2017 IEEE 9th International Conference on Cloud Computing Technology and Science

Designing Overlay Networks for Decentralized Clouds

Genc Tato
IRISA, Inria, Rennes, France
genc.tato@inria.fr

Marin Bertier
IRISA, INSA, Rennes, France
marin.bertier@irisa.fr

Cédric Tedeschi
IRISA, Université de Rennes 1 / Inria
Rennes, France
cedric.tedeschi@inria.fr

***Abstract*—Recent increase in demand for next-to-source data processing and low-latency applications has shifted attention from the traditional centralized cloud to more distributed models such as edge computing. In order to fully leverage these models it is necessary to decentralize not only the computing resources but also their management. While a decentralized cloud has various inherent advantages, it also introduces different challenges with respect to coordination and collaboration between resources. A large-scale system with multiple administrative entities requires an overlay network which enables data and service localization based only on a partial view of the network. Numerous existing overlay networks target different properties but they are built in a generic context, without taking into account the specific requirements of a decentralized cloud. In this paper we identify some of these requirements and introduce Koala, a novel overlay network designed specifically to meet them.**

I. Introduction

Recent technological developments in IoT, data analytics and latency-sensitive applications have pointed out the limitations of the centralized cloud model [16]. Unprecedented volumes of data are generated at the edge of the network and need to be pre-processed at once in order to propagate only aggregated results for further processing [17]. Additionally, latency-sensitive applications, such as augmented reality, demand physically close processing resources in order to be usable. However, current centralized clouds fail to meet these demands. Data is directly transferred to a central processing entity which is often distant from their source. This results in both high bandwidth usage and increased latency.

In order to overcome these issues, increasingly more research is being devoted to alternative architectures such as edge computing. The edge architecture aims at providing distributed computing resources at the edge of the network, close to where data is generated. However, to this day there is no actual cloud implementation based on such architecture. An emerging research focus consists in capturing the right design requirements for an implementation that can fully take advantage of this architecture.

In particular, the Discovery project [1] considers adding some computing resources in each Point of Presence (PoP) of existing backbone networks, by transforming them into small datacenters, as shown in Figure 1. However, Discovery's authors argue that just by having resources close to the users does not fully leverage locality if the management of these resources is still centralized. Therefore, they propose to decentralize the cloud management too.

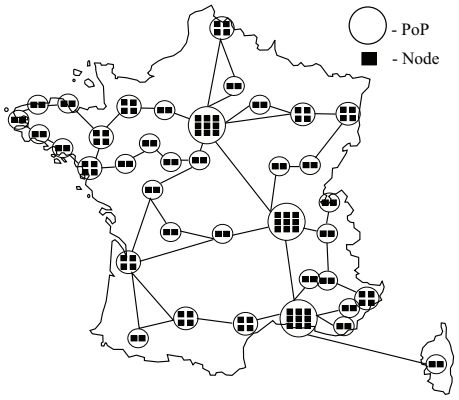


Fig. 1: Envisioned Discovery infrastructure based on Renater[1].

In contrast to traditional clouds, in decentralized clouds there is no single controller node having an overall view of all compute nodes. Each node may act as a controller or as a compute node, and has only a partial view of the network. In order to provide decentralized management, such as service localization or workflow management, these nodes need to collaborate and coordinate their actions. This can be done by means of an overlay network. However, this overlay needs to comply with the same principles of cloud decentralization mentioned earlier: reducing network utilization and latency.

In traditional overlays [20, 22], nodes build their routing tables by actively searching for their ideal entries. In addition, even when the overlay is built, continuous background maintenance protocols make sure that the overlay remains updated despite nodes joining or leaving the network. Proactive overlay building and maintenance introduces a significant cost in terms of network utilization, and worse yet, this traffic does not serve any functional purpose for the application using the overlay. Various existing overlays also implement latency-aware routing. Nonetheless, this implementation is mostly based on gossiping or active probing of multiple nodes in order to select the ones with a lower latency.

[1]The French research backbone network: https://www.renater.fr

2330-2186/17 $31.00 © 2017 IEEE
DOI 10.1109/CloudCom.2017.64

We propose Koala, an overlay network that eliminates dedicated overlay maintenance traffic, while still providing latency-aware routing. The main idea is to adapt the overlay maintenance to the application needs. In other words, nodes build and maintain the overlay using only application traffic. If the application stops, the overlay maintenance stops too. We do this by piggybacking network information in application messages. Similarly to Symphony [11], Koala nodes use the Kleinberg distribution [6] to identify their ideal routing table entries, but they do not actively search for the corresponding nodes. They rather wait to discover them as more application traffic passes through them. Additionally, Koala provides a lazy approach for implementing latency-based routing. When choosing the next step for a message, nodes use a decision function which is based on a trade-off between greedy logical routing and latency-based routing. Finally, we propose three alternatives for taking into account the edge topology where nodes are organized in small datacenters (as in Figure 1). The idea is to integrate this information within our overlay by means of node identifiers in order to avoid high inter-datacenter latency by forcing intra-datacenter routing.

The remainder of this paper is organized as follows. Section II presents Koala's related work. Section III presents the underlying mechanisms of Koala. Section IV presents some preliminary results. Section V concludes and discusses our on-going work.

II. Related Work

This paper discusses the design of an overlay network matching the requirements of an edge computing platform comprising resources that are geographically-dispersed and organized in small data-centers. Overlay networks are generally classified into two categories, namely *structured* and *unstructured*. While *structured overlays* have strong guarantees on the logical distance between nodes, they rely on costly maintenance mechanisms. In contrast, *unstructured overlays* exhibit less formal guarantees but incur a less extreme maintenance cost, especially under high churn.

A. Flexibility in routing tables

Early structured overlay networks such as Chord [22], Pastry [20] or Kademlia [13] exhibit a diameter logarithmic to the size of the network. This is enforced through strict rules on how routing tables are filled. Nonetheless, it has been shown that some flexibility can be introduced in routing tables, as advocated by Symphony [11], where routing tables are filled with randomly (yet carefully) chosen links. At the cost of an estimation of the size of the network, Symphony exhibits a routing complexity similar to Chord. Our proposal, Koala, builds upon this idea of a flexible choice to fill routing tables but advances it towards a lazy creation of these tables.

B. Reducing the diameter of the overlay

A series of works deal with enriching routing tables to reduce the routing complexity. Kelips [5] organizes nodes in groups of fixed size, each node keeping one link to every node in its own group and one to a node in every other group. Routing tables in Kelips have a size in $O(\sqrt{N})$, where N is the number of nodes. Kelips relies on gossiping to discover such nodes, incurring an important, periodic traffic. Zero-Hop DHT (ZHT) [10] reduces the diameter to $O(1)$ through direct knowledge of every node from every node, making each message delivery a 0-hop or 1-hop process. While a complete view of the network is viable on nodes within a cluster of a few thousand nodes, the efficiency of such an approach at a larger scale is questionable, regarding its maintenance cost. We devised Koala considering that such a cost is not tolerable and mostly useless when there is no application traffic.

C. Latency-awareness

Latency-awareness is introduced into overlays primarily through (i) topology-aware overlay construction, (ii) topology-aware neighbor selection, and (iii) proximity routing [2].

Topology-aware overlay construction consists in choosing logical neighbours according to their physical proximity. Fluidify [19] follows a similar path as it updates the logical identifiers according to the identifiers of physically close nodes. A similar technique is used also by Vivaldi [3], but for determining node coordinates. In contrast to Fluidify, which receives latencies from an underlying gossip protocol, Vivaldi uses application traffic, and therefore is lazier.

Topology-aware neighbor selection consists in choosing the physically closest node among viable candidates. This is done only when there is some flexibility to fill a given entry of the routing table. Therefore, T-Chord [14] relaxes Chord's constraints for having a link at a precise distance, by allowing a range of values around this distance. In this way, links with the lowest latency which satisfy the logical constraints are selected for each entry. However, similar to Fluidify, T-Chord is based on the same gossip mechanism. In Pastry [20] instead, nodes obtain lazily low-latency neighbors during the joining phase but then need to actively update them at a second stage.

Proximity routing consists in taking routing decisions not only according to the remaining logical distance to the destination, but also on the delay of the next hop. It is the lightest approach, since it does not require any routing table modification. This approach has been adopted by Hypeer [21], which aims at introducing latency-awareness in Chord by changing the order of its hops. Its authors show that by making Chord choose the next hop based also on latency improves its overall performance. Koala also uses this technique due to its lightness, but is more flexible than Hypeer as it is not restricted by the order of hops.

D. The unstructured approach to dissemination

Scamp [4], HyParView [9] and CLON [12], focus on building overlay networks specialized in disseminating a message within a group of nodes. In particular, CLON targets a multi-cloud environment and introduces some locality-awareness by favoring local nodes while building the views. Other works focus on creating links between nodes with similar contents, assuming that they serve similar queries [7].

Message dissemination does not require a structured overlay, and can be achieved using gossip-based communication, provided that nodes keep a relevant partial view of the network. While oriented towards the maintenance of overlay networks, such approaches are not suitable for point-to-point routing, which is the target of this paper.

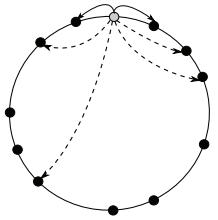
E. Laziness

Several studies target the reduction of the maintenance cost. Kademlia [13] limits the overlay's maintenance cost by leveraging application traffic. It piggybacks contact information in queries, but in a limited fashion: a message includes its sender's ID, allowing recipients to record it. In a slightly different context, Relax-DHT [8] reduces maintenance cost of data stored in a structured overlay, by relaxing the invariant that each data block is replicated on the k closest nodes of a block's root, thus avoiding the need for a systematic rearrangement of blocks whenever a node joins or leaves. In contrast to Koala, Relax-DHT does not deal with the maintenance of the network structure itself, which still relies on periodic exchanges. Yet in another context, lazy learning based on application traffic was explored to optimize the placement of multiple collaborating virtual machines in a Cloud environment [18].

III. Protocol

A. The basic structure

Koala is a ring-based protocol. Nodes are assigned unique m-bit IDs which are generated through a cryptographic hash function whose values are uniformly distributed in the circular identifier space. Each node has a routing table which is composed of (i) neighbors and (ii) long links. Neighbors are nodes with successor and predecessor IDs in the identifier space, thus forming the ring. The number of neighbors can be configured based on a trade-off between resilience and maintenance cost. Given our focus on low maintenance cost, we commonly use 4 neighbors (2 predecessors, 2 successors). Long links are nodes with IDs that vary in distance from the ID of the node itself and allow shortcuts in the ring. For each entry in the routing table Koala maintains: the logical ID on the ring, the IP address, the Round Trip Time (RTT) to this node, and another field, called the *ideal ID (IID)*, which we explain later. Figure 2 shows an example of a node and its routing table.



node ID: 0 m = 4

	ID	IP	RTT	IID	dist
neighbors	1	x.x.x.x	50	-	1
	15	x.x.y.x	150	-	1
long links	2	x.x.x.y	20	2	2
	14	x.x.z.x	250	14	2
	3	x.x.y.y	120	12	3
	10	x.x.x.z	75	8	6

Fig. 2: A small Koala network (m=4), with the routing table for node 0 (2 neighbors and 4 long links).

The choice of long links is crucial for routing performance. Similarly to Symphony [11], Koala selects long links based on a continuous version of the Kleinberg distribution. This distribution defines the probability for a node to pick a long link at a distance d as follows: $p[d] = \int_1^M 1/(d\ln(M))$, where M is the maximal logical distance in the system and d varies from 1 to M. Note that this distribution favors closer IDs over distant ones. Koala's routing is bidirectional. Thus, the distance between two IDs is the smallest of the two distances, calculated in both directions. Therefore, the maximal distance is half of the ring, so $M = 2^{m-1}$ and d varies from 1 to 2^{m-1}.

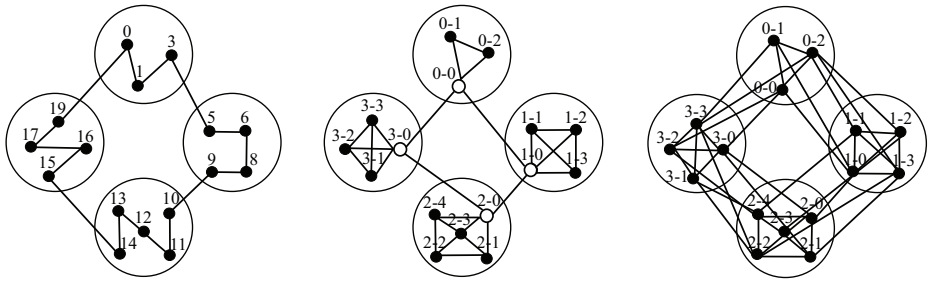
By repeatedly generating random numbers using this probability distribution function, we obtain the logical distances at which the IDs of the long links should be from the ID of the node itself. We, then, convert these distances into IDs by randomly either adding or subtracting them to the ID of the node modulo 2^m. The obtained IDs constitute the IIDs for the long links of a node. Nodes do not actively search for their ideal long links, but rather wait to learn about the existence of nodes with IDs that are close to their generated IIDs. The number of long links is configurable, but we opt for a multiple of m: $Nr_{ll} = C * m$, where C is a small constant so as to keep a routing table logarithmic in the size of the network.

B. Joining and lazily discovering the network

As detailed above, a joining node is assigned a random ID, say q. Based on this ID, it defines the IIDs for each of its long links, but without specifying their associated IDs or IP addresses. The only IP address it knows is the one of a contact node b (commonly called the *bootstrap* node) already in the network. The join procedure starts by sending a request to b. Based on a routing strategy discussed later, b forwards the request to other nodes until it reaches a potential neighbor p of q. At that point, p contacts q and they will exchange routing table information. From this exchange, q learns about its other neighbor s and performs the same exchange with it in order to place itself correctly in the ring, between p and s.

During a routing table exchange, nodes also look for potential long links. A node compares the IDs from the received entries against the IIDs of each local entry. In case a received ID is closer to the IID of a local entry than the current ID associated with that entry, then the new ID will replace the current one, and so will the IPs. On a joining node, when its entries have no IDs yet, the first received entry fills the empty fields (ID, IP) of all the local entries. During such an exchange the RTT value is set to a default value L_{def} which may not reflect the actual RTT, but it gets soon updated to a real value once a message with that entry is exchanged.

Long links are also discovered while routing application requests. Routed messages are enriched with information about nodes present in the network. This information derives from two sources: the path of the message, and the routing tables of the nodes along this path. The second source improves the diversity of information coming from the first one, since it does not strictly depend on the path itself. The way it works is as follows: each message contains in its payload a structure similar to the routing table of a node, but with a small size in order to limit the payload overhead. Upon initialization, a



a) single-ring flat b) hierarchical c) multi-ring flat

Fig. 3: Approaches for taking datacenter division into account. (Long links are not shown)

message generates random IIDs uniformly for its entries, in contrast to nodes, which use the Kleinberg distribution. When reaching a node, the message and the node will exchange their routing tables in the same way nodes do between them. Thus, at each step, a message will find IDs which are closer to its IIDs and will feed them to the next destination which might find this information useful for improving its long links.

C. Latency-aware routing

The Kleinberg distribution yields a logarithmic complexity in terms of hops when a greedy algorithm is used. Although the number of hops is a significant factor for determining the delivery time of the message, the cost of each hop is relevant too. A message can be delivered faster using many cheap hops rather than a few expensive ones.

A purely *greedy* algorithm selects the next step based only on the minimization of the remaining logical distance. On the other hand, a purely latency-based algorithm selects the cheapest entry in terms of RTT, but on the condition that it also reduces (even slightly) the logical distance. This latter condition is added in order to guarantee the correctness of the algorithm. We investigate an algorithm which takes into consideration both factors, namely logical distance and physical distance (in terms of RTT). This simple algorithm examines all the entries in the routing table and it gives a rating to each of them using the following formula:

$$R_{entry} = 1/(\alpha * d(entry.ID, dest.ID) + (1-\alpha) * norm(entry.RTT)) \quad (1)$$

where $d(entry.ID, dest.ID)$ is the remaining logical distance if we choose this entry, $norm()$ is a normalization function which converts the value of RTT into the same scale as the logical distance, and α is a coefficient which determines the weight of each of the factors. The entry with the highest rating R is selected. This function provides a tool for determining if getting as logically close as possible to the destination can justify its cost in terms of latency, and conversely if going to the closest node justifies this potentially extra hop. An $\alpha = 1$ results in a purely greedy algorithm, and an $\alpha = 0$ in a purely latency-based one. Any other value of α in the interval $(0, 1)$ represents a trade-off between the two factors and can potentially reduce the overall routing latency.

D. Static topology-awareness in edge clouds

The latency-aware routing described above is an example of discovering *dynamically* lower latency paths between any two nodes. Given our context, where nodes are organized in datacenters, dynamic discovery is useful when these nodes reside in different datacenters (inter-datacenter routing) because the physical path between them is not known. In case these two nodes reside in the same datacenter (intra-datacenter routing), discovering dynamically a path between them is not necessary because we already know that there is a direct physical link between them. This information is *statically* provided by the topology itself and we need to integrate it in our overlay.

We do this by slightly changing the way the node identifier is assigned so that it reflects also the datacenter to which the node belongs. In this way, we can distinguish between inter and intra-datacenter routing policies. We have devised three possible approaches for taking the edge topology into account, namely: *single-ring flat*, *hierarchical* and *multi-ring flat*.

a) The single-ring flat approach: aims at assigning close identifiers to nodes in the same datacenter, as shown in Figure 3a. This is done by splitting the identifier space into different equal-size intervals, each of which is reserved to only one datacenter. The size of these intervals depends on a rough estimation of the maximum number of nodes per datacenter. Note that this approach requires to increase the size m of the identifier for supporting the same system size.

b) The hierarchical approach: consists in splitting the identifier in two parts. One which identifies the datacenter and the other one which identifies the node within that datacenter. In this approach all the nodes in the same datacenter know each other directly. However, inter-datacenter routing is handled by a leader node which acts as a gateway for all the other nodes, as shown in Figure 3b (leader in white). Neighbors and long links are selected based on the datacenter part of the identifier. This approach provides a smaller ring diameter and requires less links to be lazily maintained. Nevertheless, it is not very robust as the leader is a single point of failure and it requires additional mechanisms (such as replication and leader election algorithms) to be resilient.

c) The multi-ring flat approach: is very similar to the hierarchical one except that there is no leader. In this case, all the nodes provide inter-datacenter routing (Figure 3c).

This approach is more robust than the hierarchical one as it distributes the load more evenly. It has the same ring diameter, but it might require more application traffic for each node to find its ideal long links. In addition, it enables nodes within the same datacenter to collaborate with each other in order to provide a more efficient inter-datacenter routing.

IV. Preliminary results

We evaluate Koala using PeerSim [15], a cycle-based simulator which introduces one or more events at each cycle. An event can be either a node joining or a node sending a message to another one. For the underlying physical topology we generate a two-dimensional random graph with a small vertex degree (around 2 on average) based on the Waxman model [23]. Each vertex of this graph represents a datacenter which consists of multiple nodes. The latency between two directly linked datacenters is a function of the Euclidean distance between them. The simulated latency between nodes in the same datacenter is set to a random number between 0.05 and 0.5 ms. We conducted three experiments to evaluate each of the concepts we introduced earlier: lazy network discovery, latency-awareness and static topology-awareness.

a) Lazy network discovery: In this experiment we observe the gradual creation of the Koala overlay as nodes join the network and communicate with each other. We focus particularly on the inter-datacenter routing, and therefore we assume only one node in each datacenter. In other words, the approaches discussed in III-D are not applicable (or indistinguishable). We consider a network of 1000 datacenters/nodes ($m = 10$), where each node has 4 neighbors (2 successors, 2 predecessors) and 40 long links ($C = 4$). We do not take latency into account in this experiment ($\alpha = 1$). At each cycle a node joins the network and a message is routed from a random source to a random destination. Consequently, after cycle 1000 all the nodes have joined, so from that point on (until cycle 10,000) we only perform routing operations. After each cycle we measure the latency and the number of hops for the routed message. We show the results in Figure 4.

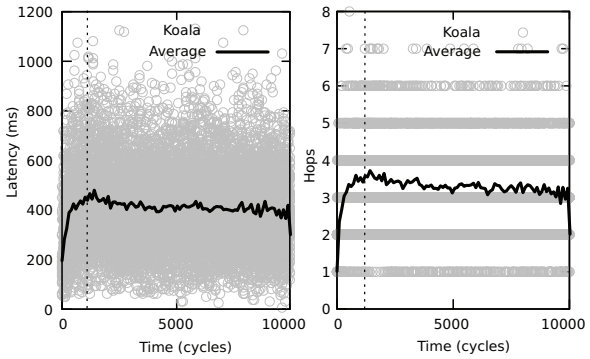


Fig. 4: Joining and learning.

We observe that as more nodes join the network (until cycle 1000, depicted in dashed lines), the average latency and number of hops grow continuously. During this phase, nodes discover their long links using both ways: by exchanging routing tables with their immediate neighbors (on join) and by searching the piggybacked information of messages passing through them. We notice that the discovered links are sufficiently close to the Kleinberg distribution for delivering a logarithmic complexity. In the second phase, after cycle 1000, nodes use only piggybacking to further improve their long links. Hence, we notice a gradual reduction of both latency and number of hops.

b) Latency-aware routing: In this experiment we demonstrate the effect of taking hops latency into account when routing a message, as described in III-C. The experimental setup is similar to the first experiment, except that we consider a network where all the nodes have already joined. Therefore, at each cycle we just route. For our evaluations we take into account only the last 1000 cycles, since learning is more stabilized at that point. We compare the latency and the hops for $\alpha = 1$, meaning that hop latency is not taken into account, and $\alpha = 0.5$, meaning that hop latency and logical distance are equally taken into account. Figure 5 shows the result of this comparison.

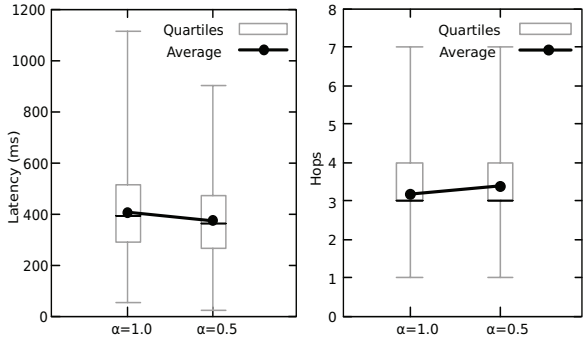


Fig. 5: Locality-awareness.

We notice that even though the average number of hops increases when $\alpha = 0.5$, the overall average latency decreases. Despite the small rate of improvement, this is an important observation as it encourages us to investigate further on the criteria we use for selecting entries in the routing table. Until now we have considered only long links which are selected based on their logical distance distribution. We suspect that selecting entries with a small latency might increase the rate of improvement as the rating function (1) would have more diversity of choice.

c) Static topology-awareness: In this experiment we show some preliminary results regarding the three approaches discussed in III-D: single-ring flat (SR-flat), multi-ring flat (MR-flat) and hierarchical. We consider a system of 1000 datacenters, each consisting of 100 nodes. The experiment setup is the same as the previous one (with $\alpha = 0.5$). However, we increase the number of cycles to the number of nodes in the system and we take into account only the last 10,000 cycles. At each cycle, all the three versions of Koala are asked to route a message from the same source to the same destination. We

measure the latency and the inter and intra-datacenter hops. We show the results in Figure 6.

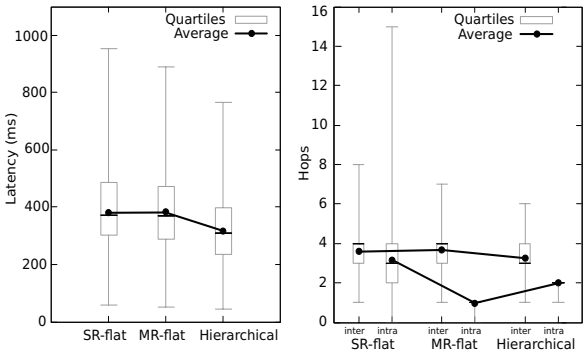


Fig. 6: Topology-awareness: comparison of approaches

Despite having a larger diameter and less links between the nodes in the same datacenter, we observe that the single-ring flat approach still delivers similar performance to the other two approaches. This is due to the use of internal low-latency hops before taking an external expensive hop to the next datacenter. This is shown also in the right part of the figure where the number of intra-datacenter hops is larger than in the other two approaches. However, for this to work we need to take latency into account. Therefore, the value of α needs to be less than 1. The hierarchical approach delivers the best performance as the leaders find quite fast their ideal long links due to the high traffic which passes through them, but this comes at the cost of being less robust. For the multi-ring flat approach in this experiment we have not considered any intra-datacenter collaboration for routing externally. We suspect that such collaboration can improve significantly its performance. Furthermore, adding some uniformly random links to each node might increase the diversity of links within the datacenter and improve even further the potential collaboration.

V. Conclusions and Future Work

Decentralized management of edge clouds requires a network overlay which leverages locality and reduces network utilization. Despite their benefits, current overlays have a high cost on network utilization due to their periodic overlay maintenance. In this paper we introduced Koala, a lazy overlay network which minimizes maintenance costs by piggybacking network information on the application traffic. Thus, if the application stops the overlay maintenance stops too. Additionally, we showed how this overlay provides latency-aware routing and takes into account the topology of an edge cloud.

Our preliminary results show that, for a uniform traffic, our overlay delivers logarithmic complexity and reduces the overall latency by examining the cost of each hop. In addition, we introduced three approaches for integrating datacenter locality into our overlay and suggested a few techniques for improving them. However, more experiments are required for determining how different factors affect this overlay. We are currently investigating further the impact of α and the inclusion of additional links selected using different criteria. Finally, we plan to devise a collaboration technique for the multi-ring flat approach in order to further improve its performance.

References

[1] M. Bertier, F. Desprez, G. Fedak, A. Lebre, A.-C. Orgerie, J. Pastor, F. Quesnel, J. Rouzaud-Cornabas, and C. Tedeschi, *Cloud Computing: Challenges, Limitations and R&D Solutions*. Springer International Publishing, 2014, ch. Beyond the Clouds: How Should Next Generation Utility Computing Infrastructures Be Designed?

[2] M. Castro, P. Druschel, C. Hu, and A. Rowstron, "Topology-Aware Routing in Structured Peer-to-Peer Overlay Networks," in *Future Directions in Distributed Computing, Research and Position Papers*, 2003.

[3] F. Dabek, R. Cox, F. Kaashoek, and R. Morris, "Vivaldi: A decentralized network coordinate system," *SIGCOMM Comput. Commun. Rev.*, 2004.

[4] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié, *Scamp: Peer-to-Peer Lightweight Membership Service for Large-Scale Group Communication*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 44–55.

[5] I. Gupta, K. P. Birman, P. Linga, A. J. Demers, and R. van Renesse, "Kelips: Building an Efficient and Stable P2P DHT through Increased Memory and Background Overhead," in *Second International Workshop on Peer-to-Peer Systems (IPTPS'03)*. Springer, 2003, pp. 160–169.

[6] J. Kleinberg, "The small-world phenomenon: An algorithmic perspective," in *Proceedings of ACM STOC'00*. ACM, 2000, pp. 163–170.

[7] G. Koloniari, Y. Petrakis, E. Pitoura, and T. Tsotsos, "Query workload-aware overlay construction using histograms," in *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, ser. CIKM '05. ACM, 2005, pp. 640–647.

[8] S. Legtchenko, S. Monnet, P. Sens, and G. Muller, "RelaxDHT: A Churn-Resilient Replication Strategy for Peer-to-Peer Distributed Hash Tables," *ACM Transactions on Autonomous and Adaptive Systems*, 2012.

[9] J. Leitao, J. Pereira, and L. Rodrigues, "Hyparview: A membership protocol for reliable gossip-based broadcast," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, 2007.

[10] T. Li, X. Zhou, K. Brandstatter, D. Zhao, K. Wang, A. Rajendran, Z. Zhang, and I. Raicu, "ZHT: A Light-Weight Reliable Persistent Dynamic Scalable Zero-Hop Distributed Hash Table," in *27th IEEE International Symposium on Parallel and Distributed Processing*, 2013.

[11] G. S. Manku, M. Bawa, and P. Raghavan, "Symphony: Distributed Hashing in a Small World," in *Proceedings of the 4th USENIX Symposium on Internet Technologies and Systems*, Seattle, USA, Mar. 2003.

[12] M. Matos, A. Sousa, J. Pereira, R. Oliveira, E. Deliot, and P. Murray, *CLON: Overlay Networks and Gossip Protocols for Cloud Environments*. Springer Berlin Heidelberg, 2009, pp. 549–566.

[13] P. Maymounkov and D. Mazières, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," in *First International Workshop on Peer-to-Peer Systems (IPTPS'02)*. Springer, 2002, pp. 53–65.

[14] A. Montresor, M. Jelasity, and O. Babaoglu, "Chord on demand," in *Fifth IEEE International Conference on Peer-to-Peer Computing*, 2005.

[15] A. Montresor and M. Jelasity, "PeerSim: A scalable P2P simulator," in *Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09)*, 2009.

[16] C. Mouradian, D. Naboulsi, S. Yangui, R. H. Glitho, M. J. Morrow, and P. A. Polakos, "A comprehensive survey on fog computing: State-of-the-art and research challenges," *CoRR*, 2017.

[17] A. Papageorgiou, B. Cheng, and E. Kovacs, "Real-time data reduction at the network edge of internet-of-things systems," in *11th International Conference on Network and Service Management (CNSM)*, 2015.

[18] J. Pastor, M. Bertier, F. Desprez, A. Lebre, F. Quesnel, and C. Tedeschi, "Locality-aware cooperation for VM scheduling in distributed clouds," in *Proceedings of Euro-Par 2014*.

[19] A. C. Resmi and F. Taiani, *Fluidify: Decentralized Overlay Deployment in a Multi-cloud World*. Cham: Springer International Publishing, 2015.

[20] A. I. T. Rowstron and P. Druschel, "Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems," in *Proceedings of IFIP/ACM Middleware'01*. Springer-Verlag, 2001.

[21] S. Serbu, P. Felber, and P. Kropf, "HyPeer: Structured Overlay with Flexible-Choice Routing," *Computer Networks*, 2011.

[22] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications," in *Proceedings of ACM SIGCOMM'01*, 2001, pp. 149–160.

[23] B. M. Waxman, "Routing of multipoint connections," *IEEE Journal on Selected Areas in Communications*, 1988.