The Wandering Token: Congestion Avoidance of a Shared Resource

Augusto Ciuffoletti

Università degli Studi di Pisa - Dipartimento di Informatica
CoreGRID Institute of Grid Information, Resource and Workflow Monitoring Services

Abstract

In a distributed system where scalability is an issue, the problem of enforcing mutual exclusion often arises in a *soft* form: the infrequent failure of the mutual exclusion predicate is tolerated, without compromising the consistent operation of the overall system. For instance this occurs when the operation subject to mutual exclusion requires massive use of a shared resource.
We introduce a scalable *soft mutual exclusion* algorithm, based on token passing: one distinguished feature of our algorithm is that instead of introducing an overlay topology we adopt a random walk approach.
The consistency of our proposal is evaluated by simulation, and we exemplify its use in the coordination of large data transfers in a backbone based network.
This algorithm is studied in the frame of the CoreGRID Institute of Grid Information, Resource and Workflow Monitoring Services, in cooperation with the FORTH Institute, in Greece.

Key words: congestion avoidance, random walk, token circulation, self-stabilization, soft mutual exclusion

1. Introduction

In an ideal distributed system all resources are equivalently able to play any role. However, in practical applications, it is often the case that the introduction of a centralized resource may be appropriate, in order to reduce the cost, or to improve the performance. The loss of scalability and fault tolerance, which is inherent to the introduction of a centralized resource, is accepted as a trade-off, but, in order to avoid resource congestion, an appropriate access control mechanism must be provided.

To appreciate the trade-off, let us introduce the following scenario: a site that produces a stream of data at a constant rate, like data from a scientific experiment, and a number of geographically dispersed labs that wants to receive the stream of data in order to analyze them or just to keep replicas (see for instance the Virgo experiment Buskulic (2002)). Data source computing capacity and network bandwidth are just adequate for the nominal number of remote users. We have here two centralized resources which is appropriate to leave as such, despite they introduce scalability and fault tolerance limitations: the experiment sensors, and the network link(s) (maybe a backbone) in common between all routes to the remote labs. Replicating such resources would be possible in principle, inappropriate in practice as long as the capacity of such resources is sufficient for the task: load sharing between distinct data sources is awkward, and redundant routes are expensive. In such case a fair sharing pattern of the centralized resources is appropriate, thus accepting a single source of data, and overlapping routes.

Locating such a resource sharing mechanism at resource-side tends to deteriorate scalability, since the resource must also negotiate the use of the service it offers. In addition, all clients should share

Email address: augusto@di.unipi.it (Augusto Ciuffoletti).

the same protocol to negotiate a share of the resource: this is a limit to the deployment for such an architecture, since all potential clients must share the same negotiation protocol used by the specific source. Consider the above scenario: if we want to avoid network congestion due to simultaneous downloads, a resource-side solution should introduce, at the very least, network performance awareness and a queuing system inside the data source.

Here we propose a client-side mechanism designed for environments where the resources are legacy. In the above example, we do not want to introduce an ad-hoc server (like, for instance, in GridFTP Allcock and Perelmutov (2005)), and we do not want to rely on traffic control techniques on the network elements (as in Jacobson (1988)). Instead, we assume that clients coordinate an access pattern that ensures a fair sharing of a plain FTP server through Internet connections.

The basic requirement of a solution to our problem is that resource performance, as observed by a client, must be nominal as long as the overall load does not exceed resource capacity. When requests overtake the capacity of the resource, it should reproduce at client side the effect of an overload, but without stress or damage for the resources. The mechanism must not introduce bounds on system size, other than those enforced by resource capacity: this excludes the adoption of centralized algorithms, that are not scalable, as well as distributed algorithms based on deterministic consensus, that have an heavy footprint.

To further specify our case study, we assume the data source produces at a rate of $650KBps$. The FTP server and the backbone offer a $200MBps$ bandwidth, which saturates with 300 subscribers [1] . We want that subscribers coordinate their access to the infrastructure in order to limit their access to the stream source, thus keeping the overall used bandwidth below $200MBps$, and that data is retrieved timely, so that the data source can flush old data. The bandwidth limit can be exceeded only exceptionally: the Service Agreement states that bursts up to $400MBps$ are delivered with an additional cost, and that packet delivery is not guaranteed over that further limit. This might justify a flexible control over the number of subscribers, that might go over the theoretical maximum of 300 subscribers.

Summarizing, unlike traditional mutual exclusion modeled by a concurrent write on a shared register, our problem statement includes the occasional occurrence of simultaneous accesses to the resource. This is due to the nature of the resource whose performance may degrade (in the case study, degradation is initially only financial) when many are executed simultaneously, but without damage for the consistency of the system. This is formally translated in the following definition:

Requirement 1 *A* soft mutual exclusion *algorithm for the* protected operation **A** *ensures that at any time, with high probability, there is just one agent enabled to perform* **A**. *The probability that more than one agent is enabled falls exponentially in the number of enabled agents.*

We propose a distributed algorithm that implements soft mutual exclusion. The algorithm falls into the *peer to peer* family, since there is no centralized agent, and all participants run the same code. It is *randomized*, in the sense that it is controlled by decisions affected by a random bias, injected in order to improve the performance, and *probabilistic*, in the sense that its performance is a random variable, with a favorable distribution.

The basic idea is sharing a single token within a given membership of agents. The distributed algorithm used to control token sharing must ensure that, with high probability, exactly one token is present in the system, and that all *peer agents* hold the token a number of times that, in the long run, converge to the same value. We obtain such result moving at each step the token to another member chosen at random within the membership, thus implementing a sort of random walk.

The random variable that is representative of the performance of the algorithm is the *return time* of the random walk: in Jonasson (1998) the authors prove that the distribution of token inter-arrival time on a peer is characterized by a small probability after a value that grows with $O(N \log N)$, where N is the number of agents in the system. We do not assume a fixed topology or a preliminary *overlay design* phase (as in Kwon and Byers (2003), aimed at multicast). We evaluate the performance of our algorithm in a full mesh that represents the transport level of the Internet. Formal results (see Jonasson (1998)) justify the claim that our algorithm may be of interest also in networks with an average degree comparable with $\log N$.

[1] we have adopted the same network capacity as in the Virgo experiment referenced above, but introducing an higher number of subscribers, to highlight scalability

The study of token circulation algorithms is one of the classical branches of distributed computing, and the literature about that topic is overwhelming: of particular interest is Dijkstra (1974), that in two pages exactly frames the problem and gives a cornerstone solution. Our solution exhibits strong relationships with such *self-stabilization* approach: however, instead of using the deterministic knowledge of neighbor's state, we enforce mutual exclusion using time constraints computed locally. We share with some *randomized* self-stabilizing algorithms the basic idea of performing random moves in order to compensate lack of information. The closure requirement (in a legal state, the application of the algorithm brings to another legal state) may be broken as a result of the application of randomized rules, either generating spurious tokens, or removing the token.

The application of random walks to the problem of token circulation is infrequent. This is probably due to the interest for a deterministic solutions of the problem, where the existence of spurious tokens corresponds to a failure. Therefore sophisticated techniques are used to recover from this event, without incurring in its generation. To this purpose, the maintenance of an overlay topology is often introduced, like in Chen and Welch (2002).

Our approach may be regarded as an evolution of Israeli and Jalfon (1990): with respect to that work, we break the *closure* requirement, which states the deterministic impossibility to produce a spurious token, and we introduce it as a low probability event. In contrast, we introduce a rule to remove spurious tokens that is more efficient of the one introduced in the above reference, as discussed later on.

In the same spirit, Thibault et al. (2004) introduces a randomized technique to circulate a token in a highly dynamic network composed of mobile agents. The authors make use of timeouts (as in Gouda and Multari (1991)) to detect a token loss event, and a flooding mechanism to avoid the generation of spurious tokens. In order to control the flooding operation, an overlay tree network is maintained, using an adjacency table contained in the token itself. The solution we propose does not make use of broadcasts to prevent the creation of spurious tokens: instead, we rely on an efficient rule based on local knowledge in order to remove them. In addition, the token does not carry any data, except its identifier.

In Malpani et al. (2001) authors discuss and compare non-probabilistic algorithms that circulate a token in a group of mobile nodes: as in the former citation, the paper addresses mobile networks, and is pervaded by routing considerations that are peculiar to that case. In our work we mainly take advantage of the adaptability of the circulating token paradigm in order to tolerate with minimal overhead a number of adverse events that are typical of a distributed environment, like the switch-off of one of the peers, but we consider that all neighbors are reachable at equal cost, and we do not need to take any record of system topology.

The algorithm introduced in this paper shares with all peer-to-peer algorithms the reliance on the existence of a membership, a number of processes that loyally execute the same algorithm. Such membership is dynamic, in the sense that new members may join, and others may leave, while the algorithm is running. The concept of membership is another pillar of distributed computing, and we do not introduce a new solution to this problem. Instead, here we give a set of requirements for a solution that applies to our scenario, and some literature that addresses the problem in a more or less suitable way.

Synthetically, the requirements are the following:

- a member knows a $O(\log N)$ number of other members, in order to perform the randomized routing of the token;
- a member behaves according to the token passing protocol;
- a member utilizes the shared resource only if it holds the token;

The first problem is largely addressed in literature: see Ganesh et al. (2003) for a solution that meets our scenario. As for the other two, one solution is to control the access to the group, and assume that the admitted members either behave loyally, or are banned from the membership. To take advantage of this assumption, both token passing operations and resource utilization must be authenticated: see Challal and Seba (2005) for a survey on the subject.

However, we observe that known techniques that address a secure membership overkill our quite simple instance: ensuring a reliable and secure token passing operation. They turn out to be more expensive, in terms of resource utilization, than the token passing algorithm itself. Therefore, we have studied and presented in Ciuffoletti (2007) a secure group membership protocol whose performance and cost are adequate to the issue presented in this paper. Since its features are relevant for the applicability of our work, we include the reference as a further reading.

The next section of this paper proceeds with the description of the algorithm: we define the relevant parameters and discuss its behavior in the *stable* case, i.e. in absence of exceptional events. Next we introduce the token loss detection event, and finally the token removal event. All such events are triggered under probabilistic assumptions, which makes an analytic evaluation of the algorithm awkward. So we opt for a simulation analysis using the parameters of our case study, as summarized in section 3. Since none of the referenced works addresses the scenario we propose, we have opted to compare our solution against a uncoordinated operation.

2. System model and the wandering token idea

The system is composed of a set of N *peer agents*, whose clocks are loosely synchronized, interconnected by a complete mesh of *links*: for each couple of agents (c_i, c_j) there is a link $l_{i,j}$ that connects them, as in a *transport level* view of the Internet.

The resource sharing problem is defined by two parameters: N_{max} the number of agents that saturates the resource and Δ_{op} the time during which access is granted to the resource, once the agent holds the token.

The solution we propose is the probabilistic self-stabilizing algorithm described in figure 1: line numbering will be used to illustrate the pseudo-code.

Let us examine the stable behavior first, when there is exactly one circulating token. In that case the behavior of an agent consists of receiving the token (see line **8** in the figure 1), performing an action associated to the presence of the token (**18-26**), and passing the token to a randomly selected peer (**27**). The associated action consists in a simple delay of Δ_{skip} seconds in case the agent already performed the protected operation less than Δ_{min} seconds ago (**21**); otherwise the agent holds the token for a time Δ_{op}, while the protected operation is performed (**24**). We assume Δ_{skip} to be significantly smaller than Δ_{op}. Given N_{max} and Δ_{op} we compute (**2**) a reasonable value for Δ_{min} as

$$\Delta_{min} = \frac{\Delta_{op} * N_{max}}{2}$$

which is half the access period that would saturate the resource. Such simple rule of thumb is appropriate in many cases.

A token loss event, which has a probability that is significantly reduced by a 4-way token passing protocol illustrated in the companion paper Ciuffoletti (2007), breaks the *stable* behavior. The *token regeneration rule* (**29-33**) is triggered when the agent does not receive one within a timeout that is obtained incrementing Δ_{min} of a random quantity (**8**). A randomized rule guarantees the absence of *synchronization* effects that might degrade the performance. To this purpose, the Poisson distribution is regarded as a convenient candidate.

The $\gamma_{generate}$ parameter corresponds to the γ parameter of such distribution, and a reasonable value is (**3**):

$$\gamma_{generate} = \Delta_{min} * N_{max} = \frac{\Delta_{op} * N_{max}^2}{2}$$

If we *rescale* such distribution in order to have N_{max} events per time units, we obtain a distribution with an inter-arrival time of Δ_{min} time units. Therefore, in our system, where N_{max} agents run in parallel, the timeouts will expire, on the average, every $2 * \Delta_{min}$ time units, which corresponds to the requested access period and is considered as a reasonable setup. Although the value of this parameter influences the behavior of the algorithm, significant variations do not modify its basic properties in a given environment.

The token generation rule does not exclude that a new token is created even if the old one is not really lost: in that case, such rule may induce the simultaneous presence of multiple, but distinct, tokens in the system. Therefore the token generation rule, which is introduced in order to recover from an unlikely token loss event, most times has the effect of disrupting the stable property by introducing *spurious* tokens.

In order to remove spurious tokens, we apply to a *token removal rule* (**12**): for this we require that tokens are timestamped when they are generated, using a coarse grain clock (**30**). The agent discards a token with id x when two conditions hold (**12**): i) the token was already received in the past at time T_{last} and ii) another token with lower timestamp was received after time T_{last}. Visually, the three tokens of which one is hold form a sandwich, and the agent silently discards the token it holds (**15**).

Such rule is justified considering that if an agent receives a token with a timestamp lower than a previously observed token y, it can conclude that token y is spurious. It does not have any convenient way to remove token y at once, since it has been already passed elsewhere, but, the next time it observes token y, it will have a chance to remove it, and nobody

```
1  comment: Compute algorithm parameters
2  Δ_min = Δ_op * N_max/2
3  γ_generate = Δ_min * N_max;
4  lasttoken = {timestamp = 0, id = NULL}
5  while (true)
6        do
7        comment: Receive token or trigger regeneration timeout
8        select(receive(token), Δ_min + poisson(γ_generate))
9        if (defined(token))
10         then
11               comment: Apply sandwich token removal rule
12               if (∃i, j, i < j ∧ history(j).id = token.id ∧ history(i).timestamp < token.timestamp)
13                 then
14                       comment: Silently remove the token
15                       discard(token)
16               fi
17               comment: Decide whether to execute the protected operation
18               if (time − (lastaccess.timestamp)) ≤ Δ_min
19                 then
20                       comment: Just skip an early token
21                       sleep(Δ_skip)
22                  else
23                       comment: Execute protected operation
24                       execute(A)
25                       lastaccess = {timestamp = time, id = token.id}
26               fi
27               send(token)
28           else
29               comment: On timeout, generate a new token
30               token = {timestamp = localclock, id = newid()}
31               execute(A)
32               lastaccess = {timestamp = time, id = token.id}
33               send(token)
34       fi
35       push(history, token)
36 od
```

Fig. 1. The wandering token algorithm

in the system might have removed token x as a consequence of the existence of token y. We understand that timestamps are not required to be accurate: in case two tokens have inconsistent timestamps, the application of the *sandwich* rule will remove the one generated before, instead of the other. This fact has no side effects on our protocol, so we conclude that, in principle, timestamps could be generated randomly.

The *sandwich* rule has two minor weaknesses. One is that the removal operation has a latency that corresponds to the inter-arrival time of the token on a given node (or the *return time*), which is of the order of $2 * \Delta_{min}$ Lovasz (1993): during that time the state of the system is not legal, and simultaneous accesses occur. The other is that token x above, in the meanwhile, might be lost: in that case token y, although generated as a spurious token, might have become the new unique token. Such drawbacks have a minor impact on system operation, and do not diminish the practical interest for the algorithm: they indicate directions for its improvement, further reducing their probability to occur.

The problem of token elimination is well studied in theory, and is often referred as a solution to the leader election problem (see Bshouty et al. (1999)). However our setting discourages a formal approach for the validation of our proposal: a complex random process controls both token generation, and token collision (or *meeting*). These two facts make smart theoretical results, that are based on an initial population of tokens, and on exact collision of tokens for token elimination, useless for our purpose. However, we note that, with respect to Israeli and Jalfon (1990), the probability of collision is augmented by widening the *collision window* so that recovery is substantially improved.

3. Simulation results

The simulation results summarized in this section reflect the case study described in the introduction: agents of our algorithm correspond to *subscribers* that require the availability of $650KBps$ over a $200MBps$ channel, fairly distributed in time. From the definition of the problem we derive that the system supports approximately 300 subscribers (N_{max}). We assume each subscriber is granted exclusive access to the channel for a time slot of a fixed size, that corresponds to Δ_{op} seconds, set to 4: this guarantees a limited return time, which limits the buffering on the server to the order of the GBytes. In case the token recently visited the same subscriber, the token will be released after Δ_{skip} seconds, set to 100 msecs, which is consistent with typical network performances.

We carried out a series of simulations using a simple (a few hundreds Perl lines) *ad hoc* discrete event simulator, which is available upon request. Each simulation lasted 10^5 seconds, corresponding to approximately one day operation. In order to simulate network unreliability, we injected token loss events every 10^4 seconds. We do not simulate variable durations of the token passing operation, which is assumed to be negligible with respect to Δ_{skip}, the minimum time a subscriber holds the token.

To have a sort of reference, we also introduce a solution to the problem that does not use any form of coordination: each subscriber issues a service request randomly. The interval between two successive requests from a given subscriber is $4 * 300 = 1200$ seconds (equal to the average return time in the token based simulation), incremented by a random bias, chosen in the interval $[-600, +600]$, that breaks synchronous behaviors.

Observing simulation results summarized in figure 2 (dashed line only), we understand that such algorithm is a *low end* solution to the *soft mutual exclusion* problem: in fact the number of seconds during which more than one protected operation is running falls exponentially with the number of simultaneous operations. However it is not applicable as a solution to our case study: the share of time when the resource is idle is 40% (abscissa 0 in figure 2), while during 8% of the time more than two subscribers are simultaneously active, thus falling in the "delivery not guaranteed" region.

The simulation of a system controlled using the wandering token algorithm requires the definition of two further parameters: Δ_{min} and $\gamma_{generate}$, which are set according to the formulas given previously. Their values are summarized in table **??**.

N_{max}	300	peer agents	from case study
Δ_{skip}	0.1	seconds	from case study
Δ_{op}	4	seconds	from case study
Δ_{min}	600	seconds	$(\Delta_{op} * N_{max})/2$
$\gamma_{generate}$	$180 * 10^3$	seconds	$\Delta_{min} * N_{max}$
γ_{loss}	$10 * 10^3$	seconds	mean time between packet loss events

Table 1
Parameters used in the simulation

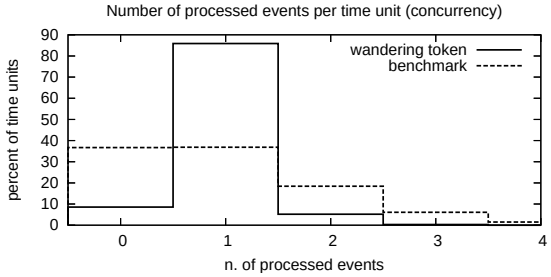

Fig. 2. Benchmark algorithm vs. wandering token: distribution of the number of concurrent operations for memberships of 300 (full load) subscribers (simulation lasted 10^5 time units, corresponding to seconds in our use case)

The comparison with the benchmark solution is clearly favorable, as shown in figure 2: the system controlled with the wandering token is idle less than 10% of the time (only due to network unreliability), while exhibiting 0.3% percent of the time (below figure resolution) with more than two subscribers concurrently downloading a chunk of data. The extra-billing zone (exactly 2 concurrent downloads) takes 5% of the time.

Another relevant parameter to evaluate the quality of our solution is the distribution of the time between successive accesses to the resource, which roughly corresponds to the inter-arrival time of the token.

In the case of the benchmark algorithm this is uniformly distributed between 600 and 1800 seconds: therefore the server can reliably flush data older than 1800 seconds.

In the case of the wandering token algorithm the evaluation is more complex, since the token inter-arrival time is ruled by a non-deterministic law. In figure 4 we see that 80% of the times the token interarrival time falls below $1200secs$, but the tail extends far after the 1800 seconds, which means a larger buffer in the source than in the uncoordinated

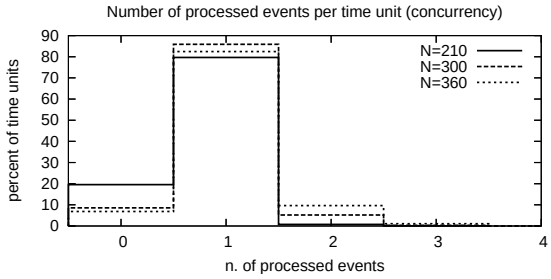

Fig. 3. Wandering token: distribution of the number of concurrent operations on a shared resource for membership size from 210 to 360 (simulation lasted 10^5 time units, corresponding to seconds in our use case)

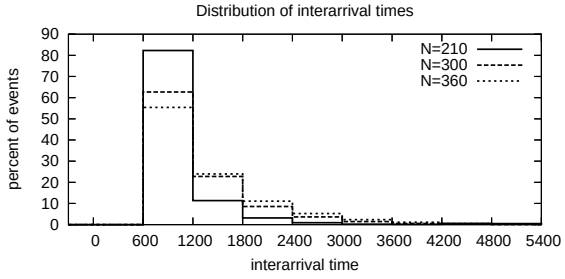

Fig. 4. Wandering token: distribution of intervals between successive firing of the protected operation (simulation lasted 10^5 time units, corresponding to seconds in our use case)

case. In our use case this is not a problem, since the data source is ready to hold TBytes of historical data.

It is interesting to see how such distributions changes when the number of subscribers does not correspond exactly to N_{max}. In figure 3 and 4 we observe that figures change smoothly varying the number of subscribers from 70% to 120% of N_{max}: the probability of concurrent access (in figure 3) does not exceed 10%, and the inter-arrival time (in figure 4) in case of overbooking, tends to have a longer tail, although more than 50% of the inter-arrival times are below 1200 seconds.

The stability of the algorithm is illustrated in Figure 5, where we see how the number of tokens varies during a simulation. We injected a token loss event every 10000 seconds (3 hours), while the system supports the nominal number of subscribers. The frequency of token loss events used in our simulation is more than ten times higher than that observed in the open Internet: further, a reliable token passing protocol has been developed Ciuffoletti (2007) that reduces the average frequency of loss events to approximately one week, using the same scale of our case study.

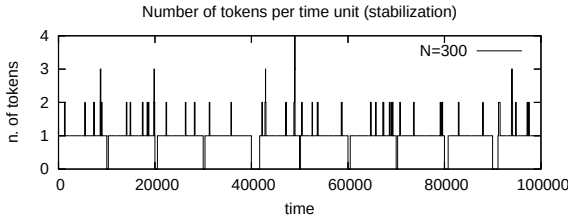

Fig. 5. Wandering token: number of tokens in the system during a simulation (full load)

Simulation results show that the algorithm promptly recovers from the presence of spurious tokens, indicated by narrow positive spikes; spurious tokens never exceed the number of two. In case the token is lost, the latency before its regeneration may be relevant: this justifies our efforts in the design of a reliable token passing protocol.

Based on the above results, we can figure out the behavior of the system in our use case. As long as the number of subscribers is N_{max} or less, concurrent access of more than two subscribers occurs during less than one percent of the time, and 80% of the times the applications have access to the backbone within $2 * \Delta_{min}$. When the number of subscribers grows over resource saturation, the chance of concurrent execution increases, but the event that more than 2 data transfers are occurring simultaneously is rare. The application is aware of the problem, since it is able to measure token inter-arrival times, that will increase linearly with the number of subscribers.

4. Conclusions

The *wandering token* algorithm is proposed as a solution for an architecture where moderating the concurrent access to a shared resource can improve performance. Its cost, in terms of communication and computation, is negligible.

The algorithm is fully scalable: the algorithm does not induce any bound on the number of agents exchanging the token. When such number overtakes the capacity of the shared resource, the *wandering token* algorithm gradually reduces the resource share granted to each agent, thus shielding the shared resource from the consequences of the overload.

The solution presented in this paper is strongly related to the provision of two other services: a reliable and secure token exchange mechanism, and the maintenance of a trusted membership. The risk of overkilling such problems, introducing algorithms that are more expensive than the resource sharing

protocol is real. We have studied such problems, and presented our results in Ciuffoletti (2007).

In order to give a intuitive support to our presentation, we have created, starting from a real application, a use case. The algorithm is currently proposed as a solution to a quite different case: the maintenance of a distributed directory of host capabilities in a Grid environment Ciuffoletti and Polychronakis (2006). We have considered more helpful the simple use case presented in this paper.

References

Allcock, W. and Perelmutov, T. (2005). Gridftp v2 protocol description. Technical Report GFD-R-P.047, OGF - GridFTP WG.

Bshouty, N. H., Higham, L., and Warpechowska-Gruca, J. (1999). Meeting times of random walks on graphs. *Information Processing Letters*, 69(5):259–265.

Buskulic, D. (2002). Data analysis software tools used during virgo engineering runs, review and future need. In *8th International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT 2002)*, page 6, Moskow.

Challal, Y. and Seba, H. (2005). Group key management protocols: A novel taxonomy. *International Journal of Information Technology*, 2(1):14.

Chen, Y. and Welch, J. L. (2002). Self-stabilizing mutual exclusion using tokens in mobile ad hoc network. Technical Report 2002-4-2, Texas A-M University – Dept. of Computer Science.

Ciuffoletti, A. (2007). Secure token passing at application level. In *1st International Workshop on Security Trust and Privacy in Grid Systems*, page 6, Nice.

Ciuffoletti, A. and Polychronakis, M. (2006). Architecture of a network monitoring element. In *CoreGRID workshop at EURO-Par 2006*, page 10, Dresden (Germany).

Dijkstra, E. W. (1974). Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644.

Ganesh, A. J., Kermarrec, A.-M., and Massouli, L. (2003). Peer-to-peer membership management for gossip-based protocols. *IEEE Transactions on Computers*, 52(2).

Gouda, M. and Multari, N. (1991). Stabilizing communication protocols. *IEEE Transactions on Computers*, 40(4):448–458.

Israeli, A. and Jalfon, M. (1990). Token management schemes and random walks yield self stabilizing mutual exclusion. In *Proceedings of the Ninth Annual ACM Symposium on Distributed Computing*, pages 119–129, Quebec City, Quebec, Canada.

Jacobson, V. (1988). Congestion avoidance and control. In *SIGCOMM ' 88*, Stanford (CA), USA. ACM.

Jonasson, J. (1998). On the cover time of random walks on random graphs. *Combinatorics, Probability and Computing*, (7):265–279.

Kwon, G. and Byers, J. (2003). ROMA: Reliable overlay multicast with loosely coupled TCP connections. Technical Report BU-CS-TR-2003-015, Boston University.

Lovasz, L. (1993). Random walks on graphs: a survey. In Miklos, D., Sos, V. T., and Szonyi, T., editors, *Combinatorics, Paul Erdos is Eigthy*, volume II. J. Bolyai Math. Society.

Malpani, N., Vaidya, N., and Welch, J. (2001). Distributed token circulation in mobile ad hoc networks. In *Proceedings of the 9th International Conference on Network Protocols (ICNP)*, page 8, Riverside, California.

Thibault, B., Bui, A., and Flauzac, O. (2004). Topological adaptability for the distributed token circulation paradigm in faulty environment. In Cao, J., editor, *Second International Symposium on Parallel and Distributed Processing and Applications – Hong Kong (China)*, number 3358 in Lecture Notes in Computer Science, pages 146–155. Springer.