

A Distributed Bootstrapping Protocol for Overlay Networks^{*}

Miguel Matos and António Sousa and José Pereira and Rui Oliveira
{miguelmatos,als,jop,rc}@di.uminho.pt

Universidade do Minho, Braga, Portugal

Abstract. Peer to peer overlay networks have become popular due to their inherent scalability and resilience properties that come naturally from their decentralized nature. Unfortunately, despite this decentralized approach there is still one important piece that remains centralized: a set of servers to provide identifiers to peers joining the system. This initial step is necessary as new peers need to obtain some contact points in order to establish links to them and join the overlay. This puts out of the protocol model a crucial mechanism as it introduces an external centralized entity to manage the process, leading to problems of scale and fault-tolerance. In scenarios where the churn rate is considerable, the cost of maintaining the list of known peers by the well-know servers may be unbearable. Furthermore, if the peer identifiers provided are not evenly distributed across the universe, this will inevitably cluster the overlay, making it more prone to partitions. In this paper we propose a fully decentralized protocol to obtain those set of initial contact peers using the already deployed infrastructure. By conducting an extensive experimental evaluation we show the effectiveness of the protocol and reason how the ideal number of contact peers may be provided in a fully decentralized fashion.

1 Introduction

Epidemic or gossip-based dissemination protocols present an attractive approach to application level message dissemination in very large scale and dynamic systems. The applicability to these scenarios comes directly from its resilience and scalability, despite its underlying simple principle.

The aforementioned scalability is achieved by spreading the dissemination load among all the participating peers. As all nodes contribute to the dissemination effort, the load imposed on each one only grows logarithmically with the system size, as has been shown in [4]. Another crucial characteristic of gossip-based protocols is the redundancy in the messages received by each node. As peers establish multiple links among them, this leads to a redundant link mesh, the overlay, and consequently to redundancy in the message transmission. This

^{*} This work is supported by HP Labs Innovation Research Award, project DC2MS (IRA/CW118736).

natural redundancy enables the resilience to node and links failures, as multiple embed dissemination trees are always available for each message injected in the system.

As identified in [10] to build an epidemic dissemination algorithm there are several issues that need to be properly addressed: membership, which addresses how peers get to know each other and how many they need to know; network awareness, which deals with reflecting the underlying network topology in the logical connections made among peers; buffer management, which deals with the amount of information each peer needs to keep in order to provide a given reliability level of the dissemination process; and message filtering, whose goal is to take into account the interest of the peers in the dissemination process.

In this paper, we deal with a particular aspect of the membership issue. Although there are multiple protocols that address membership management such as [2,7,11,6,8,3], none of them, to the best of our knowledge, address the problem that arises when a node needs to join the overlay. For this initial joining process to be successful, the joiner needs to know a set of neighbours already on the overlay to which it can connect to. This problem is known as bootstrapping and has been addressed by relying on a set of well-known servers that maintain a list of well-known peers in the overlay. Therefore, upon boot the peer contacts those servers to obtain the set of needed identifiers and then initiates the joining process.

In this paper we propose a fully distributed approach to this problem in the context of the DC2MS [1] project. The goal of the project is to provide a management infrastructure for Cloud environments. The Cloud infrastructure is composed by several data centers spread worldwide and organized in a federation. Therefore, the first goal of the project was to built an adequate membership management and dissemination protocol [9] that is able to provide reliability and scalability guarantees while minimizing the load imposed on the long distance links that connect the several data centers.

As such, we enrich the existing membership management protocol with a distributed bootstrapping mechanism in order to cope with the dynamic and fully distributed scenario as the assumption of a set of well-known poses serious impairments to reliability and scalability and present a single point of failure. Even if the servers are replicated for availability the maintenance of the set of well-known peers is still a problem. In highly dynamic environments such as the ones the DC2MS project targets, peers constantly join and leave the system, and therefore the maintenance of such set of peers may become unbearable, due to the churn factor, or even unattainable in a reliable and scalable fashion. Therefore, we propose a novel fully distributed protocol that removes the necessity of those set of well-known servers as well as the need to known any peer a priori. Our bootstrapping protocol is based on a probabilistic approach and is able to offer an arbitrary set of peers to a joining node. This further increases the usability of the protocol as it allows joining nodes to quickly become indistinguishable of nodes already on the overlay, by establishing multiple links with neighbours since the beginning.

The rest of the paper is organized as follows. In Section 2 we present the membership management protocol previously developed [9] that tackles the network awareness problem. While at first this may seem unrelated to the specific topic addressed in this paper, the goal is to give deeper insight on the way links are established among the peers in the overlay construction protocol and then, from the local knowledge of each peer, proceed to the bootstrapping protocol we propose in Section 3. Furthermore, this unusual approach to the background section is derived from the fact that, to the best of our knowledge, no previous attempt at addressing this problem in a distributed fashion exists. Then, in Section 4, we experimentally evaluate the performance of the protocol and access whether or not it achieves the desired goals. Finally, in Section 5 we conclude the paper.

2 Background

CLON [9] is an overlay management protocol that addresses the network awareness problem when establishing links among the peers participating in the dissemination effort. It builds on the work done in the Scamp protocol [2], due to its natural convergence to the right view size, which makes it adaptable to ever changing system sizes without requiring any global knowledge.

CLON is presented here because the bootstrapping algorithm proposed in this paper and described in Section 3, explicitly takes advantage of the local knowledge available in CLON nodes, as well as the semantics of the overlay management protocol. Nonetheless, the bootstrapping algorithm presented below could be applied to a variety of overlay management protocols such as [2,7,5] as all of them share similar semantics with CLON.

In Scamp joining nodes are integrated into the view of existing peers in a way that the average view size converges to $\log(N) + c$, where N is the size of the system and c a protocol parameter related to fault tolerance [4]. The value of this ideal view size has been shown previously in [4]. By integrating nodes with a probability inversely proportional to the view size, and ensuring a given number of subscriptions is sent by the joiner, the protocol is then able to converge to the ideal view size value presented above.

CLON pushes this further by integrating nodes taking into account its locality. By relying on an oracle that should be able to distinguish local nodes from remote ones, the protocol is able to integrate those nodes with different probabilities, effectively biasing the overlay to match the underlying network. The mathematic model behind the basic algorithm is well detailed in [2] and therefore we will skip it. The typical applicability scenario of CLON consist of a set of local area networks connected through long-distance wide area networks, where we want to reduce the bandwidth consumption of the dissemination effort.

Listing 1.1 presents the CLON protocol. Upon boot (lines 1 to 4) the new node obtains a contact node from the set of well-known servers, adds the contact to its view and sends an *handleSubscription* request to the contact. Upon receiving

```

1  upon init
2    contact = getContactNode()
3    view.Add(contact)
4    send(contact,handleSubscription(myId))
5
6  proc handleSubscription(nodeId)
7    for n ∈ view
8      send(n,handleJoin(nodeId))
9
10   for i=0; i < c; i++
11     n = randomNode(view)
12     send(n,handleJoin(nodeId))
13
14  proc handleJoin(nodeId)
15    keep = randomFloat(0,1)
16    keep = Math.Floor( localityOracle(viewSize,nodeId) * keep)
17
18    if (keep == 0) and nodeId ∉ view
19      view.Add(nodeId)
20    else
21      n = randomNode(view)
22      send(n,handleJoin(nodeId))
23
24  proc localityOracle(viewSize,nodeId)
25    if isLocal(nodeId)
26      return viewSize * 0.7
27    else
28      return viewSize + viewSize * 0.3

```

Listing 1.1. CLON protocol

the request (lines 6 to 12) the contact node sends an *handleJoin* to all its neighbours and c additional requests to random neighbours.

The nodes that receive the *handleJoin* (lines 14 to 22) then calculate the probability of integration by means of the *localityOracle*, that can be configured in a per-scenario fashion and shall return a positive integer indicating the preference given to that node. Small values will increase the probability of integration while higher values will decrease it. Therefore, by configuring the oracle to return low values to local nodes and high values to remote ones it is possible to effectively bias the overlay to match the underlying topology. Then, in line 16 the value returned by the oracle is weighted against a random value *keep* and, if the outcome is zero the joining node is integrated into the view of the node performing the calculation. If the node is not integrated the *handleJoin* request is forwarded to a random neighbour, in order to preserve the number of links established in the overlay.

2.1 Experimental Evaluation

For the sake of completeness we present an experimental evaluation of the impact of using a network-aware overlay management protocol in the bandwidth consumption on long-distance links such as the ones connection the data centers of a cloud infrastructure.

The simulations presented have been run on a custom simulator in Python. To this end, we used the NETWORKX package which provides graph manipulation facilities to represent the overlay and the gossiping process on top of it. To manipulate the simulation results we recurred to the RPY package which provides a wrapper to the R programming language. The simulation kernel uses discrete time and relies on global state, handling events in a FIFO fashion.

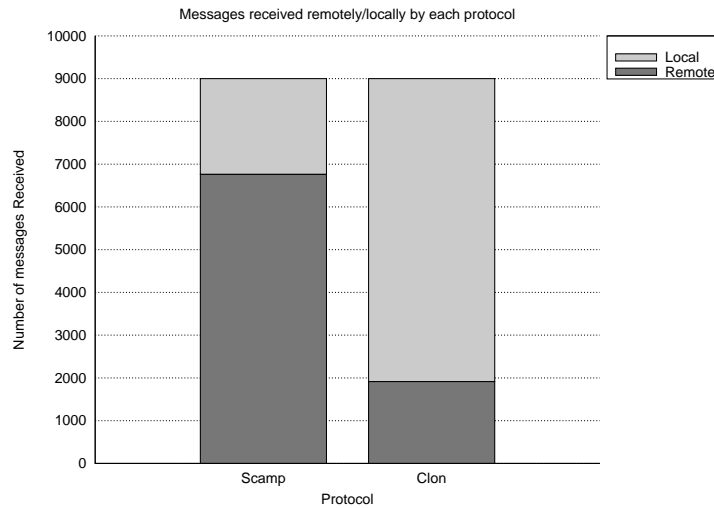


Fig. 1. Messages Received Locally/Remotely by each node.

In the experiment presented here, and in the one presented in the next Section, the scenario consists of 5 local areas with 200 nodes each, accounting for a total of 1000 nodes. Furthermore the c parameter is configured to 6 which yields that the average view size is $\log(1000) + 6 = 9$. A detailed analysis of the experimental evaluation of CLON is presented in [9] and is out of the scope of this paper. Here we just present the impact of the number of messages that cross the local and remote links when using the Scamp and CLON protocols. The dissemination protocol used relies on a flooding infect and die dissemination strategy, as it is the most used approach in epidemic dissemination.

The result of this experiment is shown in Figure 1 and consists of having each node inject a new message on the system and then counting the remote and local messages received by each node.

As it is possible to observe, the number of total messages received by each node (local + remote) is 9000 for both protocols. This comes directly from the average node degree, which is 9 and from the number of messages injected on the system 1000, one for each node. If we now look at the number of remote messages it is clear that CLON brings a significant improvement over Scamp, reducing the messages remotely received from around 7000 to 2000, a cut of more that half.

It is important to note that this improvement comes at no penalty in the reliability of the protocol in presence of faults but that analysis is out of the scope of this paper and can be seen in [9]. Furthermore, with a dissemination protocol that also takes into account locality, also presented in [9], it is possible to achieve an improvement of over an order of magnitude in the number of messages that traverse the long-distance links.

For the rest of this paper it is important to keep in mind the way nodes join the overlay in CLON, namely the way the additional c subscriptions are sent as this has a direct impact on the bootstrapping protocol described in the next Section.

3 Bootstrapping Protocol Description

To the best of our knowledge, existing overlay management protocols solve the bootstrapping problem by assuming there is an external entity that provides node identifier(s). This is, in general, not satisfactory as it puts out of the protocol model an important aspect of the overlay building mechanism, and tends to be addressed by relying on static centralized solutions, such as having one or more servers providing the set of initial identifiers. With node churn this set is hard to maintain up to date and provides a brittle solution, even if we made the unrealistic assumption that those servers do not fail. In this proposal we address this problem by fully decentralizing the initial discovery mechanism, making every node in the overlay a potential server in the peer to peer spirit. The only requirement we impose is the availability of a broadcast primitive on the local network where the new node is physically connected. This requirement is virtually guaranteed to be satisfied in modern network architectures, due to the pervasiveness of the TCP/IP communication protocol. As in the CLON protocol described above, the bootstrapping protocol also is network-aware, as it tries to provide the joining node with local and remote peer identifiers. Nonetheless, the solution is flexible enough to be used in other overlay management protocols such as [2,3,5].

Recalling the integration mechanism by which joining nodes get added to the views of other nodes, it is possible to observe that the view size converges, on average, to $\log(N)+c$. However, in the the CLON protocol presented in Section 2, a node joining the overlay only establishes one link with the contact node received from the external mechanism, which effectively impairs the connectivity of the joiner. Notwithstanding, if instead of obtaining just one contact node, the joiner obtained 'some' contacts, its connectivity will be improved since the beginning. Ideally, this value should be near the average node degree, because in this way the new node will be indistinguishable of nodes that have been on the overlay for more time. Next, we will explain the functioning of the protocol which is depicted in Listing 1.2 and then reason how the ideal result could be achieved.

Upon boot, the node uses the broadcast primitive to request contacts from all the nodes in its local area, as can be seen on lines 3 and 4. If upon reception of the request all nodes replied to the originator, this may led to problems, in a phenomena known as acknowledgement bomb. This phenomena stems from the fact that if every node in a large scale system replies to the originator of a broadcast, the network will become suddenly overloaded, and the requester may be overrun by the amount of replies. To overcome this problem, we rely on an oracle, *sendOracle*, that should instruct whether the node should reply to the request or not, with a given probability. Although the oracle may be configured

```

1 myC = c
2
3 upon init()
4   BCAST(CONTACT, myself)
5
6 proc CONTACT(nodeId)
7   if (sendOracle())
8     if(externalContactOracle(nodeId))
9       contact = randomExternalNode()
10    else
11      contact = myself #or node = randomLocalNode()
12      send(nodeId,(CONTACTREPLY, contact))
13
14 proc CONTACTREPLY(contact)
15   keep = random()
16   keep = Math.Floor((viewSize + 1 *keep)
17
18   if (keep == 0)
19     if view.size() > 0)
20       scheduleCheck()
21       view.Add(contact)
22       send(contact,handleJoin(myId))
23
24   if (myC > 0)
25     send(contact,handleJoin(myId))
26     myC = myC - 1
27
28 #possible send oracle
29 proc sendOracle()
30   totalNodesEstimate = 10(viewSize-c)
31   localNodesEstimate = totalNodesEstimate / 5 #our scenario has 5 local areas
32
33   if localNodesEstimate < 1
34     localNodesEstimate = 10viewSize
35
36   reply = viewSize / localNodesEstimate
37   seed = randomFloat(0,1)
38   return seed < reply
39
40 #possible external oracle
41 proc externalContactOracle(nodeId)
42   rand = randomFloat(0,1)
43   return rand < 0.5
44
45 proc cCheck()
46   while(myC >= 0 )
47     contact = randomNode()
48     send(contact,handleJoin(myId))
49     myC = myC -1

```

Listing 1.2. Bootstrapping Protocol

in a naive manner, lets say reply only with a probability of 10%, the amount of replies generated will be effectively reduced, alleviating the problems of the acknowledgement bomb phenomena.

If the oracle instructs the node to reply (line 7), there is another decision that needs to be made: whether to provide a local or remote node as a contact. If this is not taken into account, and joining nodes are always provided with local contacts only, the reliability of the overlay will be compromised. This comes from the fact that if joining nodes only get to know local nodes, over time the number of remote known nodes will decrease considerably and the overlay will partition around the local areas. In the Listing, we show a simplistic oracle on lines 41 to 43 which instructs the protocol to reply half the times with a remote nodes and half of the times with a local one, but naturally this could be configured to the application requirements. After this initial steps where the oracles decide about replying and the kind of node chosen, the contact is sent to the requester (line 12).

Upon reception of a reply (line 14), the joining node should decide whether or not to integrate the contact in its view, based on the same procedure of the original Scamp protocol, where the probability of integration is inversely proportional to the size of the view. The motivation behind this conditional integration is to ensure that even if the oracle of the repliers is not well configured, the view size will still be within the normal bounds of the system. Without this restriction the view size of the joining nodes could become too large and therefore impact the quality of the obtained overlay. If the contact node is to be integrated by the joining node, the latter adds it to its view, i.e. establishes a link with it, and sends it a *handleJoin* as in the CLON protocol presented in the previous Section. With these changes the join mechanism is effectively decentralized and inverted. Instead of being the contact node to send the subscription requests as in CLON, it is now the responsibility of the joining node to send them. This change has an immediate impact on the protocol as the c additional join requests sent to random neighbours must still be transmitted. As such, the joining node becomes the responsible for sending those additional requests. However, instead of sending those additional join request to just one contact, lets say the first received, we decide to distribute them among the several contacts obtained. To this end, the joining node now has an additional variable myC which is initially set to the c protocol parameter, as seen in line 1. Then, for each received contact, the joiner sends the normal *handleJoin* request plus one additional copy until c copies are sent (lines 24 to 26). To prevent the case where less than c contacts are received, and therefore not enough additional copies could be sent, upon the reception of the first contact the protocol schedules the execution of the *cCheck* procedure on a point in the future. This scheduling may be only approximate and should start when it is expected that all the contact replies have been received, which on a local area network shall be close to the first one. This procedure only checks if enough copies have been sent, and if not they are sent to randomly chosen nodes.

After analysing the protocol it is now time to clarify how we could exploit the local knowledge available, in order to obtain optimal results in the bootstrapping mechanism. Optimal in this context means that the joining node establishes as much contacts as the average view size, therefore becoming indistinguishable from other nodes. To achieve this exact behaviour, we will need global knowledge in order to calculate the ideal view size and reply exactly with that amount of contacts to the joiner. Of course this solution is not acceptable, as it will impair all the work done previously on decentralizing the entire protocol. Nonetheless, if we rely only on local knowledge it is still possible to approximate this behaviour, in a probabilistic fashion. In fact, all nodes have a powerful estimation tool of the total amount of nodes, the size of their view. As the view size converges to $\log(N)+c$ where a N is the number of nodes in the system it is straightforward to estimate locally the total number of nodes. If we also know the number of local areas available, which probably is well-known (for example the number of data centers of the cloud provider), it is possible to estimate the number of potential repliers to the contact request, i.e. the number of nodes in the local area, and thus reply with the adequate probability. An example of an oracle configured in this way is shown in lines 28 to 38. The oracle estimates the total number of nodes (line 30), calculates the number of local nodes based on this estimation (line 31) and replies with a probability based on this calculations (lines 36 to 38). Although the configuration presented should be well suited to a wide range of scenarios, we still abstract it with an oracle in order to not impair the applicability of the mechanism in other, maybe unpredicted, scenarios. It is important to notice that if the estimation of local nodes is inaccurate, which happens when the view size is inferior to c , the probability of replying adequately will be compromised. This comes from the fact that if the *totalNodesEstimate* becomes smaller than 1, in the case c is greater than the *viewSize*, then the calculation on line 36 will yield a value greater than 1 and therefore the node will always reply to the contact request. This is easy to observe as $viewSize/localNodesEstimate$ always yields a value greater than 1, when the *localNodesEstimate* is strictly smaller than 1, which will impair the optimal behaviour we intend to achieve. This abnormality is corrected by ignoring the wrong local nodes estimation and making it $10^{viewSize}$ (lines 33 and 34), which is a rough estimate of the total number of nodes.

4 Experimental Evaluation

In this Section we present the experimental evaluation of the bootstrapping protocol in order to assess if it satisfies the requisite of providing the joining nodes with several contact nodes.

To this end we used different *sendOracle* configurations, starting from a naive one and improving it to obtain the optimal configuration described in the previous Section. The results obtained can be observed in Table 4. The table is organized as follows: the first two columns describe the configuration of the *sendOracle* and *externalContactOracle*, respectively; the third column presents

the number of messages exchanged by the nodes in the runs of the bootstrapping mechanism, without considering the messages sent by the joiners after receiving the contact; in the fourth column it is possible to observe the total number of replies a given joiner obtained; finally the last three columns show the total, local and remote nodes effectively integrated in the view of the joiner.

The scenario is the same as the one described in Section 2 that is we have 5 local areas each one with 200 nodes ammounting for 1000 nodes in the total, the c parameter is set to 6 and as such the ideal view size is 9. For each oracle configuration we run 5000 join operations and extracted the averages of the results obtained. Furthermore, each new run is independent of the previous, i.e. we run the bootstrapping mechanism for a joining node, and after the process ends, we proceed to the next round with a new overlay.

As it is possible to observe for all the configurations the *externalContactOracle* is configured to return true with a probability of 50%, which means that half of the replies will be with local nodes and the other half with remote ones.

The first row of the table shows a naive configuration of the *sendOracle*, where it is configured to always return true. As such, the bootstrap generates 400 messages, 200 for the initial broadcast procedure and 200 replies to the joiner as each node always reply to the requests in this configuration. With this configuration the joiner obtains 200 replies, one for each node on its local area, but only integrates 20 on its view. This is due to the probability of integration being restricted by the actual size of the view, as explained in Section 3. Of this 20 nodes integrated into the view 12 are local and 8 are remote.

On the next configuration, in the second row of the table, the *sendOracle* only replies to the contact requests 5% of the times, which results in sending to the joiner 10 replies ($0.05 * 200 = 10$). Of this 10 replies only 5 are effectively integrated into the view of the joiner, of which 3 are local and 2 remote. The configuration of 5% is just a arbitrary small probability chosen to infer the behaviour of the bootstrapping mechanism.

On the next configuration we start to exploit the local knowledge available in order to approximate the desired optimal behaviour. In this configuration the oracle estimates the total number of nodes in the system, but does not knowns the number of local areas, and as such the probability of replying is only based on the total number of nodes estimated. Nonetheless, the result is interesting as it gets closer to the ideal value of 9 links established by the joining node.

Finally, in the last configuration we exploit the knowledge of the previous configuration but assume that the number of local areas is known beforehand, which may be reasonable for certain scenarios. This configuration corresponds to the example oracle given in Listing 1.2. With this knowledge available, it is possible to achieve the optimal results in the bootstrapping mechanism. In fact, with this configuration the joiner receives 38 contact replies and of those integrates 9 in its view, the ideal value in this scenario. Furthermore, the proportion of local and remote nodes is also closely approximated, as the biasing mechanism of CLON presented previsouly tends to build views with 7 local nodes and 2 remote ones.

To conclude the analysis of the bootstrapping mechanism, the above experiments show that it is possible to achieve near optimal configurations with the local knowledge available at each node, as in the third configuration. Furthermore, if the number of local areas of the federation is known beforehand, it is possible to obtain an optimal bootstrapping mechanism that makes joining nodes indistinguishable from the other nodes already present in the overlay.

Oracles Probabilities		Msgs Generated	Contacts Offered	Nodes Integrated		
send	external			Total	Local	Remote
1	0.5	400	200	20	12	8
0.05	0.5	210	10	5	3	2
viewSize global	0.5	286	86	13	8	5
viewSize local	0.5	238	38	9	6	3

Table 1. Different bootstrapping configurations.

5 Conclusion

In this paper we proposed a novel fully decentralized bootstrapping mechanism with the goal of removing the requirement of the traditional approach relying on a set of well known servers to provide identifiers to nodes joining the overlay. The advantages of this new bootstrapping mechanism are many fold. First, we eliminate the need to maintain a list of well-known nodes somewhere out of the model, as contact nodes are drawn from all the local nodes on the overlay. As such this also has an impact on the quality of the overlay as contact nodes are chosen more uniformly and therefore the problem of the well-known nodes and its direct neighbours having high degrees is alleviated. This problem draws from the fact that the well-known nodes tend to be frequently contacted by joining nodes and as such they will eventually integrate some of the joiners in their views, hindering the overall node degree distribution. Furthermore, a joining node now knows several initial contact points instead of just one, which effectively improves its connectivity. Finally, the subscriptions along with the c additional copies are sent to different parts of the overlay instead of only the neighbours of the contact node, as in the original protocol, which counters the clustering around those nodes.

As the experimental evaluation attests, the bootstrap protocol is able to provide the joiner with several contact nodes and, taking advantage of the local knowledge available at each node it is possible to achieve optimal configurations that allows joining nodes to become indistinguishable from the rest of the overlay. Nonetheless, if the local knowledge is not available the protocol still achieves satisfactory results, as the experimental evaluation results shown.

Despite the decentralization that could be achieved by the proposed bootstrapping protocol, there is still one important drawback. Although this mechanism works well when the overlay is well established and there are known remote nodes, the initial bootstrap of a whole local area could not be addressed with this mechanism. This is due to the fact that initially no remote nodes are known on the starting local area and as such we still require a set of well-known remote nodes to bootstrap a whole local area, provided by the administrator. Nonetheless, after this initial step the external mechanism could be discarded and, as such our proposal may be used for the rest of the life-cycle with the application with all the advantages we pointed above.

References

1. DC2MS: Dependable Cloud Computing Management Services. <http://gsd.di.uminho.pt/projects/projects/DC2MS>, 2008.
2. A.-M. K. A. Ganesh and L. Massoulié. SCAMP: Peer-to-peer lightweight membership service for large-scale group communication. In *Networked Group Communication*, pages 44–55, 2001.
3. A.-M. K. A. Ganesh and L. Massoulié. Hiscamp: self-organizing hierarchical membership protocol. In *Proceedings of the 10th workshop on ACM SIGOPS European workshop*, pages 133–139. ACM, 2002.
4. L. M. A.-M. Kermarrec and A. Ganesh. Probabilistic reliable dissemination in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems*, 14:248–258, 2001.
5. T. T. F. Makikawa, T. Matsuo and T. Kikuno. Constructing overlay networks with low link costs and short paths. *Sixth IEEE International Symposium on Network Computing and Applications*, pages 299–304, July 2007.
6. J. P. J. Leitão and L. Rodrigues. HyParView: A membership protocol for reliable gossip-based broadcast. In *IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 419–428. IEEE Computer Society, 2007.
7. A.-M. K. L. Massouli and A. Ganesh. Network awareness and failure resilience in self-organising overlay networks. In *In Proceedings of the 22nd Symposium on Reliable Distributed Systems*, pages 47–55, 2003.
8. M. Lin and K. Marzullo. Directional gossip: Gossip in a wide area network. In *In Proceedings of Third European Dependable Computing Conference*, volume 1667 of *Lecture Notes in Computer Science*, pages 364–379. Springer, 1999.
9. M. Matos, A. Sousa, J. Pereira and R. Oliveira. CLON: Overlay Networks and Gossip Protocols for Cloud Environments. Technical Report DI-CCTC-09-13, Centro de Ciências e Tecnologias de Computação, Universidade do Minho, April 2009.
10. A.-M. K. P. Eugster, R. Guerraoui and L. Massoulié. From epidemics to distributed computing. *IEEE Computer*, 37(5):60–67, May 2004.
11. D. G. S. Voulgaris and M. Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management*, 13(2):197–217, June 2005.