Fluidify: Decentralized Overlay Deployment in a Multi-Cloud World

A. C. Resmi[1] and François Taiani[1,2]

[1] Université de Rennes 1 - IRISA, [2] ESIR, Rennes, France
{rariyatt, francois.taiani}@irisa.fr

Abstract. As overlays get deployed in large, heterogeneous systems-of-systems with stringent performance constraints, their logical topology must exploit the locality present in the underlying physical network. In this paper, we propose a novel decentralized mechanism—Fluidify—for deploying an overlay network on top of a physical infrastructure while maximizing network locality. Fluidify uses a dual strategy that exploits both the logical links of an overlay and the physical topology of its underlying network. Simulation results show that in a network of 25,600 nodes, Fluidify is able to produce an overlay with links that are on average 94% shorter than that produced by a standard decentralized approach based on slicing, while demonstrating a sub-linear time complexity.

1 Introduction

Overlays are increasingly used as a fundamental building block of modern distributed systems, with numerous applications [5, 8, 11, 13, 15, 22, 25]. Unfortunately, many popular overlay construction protocols [1, 10, 27] do not usually take into account the underlying network infrastructure on which an overlay is deployed, and those that do tend to be limited to a narrow family of applications or overlays [29,30]. This is particularly true of systems running in multiple clouds, in which latency may vary greatly, and ignoring this heterogeneity can have stark implications in terms of performance and latency.

In the past, several works have sought to take into account the topology of the underlying infrastructure to realise network-aware overlays [21,28–30]. However, most of the proposed solutions are service-specific and they do not translate easily to other overlays. To address this lack, we propose a novel decentralized mechanism—called Fluidify—that seeks to maximize network locality when deploying an overlay network. Fluidify uses a dual strategy that exploits both the logical links of an overlay and the physical topology of its underlying infrastructure to progressively align one with the other. Our approach is fully decentralized and does not assume any global knowledge or central form of co-ordination.

The resulting protocol is generic, efficient, and scalable. Simulation results show that in a network of 25,600 nodes, Fluidify is able to produce an overlay with links that are on average 94% shorter than those produced by a standard decentralized approach based on slicing, while converging to a stable configuration in a time that is sub-linear ($\approx O(n^{0.6})$) in the size of the system.

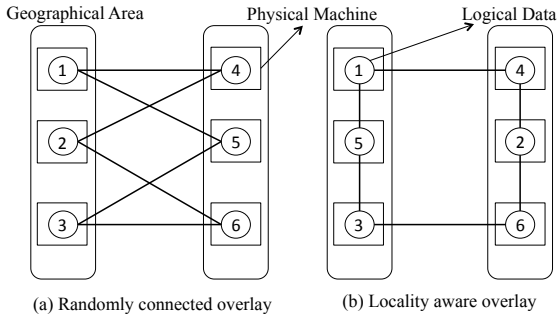


Fig. 1. Illustration of a randomly connected overlay and a network-aware overlay

The remainder of the paper is organized as follows. We first present the problem we address and our intuition (Sec. 2). We then present our algorithm (Sec. 3), and its evaluation (Sec. 4). We finally discuss related work (Sec. 5), and conclude (Sec. 6).

2 Background, Problem, and Intuition

Overlay networks organize peers in logical topologies on top of an existing network to extend its capabilities, with application to storage [22,25], routing [8,11], recommendation [1,27], and streaming [5,15]. Although overlays were originally proposed in the context of peer-to-peer (P2P) systems, their application today encompasses wireless sensor networks [7] and cloud computing [3,13].

2.1 The problem: building network-aware overlays

One of the challenges when using overlays, in particular structured ones, is to maintain desirable properties within the topology, in spite of failures, churn, and request for horizontal scaling. This challenge can be addressed through decentralized topology construction protocols [10,14,17,27], which are scalable and highly flexible. Unfortunately, such topology construction solutions are not usually designed to take into account the infrastructure on which an overlay is deployed. This brings clear advantages in terms of fault-tolerance, but is problematic from a performance perspective, as overlay links may in fact connect hosts that are far away in the physical topology. This is particularly likely to happen in heterogeneous systems, such as multi-cloud deployment, in which latency values might vary greatly depending on the location of individual nodes.

For instance, Fig. 1(a) depicts a randomly connected overlay deployed over two cloud providers (rounded rectangles). All overlay links cross the two providers, which is highly inefficient. By contrast, in Fig. 1(b), the same logical overlay only uses two distant links, and thus minimizes latency and network costs.

This problem has been explored in the past [20, 21, 28–30], but most of the proposed solutions are either tied to a particular service or topology, or limited

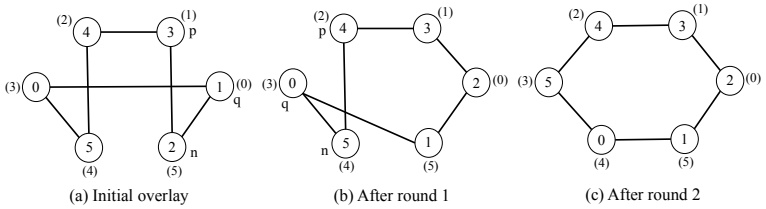


Fig. 2. Example of basic Fluidify approach on a system with n=6 and d=2

to unstructured overlays, and therefore cannot translate to the type of systems we have just mentioned, which is exactly where the work we present comes in.

2.2 Our intuition: a dual approach

Our proposal, Fluidify, uses a dual strategy that exploits both an overlay's logical links and its physical topology to incrementally optimize its deployment.

We model a deployed overlay as follows: each node possesses a physical index, representing the physical machine on which it runs, and a logical index, representing its logical position in the overlay. Each node also has a physical and logical neighbourhood: the physical neighbors of a node are its d closest neighbors in the physical infrastructure, according to some distance function $\mathsf{d}_{\text{net}}()$ that captures the cost of communication between nodes. The logical neighbors of a node are the node's neighbors in the overlay being deployed. For simplicity's sake, we model the physical topology as an explicit undirected graph between nodes, with a fixed degree. We take d to be the fixed degree of the graph, and the distance function to be the number of hops in this topology.

Fig. 2(a) shows an initial configuration in which the overlay has been deployed without taking into account the underlying physical infrastructure. In this example, both the overlay (solid line) and the physical infrastructure (represented by the nodes' positions) are assumed to be rings. The two logical indices 0 and 1 are neighbors in the overlay, but are diametrically placed in the underlying infrastructure. By contrast Fig. 2(c) shows an optimal deployment in which the logical and physical links overlap.

Our intuition, in Fluidify, consists of exploiting both the logical and physical neighbors of individual nodes, in a manner inspired from epidemic protocols, to move from the configuration of Fig. 2(a) to that of Fig. 2(c). Our basic algorithm is organized in asynchronous rounds, and implements a greedy approach as follows: in each round, each node n randomly selects one of its *logical* neighbors (noted p), and considers the *physical* neighbor of p (noted q) that is closest to itself. n evaluates the overall benefit of exchanging its logical index with that of q. If positive, the exchange occurs (Fig. 2(b), and then Fig. 2(c)).

Being a greedy algorithm, this basic strategy carries the risk of ending in a local minimum (Fig.3). To mitigate such situations, we use simulated annealing (taking inspiration from recent works on epidemic slicing [19]), resulting in a decentralized protocol for the deployment of overlay networks that is generic, efficient, and scalable.

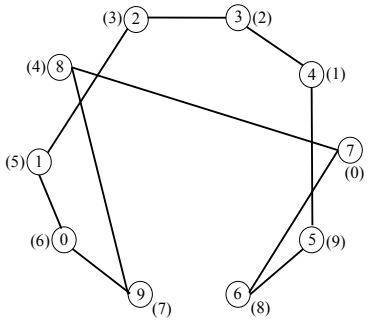


Fig. 3. Example of local minimum of a system with n=10 and d=2

3 The Fluidify algorithm

3.1 System model

We consider a set of nodes $N = \{n_1, n_2, .., n_N\}$ in a message passing system. Each node n possesses a physical (n.net) and a logical index (n.data). n.net represents the machine on which a node is deployed. n.data represents the role n plays in the overlay, e.g. a starting key in a Chord ring [17, 25].

Table 1 summarizes the notations we use. We model the physical infrastructure as an undirected graph $\mathcal{G}_{\text{net}} = (N, E_{\text{net}})$, and capture the proximity of nodes in this physical infrastructure through the distance function $\mathsf{d}_{\text{net}}()$. In a first approximation, we use the hop distance between two nodes in $\mathcal{G}_{\text{net}}$ for $\mathsf{d}_{\text{net}}()$, but any other distance would work. Similarly, we model the overlay being deployed as an undirected graph $\mathcal{G}_{\text{data}} = (N, E_{\text{data}})$ over the nodes N.

Our algorithms use the k-NN neighborhood of a node n in a graph $\mathcal{G}_x$, i.e. the k nodes closest to n in hop distance in $\mathcal{G}_x$, which we note as $\Gamma_x^k(n)$. We assume that these k-NN neighborhoods are maintained with the help of a topology construction protocol [1,10,27]. In the rest of the paper, we discuss and evaluate our approach independently of the topology construction used, to clearly isolate its workings and benefits. Under the above model, finding a good deployment of $\mathcal{G}_{\text{data}}$ onto $\mathcal{G}_{\text{net}}$ can be seen as a graph mapping problem, in which one seeks to optimize the cost function $\sum_{(n,m)\in E_{\text{data}}} \mathsf{d}_{\text{net}}(n, m)$.

3.2 Fluidify

The basic version of Fluidiy (termed Fluidify (basic)) directly implements the ideas discussed in Sec. 2.2 (Fig. 4): each node n first chooses a random *logical* neighbor (noted p, line 2), and then searches for the *physical* neighbor of p (noted q) that offers the best reduction in cost (argmin operator at line 3)[1]. The code shown slightly generalises the principles presented in Sec. 2, in that the nodes p and q are chosen beyond the 1-hop neighborhood of n and p (lines 2 and 3), and consider nodes that are k_{data} and k_{net} hops away, respectively.

[1] $\text{argmin}_{x\in S}\big(f(x)\big)$ returns one of the x in S that minimizes $f(x)$.

Table 1. Notations and Entities

n.net	physical index of node n
n.data	logical index of node n
d_{net}	distance function to calculate the distance between two nodes in physical space
$\mathcal{G}_{\text{net}}$	the physical graph (N, E_{net})
$\mathcal{G}_{\text{data}}$	the logical graph (N, E_{data})
$\Gamma^k_{\text{net}}(n)$	k closest nodes to n in $\mathcal{G}_{\text{net}}$, in hop distance
$\Gamma^k_{\text{data}}(n)$	k closest nodes to n in $\mathcal{G}_{\text{data}}$, in hop distance

Table 2. Parameters of Fluidify

k_{net}	size of the physical neighborhood explored by Fluidify
k_{data}	size of the logical neighborhood explored by Fluidify
K_0	initial threshold value for simulated annealing
$r_{\max}$	fade-off period for simulated annealing (# rounds)

```
1: In round(r) do
2:     p ← random node from Γ^{k_data}_data(n)
3:     q ← argmin_{u∈Γ^{k_net}_net(p)} Δ(n, u)
4:     conditional_swap(n, q, 0)
5: Procedure Δ(n, u)
6:     δ_n ← Σ_{(n,r)∈E_data} d_net(u, r) − Σ_{(n,r)∈E_data} d_net(n, r)
7:     δ_u ← Σ_{(u,r)∈E_data} d_net(n, r) − Σ_{(u,r)∈E_data} d_net(u, r)
8:     return δ_n + δ_u
9: Procedure conditional_swap(n, q, δ_lim)
10:    if Δ(n, q) < δ_lim then
11:        swap n.data and q.data
12:        swap Γ^{k_data}_data(n) and Γ^{k_data}_data(q)
13:    end if
```

Fig. 4. Fluidify (basic)

The potential cost reduction is computed by the procedure $\Delta(n, u)$ (lines 5-8), which returns the cost variation if n and u were to exchange their roles in the overlay. The decision whether to swap is made in $\mathsf{conditional\_swap}(n, q, \delta_{\text{lim}})$ (with $\delta_{\text{lim}} = 0$ in Fluidify Basic).

To mitigate the risk of local minimums, we extend it with simulated annealing [19], which allows two nodes to be swapped even if there is an increase in the cost function. We call the resulting protocol Fluidify (SA), shown in Figure 5. In this version, we swap nodes if the change in the cost function is less than a limit, $\Delta_{\text{limit}}(r)$, that gradually decreases to zero as the rounds progress (line 4). $\Delta_{\text{limit}}(r)$ is controlled by two parameters, K_0 which is the initial threshold value, and $r_{\max}$ which is the number of rounds in which it is decreased to 0. In the remainder of this paper, we use Fluidify to mean Fluidify (SA).

1: **In round(r) do**
2: $p \leftarrow$ random node from $\Gamma^{k_{\text{data}}}_{\text{data}}(n)$
3: $q \leftarrow \text{argmin}_{u \in \Gamma^{k_{\text{net}}}_{\text{net}}(p)} \Delta(n,u)$
4: conditional_swap$(n, q, \Delta_{\text{limit}}(r))$
5: **Procedure** $\Delta_{\text{limit}}(r)$
6: **return** $\max\left(0, K_0 \times (1 - r/r_{\max})\right)$

Fig. 5. Fluidify (SA)

4 Evaluation

4.1 Experimental Setting and Metrics

Unless otherwise indicated, we use rings for both infrastructure graph $\mathcal{G}_{\text{net}}$ and overlay graph $\mathcal{G}_{\text{data}}$. We assume that the system has converged when the system remains stable for 10 rounds.

The default simulation scenario is one in which the system consists of 3200 nodes, and use 16-NN logical and physical neighborhoods ($k_{\text{net}} = k_{\text{data}} = 16$) when selecting p and q. The initial threshold value for simulated annealing (K_0) is taken as $|N|$. $r_{\max}$ is taken as $|N|^{0.6}$ where 0.6 was chosen based on the analysis of the number of rounds Fluidify (basic) takes to converge.

We assess the protocols using two metrics:

- Proximity - captures the quality of the overlay constructed by the topology construction algorithm. Lower value denotes a better quality.
- Convergence time - measures the number of rounds taken by the system to converge.

Proximity is defined as the average network distance of logical links normalized by the diameter of the network graph $\mathcal{G}_{\text{net}}$:

$$proximity = \frac{\mathop{\mathbb{E}}_{(n,m)\in E_{\text{data}}} \mathsf{d}_{\text{net}}(n,m)}{\text{diameter}(\mathcal{G}_{\text{net}})} \tag{1}$$

where $\mathbb{E}$ represents the expectation operator, i.e. the mean of a value over a given domain, and *diameter*() returns the longest shortest path between pairs of vertices in a graph, i.e. its diameter. In a ring, it is equal to $N/2$.

4.2 Baselines

The performance of our approach is compared against three other approaches. One is Randomized (SA) (Fig. 6) where each node considers a set of random nodes from N for a possible swap. The other is inspired from epidemic slicing [9,19], and only considers the physical neighbors of a node n for a possible swap (Slicing (SA), in Figure. 8). The third approach is similar to PROP-G [20], and it only considers logical neighbours of a node n for a possible swap (PROP-G (SA),

1: **In round**(r) **do**
2: $S \leftarrow k_{\text{net}}$ random nodes from N
3: $q \leftarrow \text{argmin}_{u \in S}\ \Delta(n,u)$
4: conditional_swap$\big(n, q, \Delta_{\text{limit}}(r)\big)$

Fig. 6. Randomized (SA)

1: **In round**(r) **do**
2: $S \leftarrow \Gamma_{\text{data}}^{k_{\text{data}}}(n)$
3: $q \leftarrow \text{argmin}_{u \in S}\ \Delta(n,u)$
4: conditional_swap$(n, q, 0)$

Fig. 7. PROP-G

1: **In round**(r) **do**
2: $q \leftarrow \underset{u \in \Gamma_{\text{net}}^{k_{\text{net}}}(n)}{\text{argmin}}\ \Delta(n,u)$
3: conditional_swap$\big(n, q, \Delta_{\text{limit}}(r)\big)$

Fig. 8. Slicing (SA)

1: **In round**(r) **do**
2: $p \leftarrow$ random node from $\Gamma_{\text{data}}^{k_{\text{data}}}(n)$
3: $S \leftarrow \Gamma_{\text{net}}^{\frac{k_{\text{net}}}{2}}(p) \cup \Gamma_{\text{net}}^{\frac{k_{\text{net}}}{2}}(n)$
4: $q \leftarrow \text{argmin}_{u \in S}\ \Delta(n,u)$
5: conditional_swap$(n, q, 0)$

Fig. 9. Data-Net & Net

1: **In round**(r) **do**
2: $p \leftarrow$ random node from $\Gamma_{\text{data}}^{k_{\text{data}}}(n)$
3: $S \leftarrow \Gamma_{\text{net}}^{\frac{k_{\text{net}}}{2}}(p) \cup \Big\{ \frac{k_{\text{net}}}{2}$ rand. nodes $\in N \setminus \Gamma_{\text{net}}^{\frac{k_{\text{net}}}{2}}(p) \Big\}$
4: $q \leftarrow \text{argmin}_{u \in S}\ \Delta(n,u)$
5: conditional_swap$(n, q, 0)$

Fig. 10. Data-Net & R

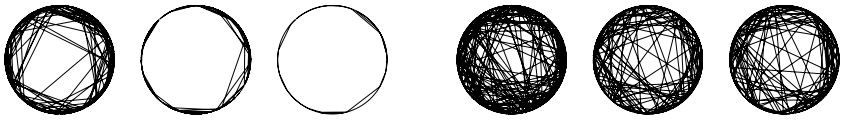
in Figure. 7). In all these approaches simulated annealing is used as indicated by (SA). The only difference between the above four approaches is the way in which the swap candidates are taken.

To provide further comparison points, we also experimented with some combinations of the above approaches. Fig. 9 (termed *Data-Net & Net*) is a combination of Fluidify (basic) with Slicing (SA). Fig. 10 (termed *Data-Net & R*) is a combination of Fluidify (basic) with Randomized (SA). We also tried a final variant, *combination-R*, in which once the system has converged using Fluidify (basic) (no more changes are detected for a pre-determined number of rounds), nodes look for random swap candidates like we did in Fig. 6.

4.3 Results

All the results (Figs. 11-18 and Tables 3-5) are computed with Peersim [18] and are averaged over 30 experiments. When shown, intervals of confidence are computed at a 95% confidence level using a student t-distribution.

Evaluation of Fluidify (SA) The results obtained by Fluidify (SA) and the three baselines on a ring/ring topology are given in Table 3 and charted in Figs. 12 and 13. In addition, Fig. 11 illustrates some of the rounds that Fluidify (SA) and Slicing (SA) perform. Fig. 12 shows that Fluidify clearly outper-



(a) Fluidify (SA) (b) Slicing (SA)

Fig. 11. Illustrating the convergence of Fluidify (SA) & Slicing (SA) on a ring/ring topology. The converged state is on the right. ($N = K_0 = 400$, $k_{\text{net}} = k_{\text{data}} = 16$)

Table 3. Performance of Fluidify against various baselines

Nodes	**Proximity(%)**				**Convergence (rounds)**			
	Fluid(SA)	Slicing(SA)	Rand(SA)	PROP-G(SA)	Fluid(SA)	Slicing(SA)	Rand(SA)	PROP-G(SA)
100	*4.06*	10.46	7.70	13.88	18.10	17.16	23.80	*17.03*
200	*2.70*	10.12	6.27	12.99	28.50	26.33	43.43	*25.13*
400	*1.71*	9.76	5.35	12.65	42.50	39.20	85.36	*38.06*
800	*1.26*	9.34	4.83	12.14	64.13	58.93	136.76	*57.16*
1,600	*0.86*	8.80	4.41	11.57	96.80	90.56	198.03	*85.13*
3,200	*0.69*	8.47	3.82	11.31	144.40	138.20	274.80	*128.14*
6,400	*0.51*	8.13	3.07	11.27	216.10	203.40	382.10	*198.24*
12,800	*0.46*	7.66	2.28	11.01	324.00	292.10	533.67	*263.32*
25,600	*0.43*	6.99	1.79	10.02	485.00	418.60	762.13	*392.81*

forms the other three approaches in terms of proximity over a wide range of network sizes.

Fig 13 charts the convergence time against network size in loglog scale for Fluidify and its competitors. Interestingly all approaches show *a polynomial convergence time*, which Randomized (SA) taking clearly longer than the others. This shows the scalability of Fluidify even for very large networks. If we turn to Tab. 3, it is evident that as the network size increases, the time taken for the system to converge also increases. Both Fluidify and Slicing (SA) converges around the same time with Slicing (SA) converging a bit faster than Fluidify. Randomized (SA) takes much longer (almost twice as many rounds). PROP-G converges faster compared than all other approaches.

The better convergence of PROP-G and Slicing (SA) can be explained by the fact that both approaches run out of interesting swap candidates more rapidly than Fluidify. It is important to note here that all approaches are calibrated to consider the same number of candidates per round. This suggests that PROP-G and Slicing (SA) runs out of potential swap candidates because they consider candidates of lesser quality, rather than considering more candidates faster.

Fig. 14 shows how the proximity varies with round for our default system settings. Initial avg. link distance was around $N/4$ where N is the network size and this is expected as the input graphs are randomly generated. So the initial proximity was approximately equal to 50%. Fluidify was able to bring down the proximity from 50% to 0.7%. A steep decrease in proximity was observed in initial rounds and later it decreases at a lower pace and finally settles to a proximity value of 0.7% as shown in Fig 14. Randomized version was also able

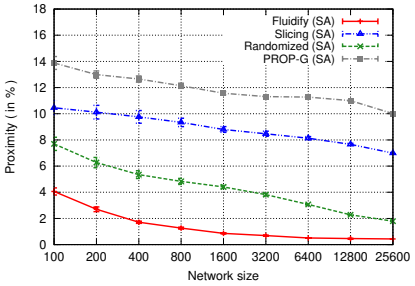


Fig. 12. Proximity. Lower is better. Fluidify (SA) clearly outperforms the baselines in terms of deployment quality.

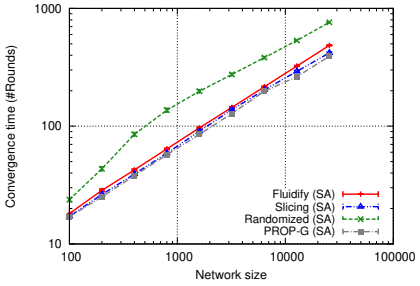


Fig. 13. Convergence time. All three approaches have a sublinear convergence ($\approx 1.237 \times |N|^{0.589}$ for Fluidify).

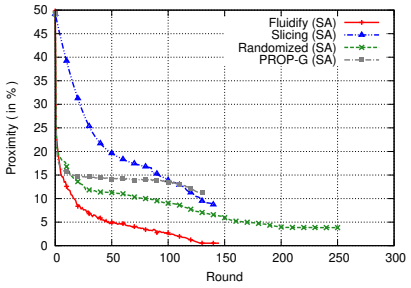


Fig. 14. Proximity over time ($N = K_0 = 3200$, $k_{\text{net}} = k_{\text{data}} = 16$). Fluidify (SA)'s optimization is more aggressive than those of the other baselines.

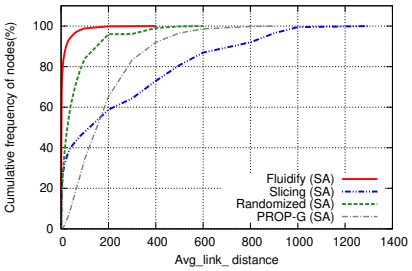
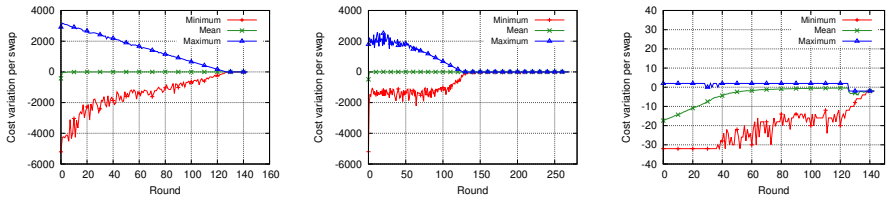


Fig. 15. Average link distances in converged state ($N = K_0 = 3200$, $k_{\text{net}} = k_{\text{data}} = 16$). Fluidify (SA)'s links are both shorter and more homogeneous.

to perform well in the initial stages but later on the gain in proximity decreases and converges to a proximity value of 3.8% after a very long time. Slicing (SA) is unable to get much gain in proximity from the start itself and converges to a proximity value of 8.4%. PROP-G performes really well in the initial rounds but after a while the performance become worse compared to Randomized (SA) and Fluidify (SA) Cumulative distribution of nodes based on the avg. link distance in a converged system for all the three aproaches is depicted in Fig. 15. It is interesting to see that nearly 83% of the nodes are having an average link distance less than 10 and 37% of the nodes were having an average link distance of 1 in the case of Fluidify. But for Slicing (SA) even after convergence, a lot of nodes are having an average link distance greater than 200. Slicing (SA) clearly fails in improving the system beyond a limit.

The maximum, minimum and the mean gain obtained per swap in a default system setting using Fluidify is shown in Fig. 16(a). As the simulation progresses the maximum, minimum and the mean value of the cost function per swap in each round starts getting closer and closer and finally becomes equal on convergence.



(a) Fluidify (SA) (b) Randomized (SA) (c) Slicing (SA)

Fig. 16. Variation of the cost function per swap over time. Lower is better. ($N = K_0 = 3200$, $k_{\text{net}} = k_{\text{data}} = 16$, note the different scales) Fluidify (SA) shows the highest amplitude of variations, and fully exploits simulated annealing, which is less the case for Randomized (SA), and not at all for slicing.

Maximum gain per swap (negative cost) is obtained in the initial rounds of the simulation. Maximum value obtained by the cost function is expected to gradually decrease from a value less than or equal to 3200, which is the initial threshold value for simulated annealing, to 0. From Fig. 16(b) it is clear that the variation of cost function for Randomized (SA) also shows a similar behaviour. Here the system progresses with a very small gain for a long period of time. As shown in Fig.16(c), the most interesting behaviour is that of Slicing (SA). It does not benefit much with the use of simulated annealing. The maximum gain that can be obtained per swap is 32 and the maximum negative gain is 2. This is because only the physically closer nodes of a given node are considered for a swap and the swap is done with the best possible candidate.

The *message cost* per round per node will be equal to the amount of data that a node exchanges with another node. In our approach the nodes exchange their logical index and the logical neighbourhood. We assume that each index value amounts to 1 unit of data. So the message cost will be $1+k_{\text{data}}$ which will be 17 in default case. The *communication overhead* in the network per cycle will be equal to the average number of swaps occurring per round times the amount of data exchanged per swap. A single message costs 17 units. So a swap will cost 34 units. In default setting, an average of 2819 swaps happen per round and this amounts to around 95846 units of data per round.

All the four approaches that we presented here are generic and can be used for any topologies. Table. 4 shows how the three approaches fares for various topologies in a default setting. Fluidify clearly out performs the other approaches.

Effects of variants Figure. 17 shows that compared to its variants like Fluidify (basic), combination-R, Data-Net & Net (Fig. 9), Data-Net & R (Fig. 10), Fluidify (SA) is far ahead in quality of convergence. Here also we consider a ring/ring topology with default setting. The convergence time taken by Fluidify is slightly higher compared to its variants as shown in Fig. 18.

Table 5 shows how varying the initial threshold value for Fluidify affects its performance. From the table it is clear that as the initial threshold value

Table 4. Performance on various topologies

Approach	Physical topology	Logical topology	Proximity(%)	Convergence(#Rounds)
Fluidify(SA)	torus	torus	2.4(±0.05)	162(±2.34)
Fluidify(SA)	torus	ring	2.6(±0.03)	171(±3.6)
Fluidify(SA)	ring	torus	1.8(±0.06)	156(±2.36)
Slicing(SA)	torus	torus	4.5(±0.05)	130(±2.16)
Slicing(SA)	torus	ring	5.2(±0.02)	128(±3.26)
Slicing(SA)	ring	torus	9.5(±0.08)	143(±4.1)
Randomized(SA)	torus	torus	3.82(±0.08)	423(±2.41)
Randomized(SA)	torus	ring	4.05(±0.04)	464(±3.28)
Randomized(SA)	ring	torus	2.7(±0.05)	442(±3.82)
PROP-G(SA)	torus	torus	4.6(±0.05)	132(±2.34)
PROP-G(SA)	torus	ring	5.6(±0.03)	130(±3.6)
PROP-G(SA)	ring	torus	10.1(±0.06)	128(±2.36)

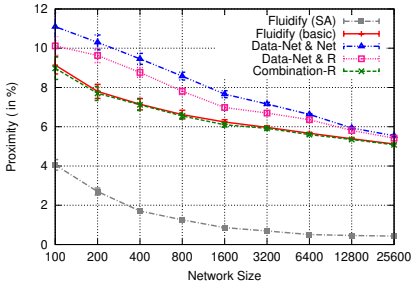


Fig. 17. Comparison of different variants of Fluidify - Proximity

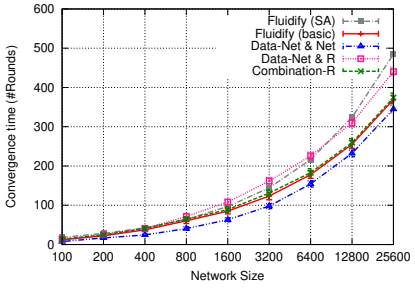


Fig. 18. Comparison of different variants of Fluidify - Convergence

Table 5. Impact of K_0 on Fluidify (SA)

K_0	**Proximity (%)**	**Convergence (rounds)**
320	2.4	156
640	1.6	145
1600	1.1	146
3200	0.7	144

increases the proximity that we obtain also become better and better. With a higher threshold value, more swaps will occur and therefore there is a higher chance of getting closer to the global minimum. The threshold value that gives the best performance is used for all our simulations.

5 Related Work

Fully decentralized systems are being extensively studied by many researchers. Many well known and widely used P2P systems are unstructured. However, there are several overlay networks in which the node locality is taken into account. Structured P2P overlays, such as CAN [22], Chord [25], Pastry [24], and

Tapestry [31], are designed to enhance the searching performance by giving some importance to node placement. But, as pointed out in [23], structured designs are likely to be less resilient, because it is hard to maintain the structure required for routing to function efficiently when hosts are joining and leaving at a high rate. Chord in its original design, does not consider network proximity at all. Some modification to CAN, Pastry, and Tapestry are made to provide locality to some extent. However, these results come at the expense of a significantly more expensive overlay maintenance protocol.

One of the general approaches used to bridge the gap between physical and overlay node proximity is landmark clustering. Ratnasamy et al. [21] use landmark clustering in an approach to build a topology-aware CAN [22] overlay network. Although the efficiency can be improved, this solution needs extra deployment of landmarks and produces some hotspots in the underlying network when the overlay is heterogeneous and large. Some [30] [29] have proposed methods to fine tune the landmark clustering for overlay creation. The main disadvantage with landmark system is that there needs to be a reliable infrastructure to offer these landmarks at high availability. Application layer multicast algorithms construct a special overlay network that exploits network proximity. The protocol they use are often based on a tree or mesh structure. Although they are highly efficient for small overlays, they are not scalable and creates hotspots in the network as a node failure can make the system unstable and difficult to recover. Later proximity neighbour selection [2] was tried to organise and maintain the overlay network which improved the routing speed and load balancing. Waldvogel and Rinaldi [12] [28] propose an overlay network(Mithos) that focuses on reducing routing table sizes. It is a bit expensive and only very small overlay networks are used for simulations and the impact of network digression is not considered.

Network aware overlays are used to increase the efficiency of network services like routing, resource allocation and data dissemination. Works like [16] and [4] combines the robustness of epidemics with the efficiency of structured approaches in order to improve the data dissemination capabilities of the system. Gossip protocols which are scalable and inherent to network dynamics can do efficient data dissemination. Frey et al. [5] uses gossip protocols to create a system where nodes dynamically adapt their contribution to the gossip dissemination according to the network characteristics like bandwidth and delay. Kermarrec et al. [6] use gossip protocols for renaming and sorting. Here nodes are given id values and numerical input values. Nodes exchange these input values so that in the end the input of rank k is located at the node with id k. Slicing method [19] [9] was made use of in resource allocation. Specific attributes of network(memory, bandwidth, computation power) are taken into account to partition the network into slices. Network aware overlays can be used in cloud infrastructure [26] to provide efficient data dissemination.

Most of the works on topology aware overlays are aimed at improving a particular service such as routing, resource allocation or data dissemination. What we are proposing is a generalized approach for overlay creation giving importance

to data placement in the system. It has higher scalability and robustness and less maintenance cost compared to other approaches. The simulated annealing and slicing approach is motivated mainly by the works [19], [6], [9]. But these works concentrated mainly on improving a single network service while we concentrate on a generalized solution that can significantly improve all the network services.

6 Conclusion and Future Work

In this paper, we presented Fluidify, a novel decentralized mechanism for overlay deployment. Fluidify works by exploiting both the logical links of an overlay and the physical topology of its underlying network to progressively align one with the other and thereby maximizing the network locality. The proposed approach can be used in combination with any topology construction algorithm. The resulting protocol is generic, efficient, scalable and can substantially improve network overheads and latency in overlay based-systems. Simulation results show that in a ring/ring network of 25,600 nodes, Fluidify is able to produce an overlay with links that are on average 94% shorter than that produced by a standard decentralized approach based on slicing.

One aspect we would like to explore in future is to deploy Fluidify in a real system and see how it fares. A thorough analytical study of the behaviour of our approach is also intended.

References

1. Bertier, M., Frey, D., Guerraoui, R., Kermarrec, A.M., Leroy, V.: The gossple anonymous social network. In: Middleware'10
2. Castro, M., Druschel, P., Hu, Y.C., Rowstron, A.: Topology-aware routing in structured peer-to-peer overlay networks. In: Future Directions in Distributed Computing. pp. 103–107. Springer-Verlag (2003)
3. DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., Vogels, W.: Dynamo: amazon's highly available key-value store. In: SOSP '07 (2007)
4. Doerr, B., Elsässer, R., Fraigniaud, P.: Epidemic algorithms and processes: From theory to applications. Dagstuhl Reports 3(1), 94–110 (2013)
5. Frey, D., Guerraoui, R., Kermarrec, A.M., Koldehofe, B., Mogensen, M., Monod, M., Quéma, V.: Heterogeneous gossip. In: Middleware. pp. 42–61 (2009)
6. Giakkoupis, G., Kermarrec, A.M., Woelfel, P.: Gossip protocols for renaming and sorting. In: DISC. pp. 194–208 (Oct 14–18 2013)
7. Grace, P., Hughes, D., Porter, B., Blair, G.S., Coulson, G., Taiani, F.: Experiences with open overlays: A middleware approach to network heterogeneity. In: Eurosys'08 (2008)
8. Gupta, A., Sahin, O.D., Agrawal, D., Abbadi, A.E.: Meghdoot: content-based publish/subscribe over p2p networks. In: Middleware'04
9. Jelasity, M., Kermarrec, A.M.: Ordered slicing of very large-scale overlay networks. In: P2P 2006 (2006)
10. Jelasity, M., Montresor, A., Babaoglu, O.: T-man: Gossip-based fast overlay topology construction. Comput. Netw. 53(13), 2321–2339 (Aug 2009)

11. Kermarrec, A.M., Triantafillou, P.: Xl peer-to-peer pub/sub systems. ACM Computing Surveys (CSUR) 46(2) (2013)
12. Krishnamurthy, B., Wang, J.: On network-aware clustering of web clients. In: SIGCOMM '00. pp. 97–110. ACM (2000)
13. Lakshman, A., Malik, P.: Cassandra: a decentralized structured storage system. SIGOPS Oper. Syst. Rev. 44(2) (2010)
14. Leitao, J., Pereira, J., Rodrigues, L.: Epidemic broadcast trees. In: SRDS'07
15. Li, B., Xie, S., Qu, Y., Keung, G.Y., Lin, C., Liu, J., Zhang, X.: Inside the new coolstreaming: Principles, measurements and performance implications. In: IEEE INFOCOM 2008
16. Matos, M., Schiavoni, V., Felber, P., Oliveira, R., Rivière, E.: Lightweight, efficient, robust epidemic dissemination. J. Parallel Distrib. Comput. 73(7), 987–999 (2013)
17. Montresor, A., Jelasity, M., Babaoglu, O.: Chord on demand. In: P2P'2005 (2005)
18. Montresor, A., Jelasity, M.: Peersim: A scalable p2p simulator. In: P2P 2009
19. Pasquet, M., Maia, F., Rivière, E., Schiavoni, V.: Autonomous multi-dimensional slicing for large-scale distributed systems. In: DAIS. pp. 141–155 (2014)
20. Qiu, T., Chen, G., Ye, M., Chan, E., Zhao, B.Y.: Towards location-aware topology in both unstructured and structured p2p systems. In: ICPP. p. 30. IEEE Computer Society (2007)
21. Ratnasamy, S., Handley, M., Karp, R., Shenker, S.: Topologically-aware overlay construction and server selection. In: INFOCOM'02. vol. 3, pp. 1190– 1199. IEEE (2002)
22. Ratnasamy, S., Francis, P., Handley, M., Karp, R., Shenker, S.: A scalable content-addressable network. SIGCOMM Comput. Commun. Rev. 31(4), 161–172 (2001)
23. Ratnasamy, S., Shenker, S.: Can heterogeneity make gnutella scalable? In: IPTPS'02 (2002)
24. Rowstron, A.I.T., Druschel, P.: Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: Middleware. pp. 329–350 (2001)
25. Stoica, I., Morris, R., Karger, D., Kaashoek, M.F., Balakrishnan, H.: Chord: A scalable peer-to-peer lookup service for internet applications. In: SIGCOMM '01. ACM (2001)
26. Tudoran, R., Costan, A., Wang, R., Bougé, L., Antoniu, G.: Bridging Data in the Clouds: An Environment-Aware System for Geographically Distributed Data Transfers. In: IEEE/ACM CCGrid. Chicago (May 2014)
27. Voulgaris, S., Steen, M.v.: Epidemic-style management of semantic overlays for content-based searching. In: Euro-Par'05
28. Waldvogel, M., Rinaldi, R.: Efficient topology-aware overlay network. In: SIGCOMM/CCR'03 (2003)
29. Xu, Z., Tang, C., Zhang, Z.: Building topology-aware overlays using global soft-state. In: ICDSC'03 (May 2003)
30. Zhang, X.Y., Zhang, Q., Zhang, Z., Song, G., Zhu, W.: A construction of locality-aware overlay network: moverlay and its performance. IEEE J.Sel. A. Commun. 22(1), 18–28 (Sep 2006)
31. Zhao, B., Kubiatowicz, J., Joseph, A.: Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Computer 74 (2001)