distance, but the same mechanisms can be applied to any metric space. For instance, topology construction has been used in the past to organize similar users in social networks based on their profiles (their "positions" in the metric space of all profiles), for various aims such as recommendation or search [2], [10], [12].

Topology construction protocols seek to self-organize a network so that each node ends up connected to its k closest nodes. (k is a parameter of the system.) To reach this goal, they use a decentralized greedy procedure that exploits two dynamic overlays layered on top of one another (Fig. 2). In each overlay, nodes maintain a small list of neighbors (its *view*). For instance, in Fig. 2, Node **A** is connected to Nodes **B**, **C** and **D** in the bottom overlay (the *peer-sampling* overlay), and to Nodes **B** and **C** in the upper overlay (the *topology construction* overlay). Periodically, each node selects a node from its view and exchanges information about its neighbors with the final goal of converging to a topology in which each node is connected to its closest neighbors.

More precisely, the bottom overlay (peer sampling) provides each node with a random sample of the rest of the network. This is achieved by having nodes exchange and shuffle their neighbors' list in asynchronous *gossip rounds* to maximize the randomness of the peer-sampling overlay graph over time [17].

The topology construction overlay sits on top of the peer-sampling one. This second overlay is initialized using views

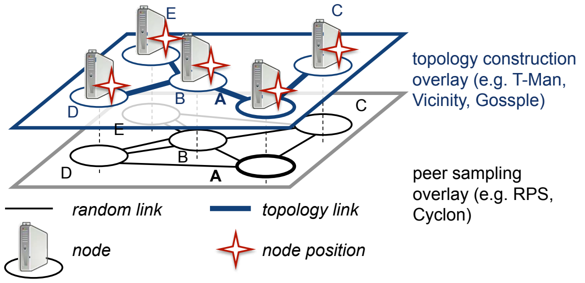

E
C
D
B
A
topology construction overlay (e.g. T-Man, Vicinity, Gossple)
E
C
D
B
A
peer sampling overlay (e.g. RPS, Cyclon)
random link
topology link
node
node position

Figure 2: Example of a gossip-based topology construction

from the peer-sampling overlay, and implements a local greedy optimization procedure that leverages the current neighbors of a node [1], augmented in some protocols by additional random neighbors returned by the peer-sampling overlay [2]. Potential new neighbors found in the peer-sampling overlay guarantees the convergence of the topology under stable conditions, while neighbors from the topology-construction overlay speed up this convergence: nodes that are already close swap their current neighbors and attempt to construct a better neighborhood based on the other node's information.

C. *Polystyrene design rationale*

Polystyrene, our shape-preserving decentralized protocol, comes in the form of an add-on layer that can be plugged into any decentralized topology construction algorithm (Fig. 3). The simple intuition behind Polystyrene consists in decoupling the positions of the nodes in the topology from the nodes themselves. As in T-Man or Vicinity, each Polystyrene node starts with one position (termed a *data point* in the rest of the paper). Data points differ from virtual nodes [19], [8] as they do not maintain any neighborhood. They are passive data, and do not execute any protocol. The set of all data points defines the underlying *shape* the topology should converge to. However, contrary to traditional topology construction systems, we allow Polystyrene nodes to change their positions (i.e. *migrate*) when nodes fail, and to redistribute themselves around the target shape. As a result, the original shape is maintained, albeit at a lower sampling density, resulting from the lower number of surviving nodes.

The above strategy requires a few additional mechanisms to work: First, we need some form of *memory* of the original shape (e.g. a ring, a torus) to ensure the positions of failed nodes survive the demise of their containing nodes. That is because we do not assume any global knowledge of the target topology by the participating nodes. Second, if surviving nodes are to occupy the space left empty by failed ones, the system will naturally end up with more data points than nodes, so nodes should be able to *host* several data points simultaneously.

We solve these issues by storing two sets of data points per node (Fig. 3): A first set of *guest* data points hold the points the node is in charge of, either as initial assignment or as a result of failures. We say that the node is a *primary holder* of these guest data points. A second set of *ghost* data points contain copies of data held elsewhere in the network. When Polystyrene starts, there are no ghosts, and only one guest data point per node: the node's original position. At any given time, guest data points are used to derive a node's actual position, which is then fed to the underlying topology construction protocol (Fig. 3). In this paper, we use a simple projection mechanism, but this is an independent piece of our protocol that can be easily adapted to more complex situation.

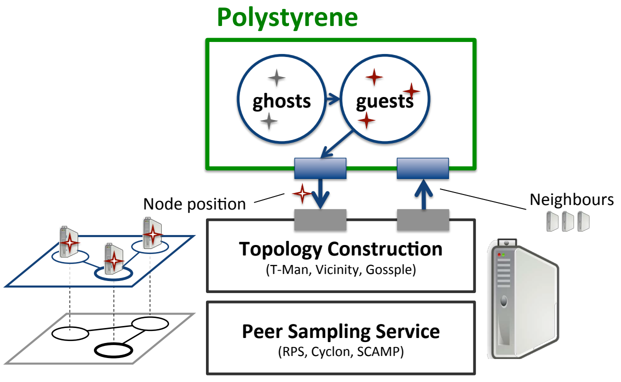


Figure 3: Polystyrene's architecture

Decoupling nodes from data points allows us to implement the migration we need to redistribute nodes around the target shape when catastrophic failures occur. Nodes migrate by periodically exchanging guest data points in order to reach a density-aware tessellation of the data space, i.e. a partition of data points across physical nodes that seeks to maximize locality (described in more detail in Sec. III). After each exchange, each node recomputes the position it provides to the underlying topology construction algorithm, moving in effect around the shape. In other words, nodes migrate by following the migration of their data points.

The ghost data points originally play no role in this migration process. Ghosts are just deactivated copies, that remain so until the node holding them detects that their primary holder has failed. When this happens, the ghost holder *activates* the relevant ghost data points into its guest set, and uses them from now on to drive its migration.

In the following section we revisit in more details each of these mechanisms, and provide the concrete algorithms we use to manage ghosts and guests, before moving on to our evaluation in Sec. IV.

III. The Polystyrene protocol

A. *System model*

We consider a set of message-passing nodes that communicate over reliable channels (e.g. TCP). Each node possesses an original position in the form of a *data point* (e.g. a list of items, a 3D point) taken from a *data space* (resp. the power-set of items, a 3D space).

The original positions of all nodes in the system define the target *shape* that the system should maintain[1]. The only constraint on this data space is that a distance can be computed between any two data points (i.e. it is a *metric space*).

[1] For ease of exposition, we assume this shape is static in the rest of the paper. It could, however, keep evolving as the algorithm executes.

Table I: A node's local state

guests	the data points currently hosted by the local node
pos	the node's virtual position
ghosts	a dictionary of inactivated data points replicated to this node, with their original nodes as keys. E.g. $p.\text{ghosts}[q]$ contains the state sent by q to p. $keys(p.\text{ghosts})$ denotes the set of nodes that have sent replicated data points to p.
backups	the nodes where the local node has replicated its state to. In the previous example, we would have $p \in q.\text{backups}$.

We assume a crash-stop fault model: nodes fail by crashing, and do not recover. We also assume nodes have access to a (possibly imperfect) failure detector (the failed variable in our pseudo code). In practice, a reactive ping mechanism, or heartbeats may be used.

In addition to its original position, each node maintains a set of four local variables listed in Table I. The first variable, guests, allows a node to *host* data points, and is a central element of our protocol. Originally, guests only contains one data point: the node's initial position. Because guests may contain more than one data point, it cannot be used directly by the underlying topology construction algorithm. Instead guests is used to compute a summary position, **pos**, that represents the node's position used for the topology construction. As for guests, **pos** is initialized with the node's initial position.

ghosts and backup are both used to insure data points survive node crashes. ghosts is a dictionary of data points that have been replicated to the local node, with the nodes from which they originate used as keys.backups is the list of nodes to which the local node has replicated its guests. Both ghosts and backups are originally empty.

As in traditional decentralized clustering algorithms [1], [2], we assume a two-layer architecture, with the lower layer providing a peer-sampling service [21], [17], and the higher layer providing a decentralized topology construction mechanism (e.g. [1]), based on each node's position. Polystyrene executes on top of the topology construction layer. For illustration purposes, we use T-Man [1] in this paper.

B. Algorithm: Overview

First, Polystyrene provides the topology construction layer with a position (Step **1** in Fig. 4). This position is used to compute a set of neighbors on a round-based basis (Step **1'**). During the first round, the polystyrene layer also initializes the backup mechanism: Each node copies its guest data points onto K other nodes (where K is a system parameter) (Steps **2**). Copied data points are stored as ghost data points in the receiving nodes (Step **2'**).

After each round of the topology construction algorithm, Polystyrene executes the remaining steps: Each node uses the failure detector (FD) to check if any of the nodes that have copied data points to it has failed. If yes, the corresponding ghost data points are reactivated, and become guests (Step **3**). Conversely, each node checks if all its backup nodes are still alive, and if they are not, pushes its guests to new backup nodes. Finally each node uses the neighborhood returned by the topology construction layer (Step **1'**) to exchange data points with its neighbors (Step **4**). This last step is very similar to a decentralized k-means algorithm [22], and is what allows Polystyrene to re-converge towards the desired shape. In the following we revisit each of these steps in more detail.

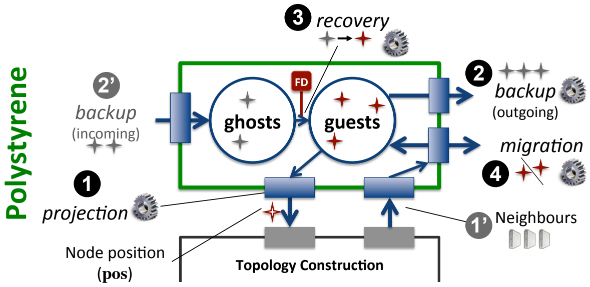


Figure 4: Polystyrene's key mechanisms

C. Projection

The projection step (Step **1**) determines the node's position that is perceived by the underlying topology construction protocol. It therefore directly influences the set of neighbors (Step **1'**) the polystyrene layer can use for its migration step (Step **4**), and should reflect the membership of the guest data points held by the node. One straightforward choice is the centroid of guest points in a vector space. This however does not work in modular spaces (such as in a logical torus we consider in our evaluation in Sec. IV), in which division is ill defined[2]. To include such spaces, we use instead the medoid of the guest points as the position of a node p, i.e. the guest point that minimizes the sum of square distances to other guest points:

$$p.\mathbf{pos} = \text{medoid}(p.\text{guests}) = x_0 \in p.\text{guests such that:}$$
$$\sum_{\mathbf{x} \in p.\text{guests}} d(x_0, x)^2 \text{ is minimum}$$

D. Backup

In order to preserve a memory of the overall *shape* of the system, nodes need to replicate the guest data points they hold, to ensure these points survive the crash of the holding node with a high probability. This is the role of the *backup* mechanism (Steps **2** and **2'** in Fig. 4, and Algorithm 1): After each round of the protocol, each node has a copy of its data points in K other nodes (line 4 in Algorithm 1). K is a system parameter reflecting how often individual data points get replicated. In particular when the system starts, each node only holds one data point (its original position), and each data point then gets pushed as a ghost point to K other nodes.

The K backup nodes to which a node p pushed its guest points remain the same across rounds, and are stored in the backups variable. When nodes fail (lines 1 and 2), new backup nodes are added as needed to keep the number of copies equal to K. Because the backup nodes of a node p remain stable, Algorithm 1 could be further improved by sending only

[2] For instance the equation $4 \equiv 2 \times x \pmod{16}$ accepts two solutions (2 and 10) making $4 \div 2 \pmod{16}$ ill defined.

Algorithm 1 Backup mechanism executed by p

```
1: backups $\leftarrow$ backups $\setminus$ failed
2: backups $\leftarrow$ backups $\cup \left\{ (K - |\text{backups}|) \text{ random nodes} \right\}$
3: for each $b \in$ backups do
4:     $b$.ghosts[$p$] $\leftarrow$ guests                ▷push operation
5: end for
```

incremental deltas to backup nodes, rather than full copies, thus reducing traffic once the system has converged.

Reciprocally, a node receiving incoming backups in its ghost set of points (Step **2'** in Fig. 4) keeps track of these nodes' provenance in the dictionary ghosts$[\cdot]$ (used at line 4 in Algorithm 1).

The level of resilience provided by this backup mechanism depends on K but also on the proportion of nodes expected to fail simultaneously. More precisely a data point will survive a failure if either its primary holder (the node holding the data point in its guests set), or one of its backup nodes (holding the data points in its ghosts dictionary) survives the failure. Because we assume catastrophic correlated failures, we spread copies as randomly as possible in the system, and choose backup nodes randomly (line 2 in Algorithm 1), using the underlying peer-sampling layer (RPS [17] in our case). There is however a downside to this strategy: In case of a localized failure, data points will take longer to percolate back to an appropriate holding node. As a result, other more localized strategies (e.g. replicating data points to nodes only a few hops away) could be considered, depending on the extent and level of correlation expected for failures.

If backup nodes are chosen so that they fail independently, K can be chosen to provide a target probability of survival, based on the proportion of nodes p_f expected to fail simultaneously. For instance, if we assume $p_f = 0.5$ (half of the nodes expected to disappear simultaneously), a probability of survival of $p_s = 99\%$ for individual data points would require:

$$\begin{array}{rcl} 1 - (p_f)^{K+1} & > & p_s \\ K & > & log(1 - p_s)/log(p_f) - 1 = 5.64 \end{array}$$

i.e. a replication factor K of at least 6.

E. Recovery

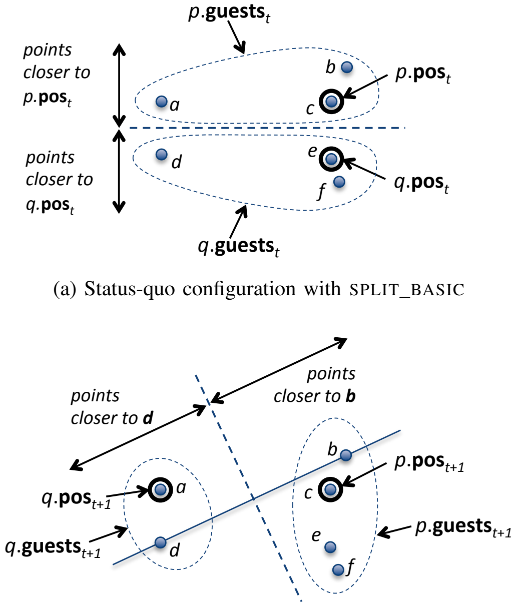
The recovery mechanism is the counterpart of the backup one: when p detects that one of the nodes whose state has been pushed to it has failed, p reinjects the data points in its current guests profile (Step **3**). Recovery insures data points are very likely to survive the demise of their holding nodes. However, as a result, surviving nodes might end up with active guest points far from each other, a situation that is problematic to converge back to the target shape, and for applications in which data points represent content that need to be routed to (as in DHT [3], [8] or storage systems [23]). The recovery step is therefore complemented by a final data migration step that re-balances data points after a failure.

F. Data points migration

The migration of data points (Step **4** in Fig. 4) is shown in Algorithms 3. The neighbor q is selected among p's local T-Man view (of size ψ, here 5), plus one random node returned by the peer sampling layer. This migration is conditioned by the SPLIT function, which distributes data points between p and q. The interaction of Algorithm 3 is a pair-wise exchange, meaning that q should not be interacting with anyone else than p while the exchange occurs. This is a common requirement of gossip-based aggregation protocols [24].

Algorithm 2 Recovery mechanism executed by p

```
for each $q \in keys(\text{ghosts}) \cap$ failed do
    guests $\leftarrow$ guests $\cup$ ghosts[$q$]               ▷recovery
    delete entry $q$ from ghosts
end for
```



(a) Status-quo configuration with SPLIT_BASIC

(b) Improved distribution with SPLIT_ADVANCED

Figure 5: Improving on the basic migration process

The SPLIT function, which distributes data points between the two interacting nodes, can be implemented in a number of ways. One first basic strategy is simply to allocate data points to the closest of the two nodes involved in the exchange (Function SPLIT_BASIC in Algorithm 4), thus implementing a simple form of distributed k-means clustering [22]. This approach can however lead to sub-optimal partitions between q and p, as illustrated in Fig. 5. Here, nodes q and p contain the guests data points $\{a, b, c\}$, and $\{d, e, f\}$ respectively (Fig. 5a). c has been selected as $p.\mathbf{pos}$, and e as $q.\mathbf{pos}$ in round t (noted with a subscript letter). Applying SPLIT_BASIC of Algorithm 4 to this configuration leads to a *status quo*: p and q do not exchange any point. This is unfortunate, as most objective clustering function (the function measuring how good a partition is) would consider the partition of Fig. 5a as sub-optimal. If we simply take the sum of square distances within each partition as our criteria[3]

$$\sum_{\substack{i,j \in \\ p.\text{guests}}} d(i,j)^2 + \sum_{\substack{k,l \in \\ q.\text{guests}}} d(k,l)^2$$

[3] Remember that because scalar division is not necessarily well defined in our space, we cannot readily use the notions of means or variance, e.g. as in Ward's traditional approach to clustering [25].

Algorithm 3 Periodic data points migration (executed by p)

```
$C \leftarrow \psi$ closest neighbors in local T-Man view
$C \leftarrow C \cup \{$ one random neighbor from RPS $\}$
$q \leftarrow$ random node from C
    ▷ Pair-wise pull-push exchange with q
$all\_points \leftarrow p$.guests $\cup$ $q$.guests                ▷pull exchange
$(points_1, points_2) \leftarrow \text{SPLIT}(all\_points, p.\mathbf{pos}, q.\mathbf{pos})$
$p$.guests $\leftarrow points_1$                         ▷updating one's state
$q$.guests $\leftarrow points_2$                               ▷push exchange
```

Algorithm 4 Basic approach to migration

```
function SPLIT_BASIC($points$,$\mathbf{pos}_p$,$\mathbf{pos}_q$)
    $points_p \leftarrow \{\mathbf{x} \in \text{points} : d(\mathbf{x}, \mathbf{pos}_p) < d(\mathbf{x}, \mathbf{pos}_q)\}$
    $points_q \leftarrow \{\mathbf{x} \in \text{points} : d(\mathbf{x}, \mathbf{pos}_q) \leq d(\mathbf{x}, \mathbf{pos}_p)\}$
    return $(points_p, points_q)$
end function
```

we see that that $\{a, d\}$ and $\{b, c, e, f\}$ would better distribute the set of data points between q and p.

For this reason, we use an alternative splitting strategy (SPLIT_ADVANCED in Algorithm 5), which combines two simple heuristics (labeled *PD* and *MD*). The first heuristic (*PD*, lines 2-4) partitions the set of data points according to one of its *diameters*, i.e. a pair of point (u, v) so that

$$d(u, v) = \max_{\substack{x,y\in \\ p.\text{guests}\cup q.\text{guests}}} d(x, y)$$

In Fig. 5b, this diameter is (d, b). In case p.guests $\cup$ q.guests contains many points (say over 30), we can approximate a diameter by taking a sample of pairs, or use bucketing techniques to minimize the cost of computation. We then distribute data points according to whether they are closer to u or v (lines 3 and 4). In Fig. 5, this results in two partitions: $\{a, d\}$ and $\{b, c, e, f\}$

The second heuristic (*MD*, lines 5 and 13) allocates each cluster to either p or q so as to minimize the movement each node will take as a result of the reallocation. We return to these two heuristics in our evaluation (Sec. IV-C) when we evaluate their impact on the behavior of Polystyrene.

Algorithm 5 Improved data point migration

```
 1: function SPLIT_ADVANCED($points$,$\mathbf{pos}_p$,$\mathbf{pos}_q$)
        ▷ Partitioning points along a Diameter (PD)
 2:     $(u, v) \in points^2$ such that $d(u, v) = \max_{x,y \in points} d(x, y)$
 3:     $points_u \leftarrow \{\mathbf{x} \in \text{points} : d(\mathbf{x}, u) < d(\mathbf{x}, v)\}$
 4:     $points_v \leftarrow \{\mathbf{x} \in \text{points} : d(\mathbf{x}, v) \leq d(\mathbf{x}, u)\}$
        ▷ Minimizing the Displacement of p and q (MD)
 5:     $m_u \leftarrow \text{medoid}(points_u)$
 6:     $m_v \leftarrow \text{medoid}(points_v)$
 7:     $\delta_{u,v} \leftarrow d(m_u, \mathbf{pos}_p) + d(m_v, \mathbf{pos}_q)$
 8:     $\delta_{v,u} \leftarrow d(m_v, \mathbf{pos}_p) + d(m_u, \mathbf{pos}_q)$
 9:     if $\delta_{u,v} < \delta_{v,u}$ then
10:         return $(points_u, points_v)$
11:     else
12:         return $(points_v, points_u)$
13:     end if
14: end function
```

IV. Evaluation

We evaluate three main aspects of Polystyrene: First, we assess Polystyrene's ability to converge to a target shape (in our evaluation a torus), and to repair this shape under catastrophic failures. This behavior is in contrast to traditional topology construction algorithms that, although they heal, cannot help but lose the original topology. Second, we evaluate how Polystyrene behaves when additional resources become available, and a large number of new nodes can be injected into the system. This could for instance occur when re-provisioning nodes from a larger pool of resources in a cloud or peer-to-peer environment. In both cases, we contrast Polystyrene's behavior against that of standard T-Man, a typical decentralized topology construction protocol, both in terms of quality of the resulting topology, and communication and storage overheads. Finally, we investigate the scalability of Polystyrene, and explore how its time to convergence evolves under various network sizes and different split functions.

A. Experimental setting and metrics

Unless stated otherwise, we use a logical torus made of 3200 nodes placed on a regular 80 × 40 grid for our experiments. The distance between two neighboring nodes on the grid is set to 1. Initially, each physical node is allocated a single data point. We configure Polystyrene to run above T-Man, which runs over RPS. In T-Man, each physical node is initialized with 10 random neighbors taken from the RPS layer. T-Man views are capped to 100 peers (rather than being unbounded as in [1]). We use otherwise typical values for the number of profiles per message ($m = 20$) and the peer sampling parameter ($\psi = 5$), taken from the original paper [1]. We set Polystyrene to use either 2, 4 or 8 back-up copies per data point, yielding an 87.5%, 96.9% or 99.8% probability of survival for data points, respectively. In all figures, we represent the 4 closest nodes returned by T-Man.

We use the following evaluation scenario, organized in three phases:

- **Phase 1: Convergence** ($r \in [0, 20[$)**:** We first let the topology converge based on the T-Man layer, while Polystyrene starts replicating data points (Algorithm 1) and monitoring nodes (Algorithm 2).
- **Phase 2: Failure** ($r \in [20, 100[$)**:** At round 20, we simulate a correlated catastrophic failure, identical to the one in Fig. 1, in which all the 1600 nodes located in one half of the torus crash. We then observe how the system converges back to a torus during the next 80 rounds.
- **Phase 3: Reinjection** ($r \in [100, 200[$)**:** At round 100, we re-inject 1600 fresh nodes, containing no data point, but with their $\mathbf{pos}$ parameters initialized. These new nodes are positioned uniformly on the torus, on a grid parallel to the original one.

For comparison purposes, we run the above scenario in two configurations: *first* with Polystyrene running atop T-Man (termed *Polystyrene*); and *second* with T-Man alone (termed *T-Man*). We evaluate both configurations along five metrics: (i) *proximity* captures the quality of the local neighborhoods constructed by the topology construction algorithm (how regular the grid is); (ii) *homogeneity* captures the quality of the shape overall (how well it resembles a torus); (iii) the *reshaping time* measures the time required to converge back to the original shape; (iv) the *average number of data points per node* accounts for the memory overhead; and (v) the *message cost* reflects the communication costs introduced by our approach.

Proximity is the main metric used in the original T-Man paper. It is the mean distance between a node and its k closest neighbors (with $k = 4$ in our case). Lower values denote better neighborhoods.

Homogeneity measures how well the original shape is conserved. It is defined as the mean distance between each initial data point and the nearest node hosting this data point as a guest (or the nearest node in the whole network if the data point has been lost with the failure), averaged over all data points.

$$homogeneity = \mathop{\mathbb{E}}_{\mathbf{x}\in\text{datapoints}}\left(\min_{n\in\hat{\text{guests}}^{-1}(\mathbf{x})}\{d(\mathbf{x}, n.\mathbf{pos})\}\right)$$

where $\hat{\text{guests}}^{-1}(\mathbf{x})$ represents the primary holders of the data point $\mathbf{x}$—i.e. the inverse image of $\text{guest}(n) = n.\text{guests}$, $\text{guests}^{-1}(\mathbf{x}) = \{n \in \text{nodes} : \mathbf{x} \in n.\text{guests}\}$—except when this inverse image is empty, in which case all nodes are returned:

$$\hat{\text{guests}}^{-1}(\mathbf{x}) = \begin{cases} \text{guests}^{-1}(\mathbf{x}) & \text{if } \text{guests}^{-1}(\mathbf{x}) \neq \emptyset \\ \text{nodes} & \text{otherwise} \end{cases}$$

When T-Man is used alone, we simply consider that a node's position is the single data point contained by this node ($n.\text{guests} = \{n.\mathbf{pos}\}$), and use the same definitions as above. Lower values denote a better shape.

Based on the homogeneity we define the *reshaping time*, i.e. the duration it takes for Polystyrene to converge back to a homogeneous shape after a major perturbation. To define that time, we use a *reference homogeneity value*, noted $\mathcal{H}$. $\mathcal{H}$ is a rough estimate of the maximum homogeneity one can expect in an ideally homogeneous distribution of $|N|$ nodes and $|P|$ data points over a 2D surface of area $\mathcal{A}$. In this case, each node $n \in N$ can be considered to be in charge of a zone of area $\frac{\mathcal{A}}{|N|}$ and diameter $\sim \sqrt{\frac{\mathcal{A}}{|N|}}$. As a result, if the distribution of both nodes and data points are ideal, each data point $p \in P$ can be assumed to be at most within a distance of $\mathcal{H}_{\mathcal{A}}^{|N|} = \frac{1}{2}\sqrt{\frac{\mathcal{A}}{|N|}}$ of their closest node:

$$homogeneity_{\text{id}}(\mathcal{A}, |N|) \leq \mathcal{H}_{\mathcal{A}}^{|N|} = \frac{1}{2}\sqrt{\frac{\mathcal{A}}{|N|}}$$

We say that our topology has been *successfully reshaped* when the measured homogeneity becomes less that $\mathcal{H}_{\mathcal{A}}^{|N|}$. The number of rounds necessary to reach this stage is the *reshaping time*. In our scenario, the grid of our 40×80 torus has a step of 1, and thus an area of $\mathcal{A}_{40\times 80} = 3200$. Before the failure, and after the reinjection, each of the 3200 nodes handles on average an area of 1, resulting in a reference homogeneity of $\mathcal{H}_{40\times 80}^{3200} = \frac{1}{2}$. After the failure, $N = 1600$ nodes survive, yielding a reference homogeneity of $\mathcal{H}_{40\times 80}^{1600} = \frac{1}{2}\sqrt{\frac{3200}{1600}} = \frac{\sqrt{2}}{2}$ ≈ 0.71.

The *average number of data points per node* measures the local memory overhead of our solution. We count both *guests* and *ghosts* data points, and consider that T-Man only has one guest data point (the node's position), and no ghosts. Without failure, the expected number of data points per node is $1 + K$: each node hosts one data point that is replicated K times.

Table II: Reshaping time and reliability, 40×80 torus, averaged on 25 experiments, confidence interval at 95%

K	**Reshaping time (rounds)**	**Reliability (%)**
2	5.00 ± 0.000	87.73 ± 0.18
4	6.96 ± 0.083	96.88 ± 0.10
8	9.08 ± 0.114	99.80 ± 0.03

Finally, we measure the *message cost* of Polystyrene and T-Man per round and per node. We assume a single coordinate uses the same size as a node ID, and take this as our arbitrary communication unit. Under these assumptions, sending a node descriptor (its ID, plus its coordinates) counts as 3 units, while a set of 2D coordinates counts as 2. In a first approximation, we ignore overheads caused by the underlying communication network (e.g. headers, checksums), and do not include the peer sampling protocol in our measurements.

B. Results

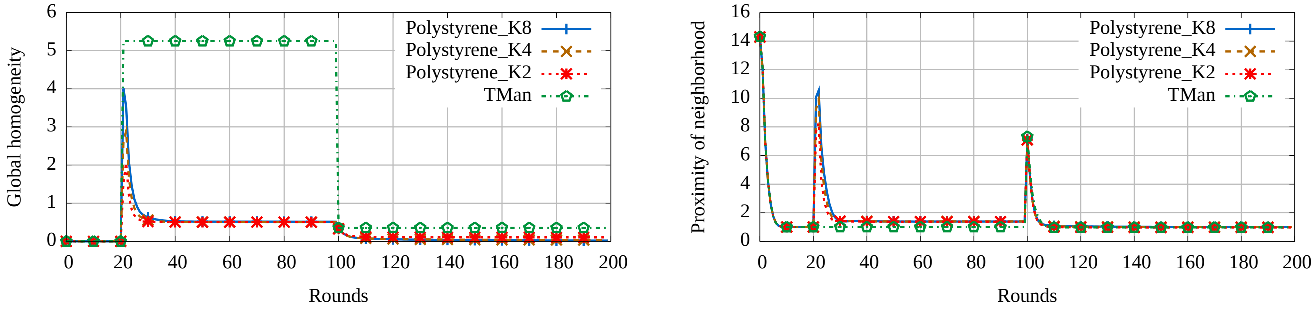
The results for the scenario we have described are shown in Figures 6, 7, and Table II. All results were computed with PeerSim [26]. The code of our simulation is freely available on-line[4]. Results are averaged over 25 experiments, and when mentioned, intervals of confidence are computed at a 95% confidence level. All results presented in this section use the SPLIT_ADVANCED function for migration (Sec. III-F).

Fig. 6a shows that Polystyrene clearly outperforms T-Man and can reform the overall torus after the catastrophic failure, with an homogeneity converging below the reference homogeneity $\mathcal{H}_{40\times 80}^{1600} \approx 0.71$ in less than 10 rounds for all values of K (e.g. $homogeneity = 0.61 \pm 2.9 \times 10^{-3}$ in round 28 for $K = 4$, illustrated in Fig. 8a). T-Man is unable to perform such repair (*homogeneity* stable at 5.25 ± 0.0 after the failure, corresponding to Fig. 1c). Similarly, after nodes have been re-injected at round 100 (Phase 3), T-Man is unable to reconstruct a uniform shape (Fig. 9a). Its homogeneity remains at $0.35 \pm 2.34 \times 10^{-17}$ at round 199 (Fig. 6a). By contrast, Polystyrene returns to a homogeneity close to zero (10 times lower at $0.035 \pm 9.3 \times 10^{-4}$ in round 199 for $K = 4$, illustrated in Fig. 9b).

In terms of proximity (Fig. 6b), Polystyrene maintains neighborhoods that are almost as good as those of T-Man when half of the nodes are gone (proximity $= 1.50 \pm 0.01$ in round 28 for $K = 4$, vs. 1.005 ± 0.0). After the reinjection, Polystyrene is on par with T-Man with a proximity of 1.02 ± 4.74^{-3} (round 125, $K = 4$), vs. $0.97 \pm 9.35 \times 10^{-17}$ for T-Man.

The reshaping time of Polystyrene is short: only 6.96 ± 0.083 rounds for $K = 4$ (Table II). The convergence slows down with a higher replication factor (9.08 ± 0.114 for $K = 8$), as a higher number of redundant data points need to be deduplicated. Higher values of K provide however a better reliability, directly in line with our discussion in Sec. III-D, leading to a trade-off between speed and reliability.

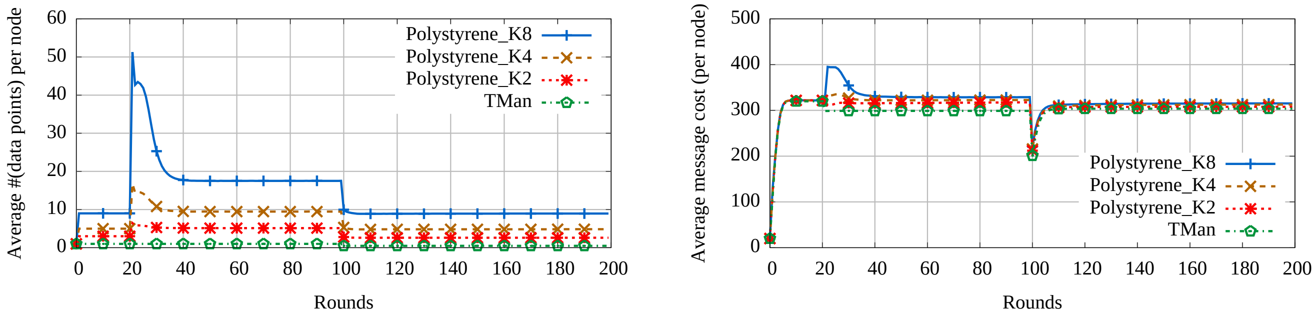
[4] http://ftaiani.ouvaton.org/7-software/index.html#polystyrene



(a) Homogeneity (the lower the better)

(b) Proximity (the lower the better)

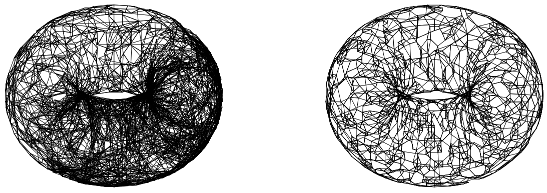
Figure 6: Polystyrene clearly outperforms T-Man in terms of homogeneity (a), while continuing to provide optimal neighborhoods (b) when half of the torus fails ($r = 20$), and nodes are re-injected ($r = 100$).



(a) Memory overhead (data points per node)

(b) Communication cost (1 ID = 1 coordinate = 1 unit)

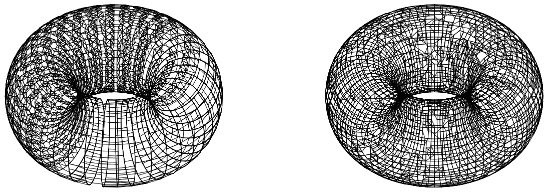
Figure 7: Once stabilized, the memory overhead of Polystyrene is directly linked to its level of replication (a), which in turn determines the system's reliability (Tab. II). In terms of communication, Polystyrene causes almost no additional cost over that of T-Man (b).



(a) Repair started ($r = 22$)

(b) Repair completed ($r = 28$)

Figure 8: Repair with Polystyrene (K=4)



(a) TMan

(b) Polystyrene

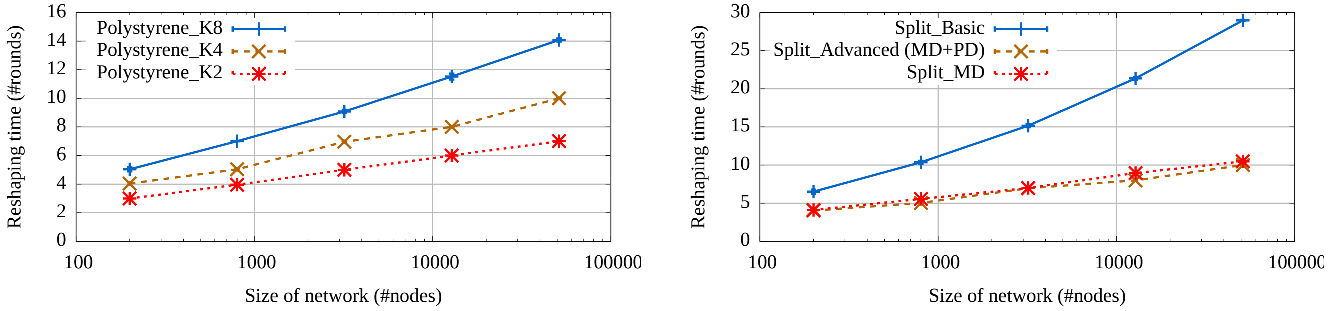
Figure 9: Effect of the reinjection at $r = 125$

The memory cost of Polystyrene (Fig. 7a) is directly linked to the level of replication in the system (K). Once stabilized, Polystyrene stores $|P| \times (K+1)$ copies of data points, where P is the set of unique data points present in the system. After the catastrophic failure only a few data points get lost, while the number of hosting nodes is halved, resulting in twice as many data points per nodes, as observed in Fig. 7a (17.73 ± 0.029 data point per node at round 40 for $K = 8$). The spike just after the failure (round 20) is caused by the eager backup behavior of Polystyrene (Sec. III-D): The ghosts data points that have just been reactivated are replicated by their new hosting nodes, leading to a high number of redundant copies. These copies rapidly disappear as the migration process (Sec. III-F) detects and removes them.

Finally, the message overhead (Fig. 7b) of Polystyrene is small: Most of the communication overhead (e.g. 93.6% for $K = 8$) is caused by T-Man. Because nodes move, T-Man must update their positions in its view in each round, causing most of the traffic. To this base cost, Polystyrene only adds the migration of data points, and back-ups (optimized to send incremental updates). Of course, these costs depend on the nature of data points, and larger objects (e.g. videos) would require additional heuristics (e.g. bloom filters) to minimize them. These are however out of the scope of this paper.

C. *Scalability and split function*

Polystyrene is particularly scalable: The reshaping time is close to logarithmic and as low as 14.08 ± 0.11 rounds for $51,200$ nodes and $K = 8$ (160×320 torus, Fig. 10a). This good performance is largely due to the heuristics introduced in SPLIT_ADVANCED: For $K = 4$ and $51,200$ nodes the



(a) $K\in\{2,4,8\}$, splitting with SPLIT_ADVANCED

(b) Impact of the split function, $K=4$

Figure 10: Polystyrene converges back very rapidly to a full torus (*reshaping time*) in a time that is almost logarithmic in the size of the network (a). The heuristics of the SPLIT function plays a key role in the protocol's performance (b).

use of the diameter heuristic (PD) alone more than halves the reshaping time ($\div 2.76$), while the combination of both heuristics brings in almost a threefold improvement ($\div 2.90$, converging in 10 rounds, Fig. 10b).

V. Related Work

The way we decouple data points from actual nodes in Polystyrene is reminiscent of the approach used in Distributed Hash Tables (DHT) and decentralized storage systems such as Chord [8], Pastry [16], Tapestry [27], CAN [3], or VoroNet [28] to allocate keys (rather than data points) to nodes. In these works, nodes are allocated zones from a key space and keys are allocated to nodes accordingly. Many of these systems provide some fault-tolerance or load-balancing [29], but very few consider the specific kind of correlated catastrophic failures that we address [30]. We review below those of these works that share the strongest commonalities with what we propose.

CAN [3] is a storage service using a d-torus. Peers own specific partitions (a.k.a. zones) of the space and store data items whose IDs fall in their zones. On join, a new node is allocated one part of a randomly split existing zone, with a possible localized load balancing mechanism between neighbors. This contrasts with our approach where nodes progressively migrate where the node density is lower, not randomly. Furthermore, CAN includes fault-tolerance features with respect to routing but none are designed against catastrophic failures, and lost content must be reinjected by the original holders.

Similarly, VoroNet organizes data points (termed *objects*) in an attribute space according to a Voronoi diagram.Our migration process is thus similar to VoroNet, but while we use it for load balancing and repair, VoroNet seeks to establish links between objects. Moreover, it relies on a Delaunay triangulation, a costly process. Finally, VoroNet does not include any particular fault-tolerant mechanism.

Meghdoot [6] is a peer-to-peer publish/subscribe system designed to support range queries over n attributes $A_1, A_2, ..., A_n$, by relying on a $2n$-dimensional CAN network. A subscription defines a range predicate $[l_i, h_i]$ over each attribute A_i, so it is represented by a *subscription point* of coordinates $\big((l_i, h_i), 1 \leq i \leq n\big)$ in the $2n$-dimensional CAN. To process an incoming subscription s at a CAN node, s is routed to the peer owning the zone where the corresponding subscription point lies. Meghdoot also provides a fault-tolerance mechanism relying on replication, similar to our work. By using the unexploited part of the CAN space (e.g. the triangle below the diagonal in a 2D CAN for one attribute), Meghdoot ensures that stored subscriptions are persistent even in the presence of failures: Whenever one of the nodes is faulty, its load is taken over by the node holding the symmetric point over the diagonal. Meghdoot does not however implement any migration mechanism to homogenize node density, and its replication factor is limited by the number n of attributes.

Finally, Glacier [30] is one of the few works that explicitly considers catastrophic correlated failures in decentralized storage. It assumes the existence of an underlying DHT providing a circular key space and its core mechanism consists in replicating fragments of each object using an erasure code, so that any object can be rebuilt from any r of its original n fragments. Contrary to our work, Glacier primarily considers catastrophic failures correlated in time (all nodes fail simultaneously), but not in space (all nodes from the same part of the overlay fail), and it stores the fragments of an object at periodically placed deterministic positions along the key ring. The mechanisms provided by Glacier are thus orthogonal to the services of the underlying overlay (as long as this overlay meets Glacier's requirements). In particular, Glacier does not consider the problem of reconstructing the overlay's topology as we do, and can therefore be seen as complementary to our work.

The particular migration process we have used provides a form of decentralized k-means clustering where, contrary to traditional k-means, none of the nodes has ever access to all data points. Instead each migration step between two nodes can be seen as one k-means step run in isolation. Other decentralized k-means clustering techniques have been proposed [22], [31], albeit usually not in the context of topology construction. For instance, the work in [22] explores an implementation using distributed agents, where each agent holds one single data point ("document"), and maintains three types of links (*unmatched*, *matched* and *connected*). Agents found in the same clusters act collectively to dynamically move links between these categories and thus converge to a set of clustered components. By contrast, the work proposed in [31] uses a peer-to-peer infrastructure in which each node holds multiple data points, as we do. However, data points do not move between nodes, but rather centroid information (along with the weight of associated clusters) is disseminated in the peer-to-peer network. Thus, the protocol does not influence the P2P topology as we do, and instead considers it a given.

VI. Conclusion

We have proposed a novel decentralized epidemic protocol that is able to maintain the general shape of a target topology under catastrophic correlated failures. As decentralized approaches get deployed in datacenters and cloud environments, such a capability appears increasingly important for several reasons. Clouds and datacenters can greatly benefit from placement decisions that are correlated with the underlying physical infrastructure (e.g. all the virtual machines handling contiguous keys hosted in the same rack). This correlation with the physical infrastructure brings, however, very real risks of systemic catastrophic failures, in which a large part of an overlay topology suddenly disappears. This is particularly likely to happen in multi-datacenter or multi-cloud scenarios and may significantly hamper the performance of such systems.

Our approach acts as an additional layer that can be added to any decentralized topology construction algorithm. It results from the combination of four basic mechanisms (projection, backup, recovery and migration), which together ensure that surviving nodes can adapt both to the loss and reinjection of resources in the system while maintaining the overall shape of the initial system topology. Our evaluation in particular shows that Polystyrene converges quickly back towards a homogeneous distribution of nodes on the target shape after losing as much as half of the system, while traditional topology constructions remain essentially static in such scenarios.

One aspect of our solution we would like to explore in the future is its high modularity. Any of its four components can be configured independently, and we aim to further investigate how they influence the protocol's convergence and its load-balancing properties. A second aspect we plan to investigate is the use of Polystyrene in cloud-based decentralized storage and recommendation solutions, which typically ignore correlated failures, and, we argue, are in need of such solutions.

Acknowledgment

This work has received a French government support granted to the CominLabs excellence laboratory (Project "DeSceNt: Plug-based Decentralized Social Network") and managed by the National Research Agency in the "Investing for the Future" program under reference Nb. ANR-10-LABX-07-01.

References

[1] M. Jelasity, A. Montresor, and O. Babaoglu, "T-man: Gossip-based fast overlay topology construction," *Comp. Netw.*, vol. 53, no. 13, 2009.

[2] S. Voulgaris and M. v. Steen, "Epidemic-style management of semantic overlays for content-based searching," in *Euro-Par'05*.

[3] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker, "A scalable content-addressable network," in *SIGCOMM'01*.

[4] B. Li, S. Xie, Y. Qu, G. Y. Keung, C. Lin, J. Liu, and X. Zhang, "Inside the new coolstreaming: Principles, measurements and performance implications," in *IEEE INFOCOM 2008*.

[5] D. Frey, R. Guerraoui, A.-M. Kermarrec, B. Koldehofe, M. Mogensen, M. Monod, and V. Quéma, "Heterogeneous gossip," in *Middleware'09*.

[6] A. Gupta, O. D. Sahin, D. Agrawal, and A. E. Abbadi, "Meghdoot: content-based publish/subscribe over p2p networks," in *Middleware'04*.

[7] A.-M. Kermarrec and P. Triantafillou, "Xl peer-to-peer pub/sub systems," *ACM Computing Surveys (CSUR)*, vol. 46, no. 2, 2013.

[8] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup service for internet applications," in *SIGCOMM'01*.

[9] A. Lakshman and P. Malik, "Cassandra: a decentralized structured storage system," *SIGOPS Oper. Syst. Rev.*, vol. 44, no. 2, 2010.

[10] M. Bertier, D. Frey, R. Guerraoui, A.-M. Kermarrec, and V. Leroy, "The gossple anonymous social network," in *Middleware'10*.

[11] X. Bai, M. Bertier, R. Guerraoui, A.-M. Kermarrec, and V. Leroy, "Gossiping personalized queries," in *EDBT '10*.

[12] J. Carretero, F. Isaila, A.-M. Kermarrec, F. Taiani, and J. Tirado, "Geology: Modular georecommendation in gossip-based social networks," in *ICDCS'2012*.

[13] A. Boutet, D. Frey, R. Guerraoui, A. Jégou, and A.-M. Kermarrec, "WhatsUp Decentralized Instant News Recommender," in *IPDPS 2013*.

[14] M. Chowdhury, M. Zaharia, J. Ma, M. I. Jordan, and I. Stoica, "Managing data transfers in computer clusters with orchestra," in *SIGCOMM '11*.

[15] M. Zaharia, D. Borthakur, J. Sen Sarma, K. Elmeleegy, S. Shenker, and I. Stoica, "Delay scheduling: A simple technique for achieving locality and fairness in cluster scheduling," in *EuroSys '10*.

[16] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems," in *Middleware'01*.

[17] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen, "Gossip-based peer sampling," *ACM TOCS*, vol. 25, 2007.

[18] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao, "Oceanstore: an architecture for global-scale persistent storage," *SIGPLAN Not.*, vol. 35, 2000.

[19] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: amazon's highly available key-value store," in *SOSP'07*.

[20] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié, "Peer-to-peer membership management for gossip-based protocols," *IEEE Trans. Comput.*, vol. 52, 2003.

[21] S. Voulgaris, D. Gavidia, and M. van Steen, "Cyclon: Inexpensive membership management for unstructured p2p overlays," *Journal of Network and Systems Management*, vol. 13, no. 2, pp. 197–217, 2005.

[22] E. Ogston, B. Overeinder, M. van Steen, and F. Brazier, "A method for decentralized clustering in large multi-agent systems," in *AAMAS'03*.

[23] A. Rowstron and P. Druschel, "Storage management and caching in past, a large-scale, persistent peer-to-peer storage utility," in *SOSP'01*.

[24] M. Jelasity, A. Montresor, and O. Babaoglu, "Gossip-based aggregation in large dynamic networks," *ACM TOCS*, vol. 23, no. 3, 2005.

[25] J. H. Ward, "Hierarchical grouping to optimize an objective function," *J. American Stat. Ass.*, vol. 58, no. 301, 1963.

[26] A. Montresor and M. Jelasity, "Peersim: A scalable p2p simulator," in *P2P'09*.

[27] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph, "Tapestry: An infrastructure for fault-tolerant wide-area location and routing," EECS Dept., U. California, Berkeley, Tech. Rep. UCB/CSD-01-1141, 2001.

[28] O. Beaumont, A.-M. Kermarrec, L. Marchal, and É. Rivière, "Voronet: A scalable object network based on voronoi tessellations," in *IPDPS'07*.

[29] J. Ledlie and M. Seltzer, "Distributed, secure load balancing with skew, heterogeneity and churn," in *INFOCOM 2005*.

[30] A. Haeberlen, A. Mislove, and P. Druschel, "Glacier: highly durable, decentralized storage despite massive correlated failures," in *NSDI'05*.

[31] S. Datta, C. Giannella, and H. Kargupta, "Approximate distributed k-means clustering over a peer-to-peer network," *IEEE Trans. on Knowl. and Data Eng.*, vol. 21, no. 10, 2009.