Lazylon: a Lightweight and Churn-aware Membership Protocol

Pierre Louis Aublin
SARDES team
INRIA Grenoble – Rhône-Alpes
Email: pierre-louis.aublin@inria.fr

***Abstract*—Gossip-based peer sampling protocols are used in peer-to-peer systems to construct robust overlays. However the existing protocols do not have any mechanism to adapt themselves to the churn. In this paper we present Lazylon, a protocol based on Cyclon which modulates its shuffle period according to the churn rate. We will see that this protocol allows us to keep a random network in presence of massive failures and that it uses the bandwidth according to the churn. Finally the protocol can be enhanced and this work can be seen as an introduction.**

***Index Terms*—Peer-to-peer, membership management, gossip, peer-sampling, epidemic protocol**

I. Introduction

A. Presentation

Nowadays applications are more and more distributed instead of being centralised, to lower down the costs and to be more scalable. For example if a lot of people is watching the TV, it will be better for a new client to ask other clients instead of the overloaded server. By that the application moves from a client/server model to a peer-to-peer model, where each machine, also called a peer, is at the same time a client and a server.

As this is completely decentralised, all the peers need to build and maintain a graph (called overlay) which has to be resilient to peer failures, thus it has to look like a random graph [1].

This is where the peer-sampling service steps in, by letting the peers know each others [2]. Usually this service uses a gossip-based approach, because it is lightweight and robust, even if peers are continuously joining or leaving the system.

Gossip-based protocols were first introduced to maintain replicated databases [3]. They can be used as a building-block in monitoring systems [4], but also to handle events such as flash crowds [5].

In this paper we present Lazylon, a membership protocol which allows the peers to change their shuffle period according to the churn rate. Our protocol is built upon Cyclon [6], a highly scalable and robust membership protocol.

For that, we will first present Cyclon and Lazylon protocols in Sections 2 and 3 respectively. Then, in Section 4, we are going to evaluate our protocol. In order to do so we have used a cycle-based simulator written in Java at the INRIA Grenoble – Rhône-Alpes. Next, we will talk about possible improvements in Section 5. Finally, we conclude in Section 6.

B. Background

In peer-to-peer systems, the peers communicate directly with other peers to exchange data, like the Gnutella protocol. Hence the peers need to have access to a list of other peers. This can be done using a server (this is the case with BitTorrent), but this can also be done by the peers themselves: for instance this is what GosSkip [7] does. In the second case, as the peers do not have an infinite amount of memory, they can have only a partial view of the other peers in the network. Its size is noted c, because this view is sometimes called the cache.

For example, the Lpbcast [8] protocol maintains a partial view of the network and ensures the broadcast of a message over all the peers with a high probability (Lpbcast stands for Lightweight Probabilistic Broadcast protocol). To do so, when a peer receives a message, it tries to modify its view by removing the peers that have unsubscribed and adding the peers that have recently subscribed in the network. This operation, where a peer exchanges its view with another peer, is called a shuffling. This operation is the basic block in active peer-sampling protocols as it allows the peers to keep up-to-date their view of the network and to know new peers.

During a shuffle, a peer P sends a list of peers to the peer Q it gossips with. The size of this list is called the shuffle length and it is noted l. Finally, the Lpbcast protocol shuffles its view periodically. We call that period a shuffle period or a round.

There also exist protocols that shuffle only upon an event. This is the case for Scamp [9] and HiScamp [10]. In most of the cases the view size is fixed, but for these two protocols it changes according to the size of the network. Moreover these two protocols modify the overlay structure only when new peers are joining and leaving. Hence there is a special way to join or leave the network, and the only mechanism for a peer to detect that it is isolated from the network is the number of received messages: if it has not received messages for a given period then it is isolated and it has to join the network again.

By that it takes time to detect failures, which is not (or at least less) the case with active protocols, that is to say protocols which shuffle periodically their view, like Cyclon or Newscast [11]. These two protocols are very similar. Basically there are 3 main differences between them. Firstly Newscast choose the peer to gossip with at random. Secondly it sends to this peer its whole view. Thirdly when a peer receives a view it merges it with its own view, while Cyclon peers exchange their views. But as Cyclon it gives an age (in fact a timestamp) to each peer, to keep only the freshest entries in its view. Finally, even though they are similar, the results are not the same: Newscast in-degree distribution shows that a lot of peers have a small in-degree, while a few peers have a high in-degree. The in-degree of a peer *P* corresponds to the number of peers that have *P* in their view, i.e. that know *P*, at a given moment. We are interested in the in-degree distribution because it is an important property in graphs, and it allows us to express the randomness of the network. Moreover it gives information about the robustness of the network: it shows the existence of weakly or highly connected peers [6]. Furthermore Newscast peers forget dead peers much more rapidly than Cyclon peers. The dead peers, also called stale references, are references in a peer's view toward peers that are no longer present in the network, for example because of a failure.

There is another protocol which handles churn, i.e. joining or leaving peers, in a better way than Cyclon: HyParView [12]. This protocol uses TCP connections between peers to have a better reliability, like NeEM [13]. It uses 2 views, like Scamp, but not for the same purpose. And it shuffles these views by a membership protocol. A membership protocol is a protocol which allows peers to know each others. It uses the peer-sampling service to provide peers to gossip with. Cyclon, Lpbcast, Scamp, HiScamp, NeEM, etc. are membership protocols, but GosSkip or CREW [14] are not membership protocols. Indeed even though these protocols maintain a view, they do not use it to improve the robustness of the overlay, but only to send data. Unfortunately one of HyParView's drawback is that it floods the network with its messages.

II. Cyclon

Cyclon is an inexpensive gossip-based membership management protocol for unstructured P2P overlays, fully detailed in [6]. In this section we are going to present this protocol, and to expose some of its properties

A. Protocol description

Cyclon peers' view contains a list of <*peer, age*>. The peers are identified by their ip and the port number. The age is used to have an up-to-date overlay and also to control the distribution of pointers toward peers in the view. To modify this view, the peers shuffle periodically. According to [6], the shuffling period should be of at least 10 seconds. Finally, the view has a fixed size.

To join the network, a new peer *P* needs only to know one other peer, known as the *introducer*. Several methods exist for *P* to know a peer in the network, like multicast addresses or contacting a special server.

When a peer wants to exit the network, there is no protocol for that, and it simply exits the network. Thus departures either normal or resulting of a failure are handled in the same way.

When a peer *P* performs the periodic shuffling, it executes the following steps:

1) increase by 1 the age of all neighbours
2) select neighbour Q with the highest age among all neighbours and $l - 1$ other random neighbours ($1 \leq l \leq view\ size$)
3) replace *Q*'s entry with a new entry of age 0 and *P*'s address
4) send the updated subset to *Q* (<*shuffling request*>)
5) receive from Q a subset of no more than l of its own entries (<*shuffling response*>)
6) discard entries pointing to *P* and entries already in *P*'s view
7) update *P*'s view to include all remaining entries, by firstly using empty view slots (if any) and secondly replacing entries among the ones sent to *Q*

When a peer P receives a message of type <*shuffling request*>, it executes the following steps:

1) select at random a subset of size no more than l
2) send this subset to the initiating peer (<*shuffling response*>)
3) receive from Q a subset of no more than l of its own entries (this is this message)
4) discard entries pointting to *P* and entries already in *P*'s view
5) update *P*'s view to include all remaining entries, by firstly using empty view slots (if any) and secondly replacing entries among the ones sent to *Q*

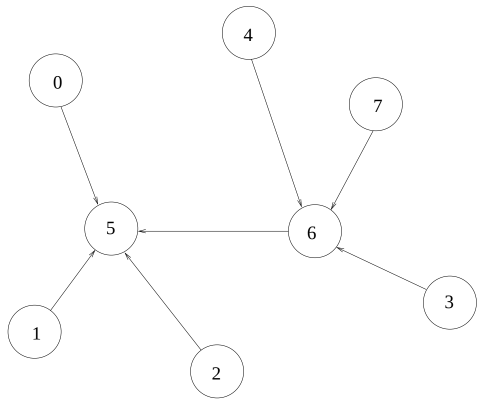
The Figure 1 presents an example of the shuffling operation. In this example the network is composed of the peers 0 to 7, the view size is 4, the shuffle length is 2 and the peer 5 initiates the operation. This peer sends the subset {5, 2} to the peer 6 in the <*shuffling request*>. Then the peer 6 receives this message and sends the <*shuffling response*> with the subset {4, 7}. It merges the received subset by adding the peer 5 and replacing the peer 4 by the peer 2 in its view. At the same time the peer 5 receives the <*shuffling response*> and merges the subset with its view by removing the peers 6 and 2.

B. Properties

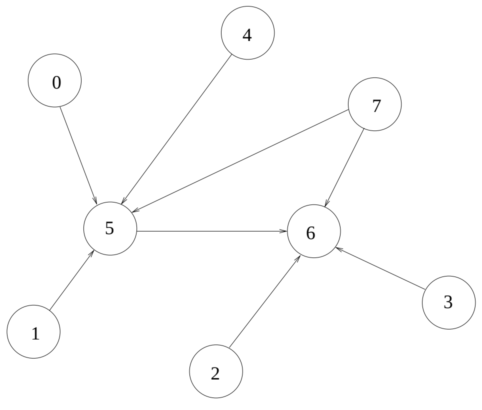
In this section we will see some properties of Cyclon. For that we have used our simulator saying that a cycle corresponds to Cyclon's shuffle period, and setting the view size to 20 and the shuffle length to 9. These parameters are the same through all this paper.

Firstly the in-degree distribution looks like a Gaussian curve centred (according to [6]) in $view\ size$. The Gaussian distribution in mathematics is associated with the normal law, which expresses the random behaviour of a variable [15]. Therefore it expresses the randomness of our network graph.

Secondly, Cyclon converges in less than 10 cycles to a random graph, whatever the bootstrapping method is. To see that we have used three different bootstrapping methods: *Random* where the views are filled at random; *Growing* where the views are initially filled only with the first peer which has



(a) Before shuffling



(b) After shuffling

Fig. 1. Representation of a shuffling operation initiated by the peer 5. $P \rightarrow Q$ means that the peer *Q* has *P* in its view.

been created; and *Ring* where each peer has a pointer toward the previous and the next (in terms of peer id) peer.

Finally, Cyclon is a good protocol but it sends a lot of messages in a linear way: with 1000 peers, $2 * 10^7$ messages were sent during 10000 cycles. So the bandwidth cost is always the same, even if there is no churn and the overlay is random. This is why in the next section we are going to present Lazylon, a protocol which modulates its shuffle period according to the churn rate.

III. Lazylon

In this section we are going to present our protocol, Lazylon. Its name comes from the fact that it is based on Cyclon but is lazier than it.

A. Protocol description

Lazylon is a membership management protocol which detects churn and changes its shuffle period according to it. And as Lazylon is based upon Cyclon, it handles messages of type <*shuffling request*> or <*shuffling response*> like Cyclon. The periodic shuffling operation of Lazylon peers is the same as the one of Cyclon peers. Moreover, in Cyclon, when a peer *P* gossips with a peer *Q*, it puts a timer on *Q*, and if it gets no reply within a predefined time, then it assumes this peer not to be active. Therefore, upon the timeout, we increase by one the number of non-active peers since the last churn rate computation. This number is stored in the $nbChurn$ variable and its value is 0 when the peer is created.

Then the peer periodically changes its shuffle period according to the following algorithm:

```
1:  every $CRU$ $period\_time\_unit$ do
2:    $prevChurnRate \leftarrow churnRate$
3:    $churnRate \leftarrow nbChurn$
4:    $nbChurn \leftarrow 0$
5:
6:    if $churnRate = 0$ then
7:      $inc \leftarrow max_{i=0}^{SST}(i)$ such that $SP + i \leq MSP$
8:      $SP \leftarrow SP + inc$
9:    else if $churnRate > prevChurnRate$ then
10:     $maxDec \leftarrow SP - 1$
11:     $maxChurnRate \leftarrow CRU \ / \ SP$
12:     $decrease \leftarrow maxDec * churnRate \ / \ maxChurnRate$
13:     $SP \leftarrow SP - \lfloor decrease \rfloor$
14:   else if $churnRate < prevChurnRate$ and $SP < MSP$ then
15:     $SP \leftarrow SP + 1$
16:   end if
17: end every
```

In this algorithm there are some global variables:

SP	This is the current shuffle period value.
MSP	This is the maximal shuffle period. We need an upper bound on the shuffle period because if we don't do that, and if the network experiments high churn after a long and calm period, it will take too much time to recover from that failure. This limitation can be seen as the speed limitation on the road: cars can go faster than it, but for security reasons this is forbidden.
CRU	This variable controls how often we compute the churn rate. It means Churn Rate Unit and corresponds to a period of time. Its value is equal to MSP because we obtained the best results with that.
SST	This is the shuffle step. If it has a low value then it will take time for the peers to reach MSP, while if it has a too high value then it will take more time to recover from a sudden failure.

From the lines 2 to 4 we compute the churn rate and we save the previous churn rate value.

Then it is time to adapt its shuffle period. For that we have 3 cases: if there is no churn (lines 6–8), if there is more churn (lines 9–13), or if there is less churn (lines 14–15).

On line 7, $max_{i=0}^{SST}(i)$ **such that** $SP+i \leq MSP$ means the maximal value for i between 0 and SST such that the actual shuffle period plus i is less or equal than the maximal shuffle period. For example, let's say that $SST = 5$, $SP = 10$ and $MSP = 20$. Thus the maximal value for i is SST, because $SP + SST \leq MSP$. On the other hand, if $SP = 17$, then

$i = 3$, because $SP + 3 \leq MSP$ but $SP + 4 > MSP$. By this increase the nodes will shuffle less, thus they will use less bandwith.

Finally, one can ask why we shuffle less when there is less churn. The answer is simple: the results are almost the same as if we do not, but with that we send less messages (order of hundreds of thousands).

B. *The* MSP *value*

Choosing the maximal shuffle period value has an impact on the network one may consider. Indeed the less MSP, the less stale references we get.

Moreover, when $\text{MSP} = 10$ the number of sent messages is almost linear and it becomes greater when we increase the value of MSP in presence of continuous churn. Hopefully in all the cases the number of sent messages is less than the one of Cyclon.

Finally, the mean in-degree is not affected by this choice, even if it was 19 instead of 20 with $\text{MSP} = 10$.

To conclude, we have chosen $\text{MSP} = 50$ in our design because it seems to be a good compromise. However we don't have a proof and maybe one can find a better value. We can also say that if one knows he will have churn at high rates in his network, then it is better to choose a low value, like 10, because the peers will send few messages (compared to other values) and they will have a few stale references. However, if there is no churn or at least a few churn and if we have to pay attention to the bandwidth, then it is better to choose a higher value.

IV. Evaluation

During our simulations, in most of the cases the peers where shuffling (at least at the beginning for Lazylon peers, as we will see) every cycle. However, to allow Lazylon peers to shuffle more than Cyclon peers in presence of churn, we have set the default shuffle period to 10 cycles in presence of churn. Moreover we have set MSP at 50, SST at 5 and we had 1000 peers.

The churn was either continuous, i.e. a certain amount of peers depending on the churn rate was leaving and joining the network every 500 cycles from cycle 1000 to 3000, or it was massive, i.e. half of the peers where killed at cycle 500. In the case of continuous churn the experimented rates where 1%, 5%, 10%, 30%, 50% and 80% of peers. We also took a look at how peers arrival affects the network, but the results where the same for the two protocols with every bootstrapping method presented in section II-B.

First of all, Lazylon peers change their shuffle period according to the churn rate, as we can see in the Figure 2.

This change results in less sent messages: without churn Lazylon has sent 97.31% less messages than Cyclon. When there is churn the number of sent messages becomes greater, but since there is no more churn it does not increase that much, as shown in Figure 3. With this result we clearly see that Cyclon uses a lot of bandwith for nothing.

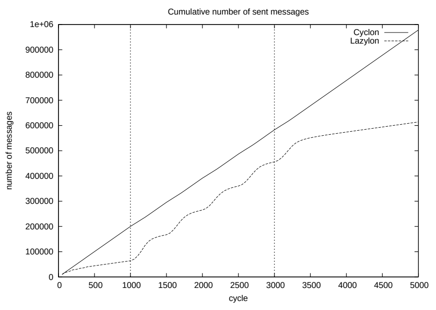


Fig. 3. Total number of sent messages between cycles 0 and 5000 with a churn rate of 10% in presence of continuous churn for Cyclon and Lazylon. The vertical lines represent the start and the end of the churn.

Moreover the adaptive shuffle period makes Lazylon peers to recover (i.e. to reach again a mean in-degree equals to the view size) more rapidly than Cyclon peers in case of massive failures. While with continuous churn Cyclon peers recover more rapidly. In both cases the order of difference is of hundreds of cycles.

Finally Lazylon's drawbacks are that it leaves more stale references in the views and it takes more time to remove them, as shown in Figure 4. Also, as the nodes don't have the same shuffle period, the in-degree distribution is not random anymore: during our experiments we obtained 2 random topologies in our network and not only one for all the nodes. But this can be simply solve by aggregating the average churn rate value.

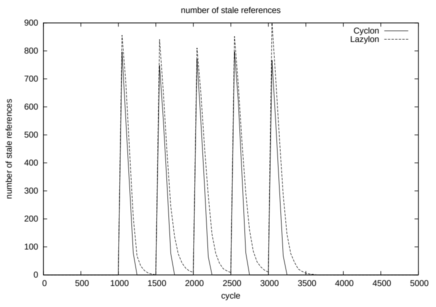


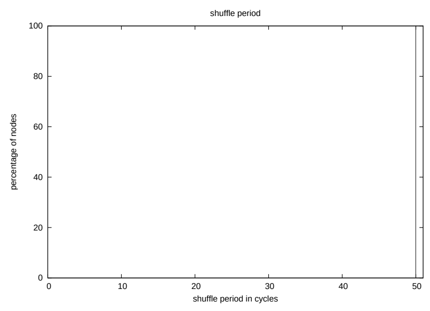
Fig. 4. Total number of stale references through the simulation in presence of continuous churn at a rate of 5% for Lazylon and Cyclon.

V. Enhancements

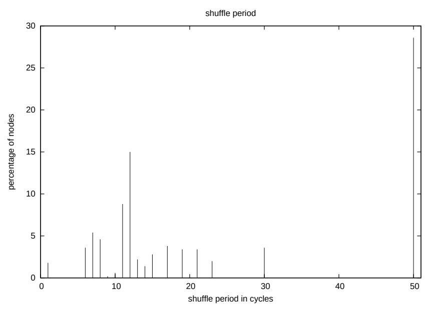
We have seen in the previous section that Lazylon is cheaper than Cyclon, and that it recovers from churn slower than Cyclon, except when there is a massive failure. This is why we will now see how to enhance our protocol. Of course we present them one by one, but it is possible to mix them in order to obtain better results.

A. *Possible enhancements*

In this subsection we will lists the enhancements we have tested, while we will show some results about them in the next subsection.



(a) Before churn



(b) 250 cycles later

Fig. 2. peers' shuffle periods at cycles 999 (a) and 1249 (b), with continuous churn at a rate of 30%.

Firstly, an idea to decrease the number of stale references is to change the way the protocol chooses the peers to send when it performs a shuffle operation. The strategy of Cyclon peers is to select these peers at random, but we have found an alternative strategy which lead us to better results. Indeed, instead of choosing the peers at random, we choose the l (where l is the shuffle length) newest peers in its view (or peers at random if $view\ size < l$). Therefore the probability to send a reference to an inactive peer is smaller than with the default strategy of Cyclon, and by that the peers clear more rapidly the oldest peers (which are the ones which have more chances to be inactive) from their view.

Secondly, we can take a look at the number of received messages: if a peer receives less messages than before, then maybe this is because there is churn and because it is removed from other peers views. However we need to be careful and not to check the number of received messages when there is no churn, otherwise as the peers shuffle period is increasing the peers will think there is churn.

Thirdly we can put a lower bound on the decrease. The idea behind this enhancement is that, as we have seen during the evaluation, with low churn rates the churn is badly detected. Therefore even if the churn rate is low, we are sure that the shuffle period will decrease at least by that lower bound.

Finally, one can observe that in our protocol we compute always the churn rate at least every CRU cycles, even if the shuffle period is less than this value. This is why we can think of the following enhancement: instead of doing that, we can have a sliding window of size CRU. This window is then updated and used at each shuffle to compute the new shuffle period. To experiment this idea we have modified Lazylon's code in the following way:

1) When the peer checks if the neighbour it gossips with is active or not, it also saves this information for the sliding window.
2) Just after this operation it computes an aggregated value of the sliding window content and the new information, and it saves the couple <*new information, aggregated value*> into the sliding window, first removing the oldest element.
3) if the current shuffle is less than the maximal one divided by 2 then it uses the sliding window to change its shuffle period, otherwise it uses the "normal" way. This is not to change the shuffle period by little steps if it was high and then churn appears, because with these little steps it will take more cycles to recover from a massive failure.

The aggregated value is the sum of all the values, knowing that the first quarter is added twice, to weight the newest values more than the old ones, but other and better computations are surely possible. By that we make a difference between configurations which will otherwise result in the same sum, for instance $111 \cdots 000$ and $000 \cdots 111$. We plan to experiment on other strategies for computing the aggregate churn rate.

Then, if we use the sliding window to change the shuffle period, we increase (resp. decrease) the shuffle period by SST when the computed value is less (resp. greater) than the previous one. And in the case where the computed value is equal to 0, then there is no churn so we increase the shuffle period by SST.

B. Results

For each of these optimisations and in presence of churn there is more send messages (order of hundred of thousands) than without. Thus the nodes' shuffle period is lower and the views are refreshed more often.

Moreover, except for the sliding window where it was the same, the number of stale references was lower, as we can see in the Figure 5. The interesting thing we have observed

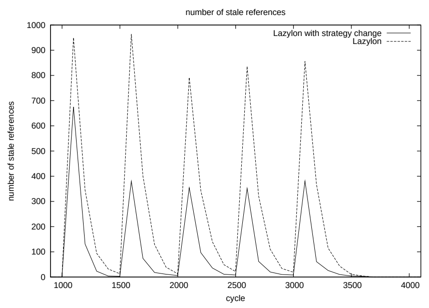


Fig. 5. Number of stale references for Lazylon with and without the strategy change, with continuous churn at a rate of 10%.

with the sliding window is that it took roughly 10 cycles less for the peers to recover from the massive failure than with or without the other strategies. However in all the cases the recovery is achieved as fast as without the optimization.

Finally, even if the peak was sometimes lower with an enhancement, the in-degree distribution looked like the same with or without the enhancements.

VI. Conclusion and future work

In this paper we have presented Lazylon, a protocol based upon Cyclon which detects the churn and adapts its shuffle period according to it, to keep an overlay as much random as possible so as to be very robust. As we have seen it achieves this goal in presence of massive failure, but it splits the network in a random and a non-random part in case of continuous churn. Moreover a drawback is that it leaves stale references longer than Cyclon. However we have seen possible improvements to enhance it.

Now that we have a protocol which sends less messages than Cyclon with good properties, we can leverage it to send other messages, for example to aggregate information [16], or for a better churn rate detection [17].

Furthermore a gossip-based protocol to detect churn already exists [18]. This protocol sends regularly messages to other peers and keeps a heartbeat counter for each peer in its view. Then if the time between the last update of this counter and now is greater than a given threshold, it assumes this peer to have failed. While that was not tested in our work, we think that this protocol may be interesting.

There is also another possible future work: it will be easier to say in our protocol sentences like ”if there is a lot of churn then decrease highly the shuffle period”. This is what fuzzy logic does: fuzzy logic [19] is a form of logic which manipulates vague concepts to adapt a system where criteria are hard to define. Usually the rules are supplied by experts [20], but it can also be done using neural networks [21].

Finally, it will be a good idea to test our protocol in the real-world, or at least with traces from existing peer-to-peer systems. Such traces exist[1], and they are analysed by various papers [22], [23], [24]. The only difficulty for using these traces is that we need to adapt both them and our simulator in order to have the relationship between them and our virtual peers.

Acknowledgment

The author would like to thank everyone from the SARDES team, and more particularly Vivien Quéma, Alessio Pace and Willy Malvault, for their precious help and support during this work.

References

[1] B. Bollobas, *Random Graphs*, W. Fulton, A. Katok, F. Kirwan, P. Sarnak, B. Simon, and B. Totaro, Eds. Cambridge University Press, 2001.

[2] M. Jelasity, S. Voulgaris, R. Guerraoui, A.-M. Kermarrec, and M. van Steen, “Gossip-based peer sampling,” *ACM Trans. Comput. Syst.*, vol. 25, no. 3, p. 8, 2007.

[1]For example Edonkey traces made by Fabrice Le Fessant in 2003/2004: http://fabrice.lefessant.net/traces/

[3] A. Demers, D. Greene, C. Houser, W. Irish, J. Larson, S. Shenker, H. Sturgis, D. Swinehart, and D. Terry, “Epidemic algorithms for replicated database maintenance,” *SIGOPS Oper. Syst. Rev.*, vol. 22, no. 1, pp. 8–32, 1988.

[4] A.-M. Kermarrec and M. van Steen, “Gossiping in distributed systems,” *Operating Systems Review*, vol. 41, no. 5, pp. 2–7, 2007.

[5] A. Stavrou, D. Rubenstein, and S. Sahu, “A lightweight, robust p2p system to handle flash crowds,” *SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 3, pp. 17–17, 2002.

[6] M. v. S. Spyros Voulgaris, Daniela Gavidia, “Cyclon: Inexpensive membership management for unstructured p2p overlays,” *Journal of Network and Systems Management*, 2005.

[7] R. Guerraoui, S. Handurukande, K. Huguenin, A.-M. Kermarrec, F. Le Fessant, and E. Riviere, “GosSkip, an Efficient, Fault-Tolerant and Self Organizing Overlay Using Gossip-based Construction and Skip-Lists principles,” in *IEEE International Conference on Peer-to-Peer Computing*, 2006.

[8] P. T. Eugster, R. Guerraoui, S. B. Handurukande, P. Kouznetsov, and A.-M. Kermarrec, “Lightweight probabilistic broadcast,” *ACM Trans. Comput. Syst.*, vol. 21, no. 4, pp. 341–374, 2003.

[9] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulie, “SCAMP: Peer-to-peer lightweight membership service for large-scale group communication,” in *Networked Group Communication*, 2001, pp. 44–55. [Online]. Available: citeseer.ist.psu.edu/ganesh01scamp.html

[10] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié, “Hiscamp: self-organizing hierarchical membership protocol,” in *EW10: Proceedings of the 10th workshop on ACM SIGOPS European workshop*. New York, NY, USA: ACM, 2002, pp. 133–139.

[11] M. Jelasity, W. Kowalczyk, and M. V. Steen, “Newscast computing,” 2003.

[12] J. Leitao, J. Pereira, and L. Rodrigues, “Hyparview: A membership protocol for reliable gossip-based broadcast,” in *Dependable Systems and Networks, 2007. DSN ’07. 37th Annual IEEE/IFIP International Conference on*, June 2007, pp. 419–429.

[13] J. Pereira, L. Rodrigues, M. J. Monteiro, R. Oliveira, and A.-M. Kermarrec, “Neem: Network-friendly epidemic multicast,” *Reliable Distributed Systems, IEEE Symposium on*, vol. 0, p. 15, 2003.

[14] M. Deshpande, B. Xing, I. Lazardis, B. Hore, N. Venkatasubramanian, and S. Mehrotra, “Crew: A gossip-based flash-dissemination system,” in *Distributed Computing Systems, 2006. ICDCS 2006. 26th IEEE International Conference on*, 2006, pp. 45–45.

[15] W. Feller, *An Introduction to Probability Theory and Its Applications, Volume 1*. Wiley, January 1968. [Online]. Available: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0471257087

[16] D. Kempe, A. Dobra, and J. Gehrke, “Gossip-based computation of aggregate information,” in *FOCS ’03: Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science*. Washington, DC, USA: IEEE Computer Society, 2003, p. 482.

[17] N. Hayashibara, A. Cherif, and T. Katayama, “Failure detectors for large-scale distributed systems,” *Reliable Distributed Systems, IEEE Symposium on*, vol. 0, p. 404, 2002.

[18] R. Van Renesse, Y. Minsky, and M. Hayden, “A gossip-style failure detection service,” Ithaca, NY, USA, Tech. Rep., 1998. [Online]. Available: http://portal.acm.org/citation.cfm?id=866975

[19] G. J. Klir and B. Yuan, *Fuzzy Sets and Fuzzy Logic: Theory and Applications*. Upper Saddle River, NJ, USA: Prentice Hall PTR, May 1995. [Online]. Available: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0131011715

[20] B. Kosko and S. Isaka, “Fuzzy logic,” vol. 269, no. 1, pp. 76–?? (Intl. ed. 62–??), Jul. 1993.

[21] C.-T. Lin and C. Lee, “Neural-network-based fuzzy logic control and decision system,” *Computers, IEEE Transactions on*, vol. 40, no. 12, pp. 1320–1336, Dec 1991.

[22] J. Pouwelse, P. Garbacki, D. Epema, and H. Sips, “A measurement study of the bittorrent peer-to-peer file-sharing system,” 2004. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.3.4761

[23] S. Saroiu, P. K. Gummadi, and S. D. Gribble, “A measurement study of peer-to-peer file sharing systems,” 2002.

[24] S. B. Handurukande, A.-M. Kermarrec, F. Le Fessant, L. Massoulié, and S. Patarin, “Peer sharing behaviour in the edonkey network, and implications for the design of server-less file sharing systems,” in *EuroSys*, 2006, pp. 359–371.