Collision avoidance for Delay_Req messages in broadcast media

Augusto Ciuffoletti
University of Pisa
Italy

***Abstract*—The time accuracy of the Precision Time Protocol deteriorates in consequence to Delay_req/Delay_resp session collisions common for applications using shared broadcast media. In this paper we propose a protocol that coordinates Delay_req/Delay_resp sessions with minimum changes to the original PTP protocol. Simulations illustrate protocol's operation and demonstrate significant reduction of session collisions.**

I. Introduction

The IEEE-1588 protocol is a clock synchronization protocol applicable in a wide range of environments [5]. We focus on the avoidance of collision events that may arise between two-way sessions, that are based on one pair of Delay_Req/Delay_Resp packets.

These sessions are run periodically to compensate random variations in clock frequency: once an upper bound of these variations is given, it is possible to compute the time between two successive two-way sessions on a given slave clock.

One relevant feature of two-way sessions is that they are launched by slave clocks. The IEEE-1588 protocol does not envision any sort of coordination of two-way sessions which therefore, under certain operational conditions, may bring to collisions between sessions launched by distinct slaves. Such collisions are considered as undesirable events: they induce clock jitters and generally degrade the performance of the system [7].

The IEEE-1588 contains a norm which is meant to break unwanted correlations between two-way sessions: the period between two-way sessions is decremented of a random amount, thus backing up a possible tendency to aggregation in time. The application of this norm guarantees that two-way sessions are uniformly distributed in time, but does not introduce any form of coordination.

The property of uniform distribution in time corresponds to a Poisson distribution of two-way sessions in the whole system; the number of events is determined by the period between two way sessions on a slave clock (that we indicate with δ), and by the number of slave clocks (that we indicate with n). During a time unit, we observe $\lambda = \frac{n}{\delta}$ two-way sessions, where λ is the rate parameter of a Poisson process.

When the time required to complete a two-way session (that we indicate with τ) is a significant fraction of the average time between events in the process P, the occurrence of a collision becomes significant in its turn. Multiple collisions are also possible, meaning that more than two sessions can be simultaneously active at the same time.

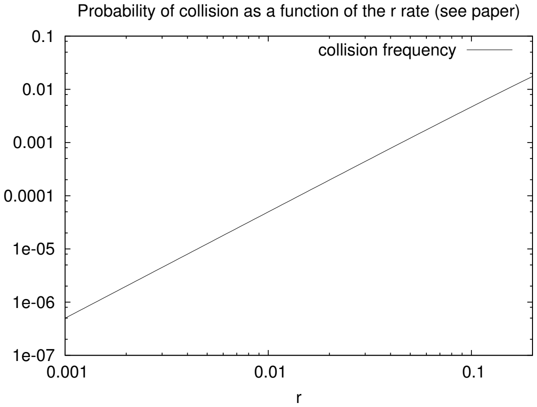

Fig. 1. Probability of collision as a function of $r = \frac{n\tau}{\delta}$ (logarithmic scale on both axis)

With a more formal statement, the probability of observing k overlapping sessions for a process with intensity λ corresponds to the well-known expression:

$$P([N(t+\tau) - N(t)] = k) = \frac{e^{-\lambda\tau}(\lambda\tau)^k}{k!} \qquad (1)$$

In figure 1 we see the dependency of the probability of a collision from the rate $r = \frac{n\tau}{\delta}$:

$$P_{coll} = 1 - (1+r)e^{-r} \qquad (2)$$

This figure justifies the claim that a collision avoidance protocol is bound to the r parameter: when r is high, collisions occur more frequently. We observe that the definition of r implies that it tends to increase with the size of the system and with the required precision. These are the distinguished features of potential successful use cases for our protocol. We do not want to go further in the direction of describing potential use cases, or restrict the scope of this paper to one specific scenario: even simpler ones depend on a number of external conditions in order to be exhaustively justified, and other may emerge in not yet experienced conditions. We just want to introduce the basic principles of operation of a protocol, and justify the reasonableness of the underlying idea.

The solution we propose specifically targets a broadcast media: in a different environment the collision problem does not emerge. We therefore feel authorized to exploit broadcast features: this turns out to be extremely convenient in the coordination of a distributed protocol.

The solution consists of an extension of the IEEE-1588 Precision Time Protocol: the extension significantly reduces collision events between two-way sessions, and simplifies their management when they occur.

The protocol that supports these additional features is extremely robust, and preserves the resilience of the PTP. We exploit as far as possible decentralized control techniques, that make the protocol resistant to slave and master failures. In case of failure of the master, the protocol may continue a degraded operation waiting for the master to resume.

II. A coordination protocol based on token circulation

We aim at enforcing system-wide mutual exclusion in the execution of two-way sessions. The algorithm we envision is based on a *token circulation* mechanism [4]: each slave clock that executes a two-way session grants the privilege of executing the next two-way session to another slave. Since communication is supported by a broadcast medium, each token passing operation is observed by all slaves and by the master clock.

The basic idea is extremely simple, but must be complemented with a number of additional features, in order to be applicable. We divide the explanation of our instance of token circulation in three steps:

- the algorithm that is run while there is exactly one slave holding the privilege (*stable operation*) and
- the actions undertaken when a new slave clock enters the system, or leaves the system as a consequence of a failure (*join and leave*) and
- the implementation of the *token passing* operation.

The explanation of how the token passing operation is implemented is delayed to the last step; we start considering that the performance of a token passing operation is comparable to the delivery of a message from the sender to the receiver: we will see later that, in fact, there is not a message dedicated to this operation.

During *stable operation*, in order to ensure a fair share of synchronization opportunities, we need to overlay a subnetwork used for token circulation: a number of deterministic solutions that enforce an overlay ring are found in the literature. However, they are very slow in recovering from failures. Here we propose a token circulation algorithm that is targeted on the peculiar features of our environment.

We observe that slave clocks do not need to run two-way sessions at regular intervals: the time between two successive executions may be variable, as long as a given maximum is not exceeded.

This opens the way to non-deterministic algorithms: the token may not visit all slave clocks in a deterministic sequence, but instead it is allowed to follow any route that satisfies certain requirements. The simplest rule consists of passing the token, at each round, to another peer chosen at random: such technique has been already applied to other environments [3] the time between two following visits of the token to a given slave clock, the *return time* of the token, has favorable properties, even in case failures occur.

The return time of the token to a given slave clock is of paramount importance for our application: it is a stochastic variable that is found describing *random walk* models [6]. Since at each token passing operation, each slave clock has an identical probability $\frac{1}{n}$ to be selected as the destination of the token, the number of two-way sessions in the system between two successive visits of the token has an exponential distribution with a mean of n (here we approximate the geometric distribution of Bernoulli trials with an exponential since n is assumed to be large). If two-way sessions occur at regular intervals, every τ time units, the return time has exponential distribution with mean $n\tau$.

The randomized token routing algorithm introduces an assumption that is not realistic in a distributed system: in fact, in order to select another slave clock *at random*, the sender should access a registry of all clock identifiers. The access to a shared resource would re-instantiate a mutual exclusion problem, while the maintenance of a local cache, in principle viable, is impractical since the cache might contain an unpredictable number of slave identifiers.

A viable solution consists in maintaining a local cache but of limited size: it is a known result that the *random walk* properties (included the distribution of the *return time*) are preserved when the graph is not complete, but neighbors are selected at random.

Distributed algorithms that maintain a list of $k \ll n$ neighbors selected at random has been recently investigated with reference to *ad hoc* networks [2]. Our environment being simpler to manage, we propose a simplified algorithm.

In essence the algorithm, which is run by a slave during a two-way session, consists in replacing one of the identifiers in the local list with the identifier of the slave that executed a two-way session m times ago, the m parameter being around 10 for very large networks. The slave clocks will keep updated a FIFO containing the m most recently visited slave units identifiers by observing token passing operations in the network.

The algorithm above does not ensure an upper limit to the return time: for any Δ the probability of a return time larger than Δ never goes to zero. Instead we need to ensure that each clock is synchronized at least every δ time units.

We consider that an effective way to cope with this deterministic requirement consists in exploiting a global view of the system, which is maintained on the master clock: whenever one of the slave clocks appears to have been waiting δ time units, the master *re-routes* a token to feed the starving slave. We will see later on, when explaining token implementation, how this is obtained.

The above rule also helps compensating an issue of the neighbors list construction algorithm. A basic requirement is that each peer has identical probability to appear in one list; however, this property is exposed to a progressive degradation: identifiers that are poorly represented in the lists for stochastic reasons have less chances to advertise themselves, and are progressively excluded from the system.

A disadvantage of the *token re-routing* technique is that the utilization of centralized knowledge, in our case located in the master clock, partially degrades the *distributedness* of

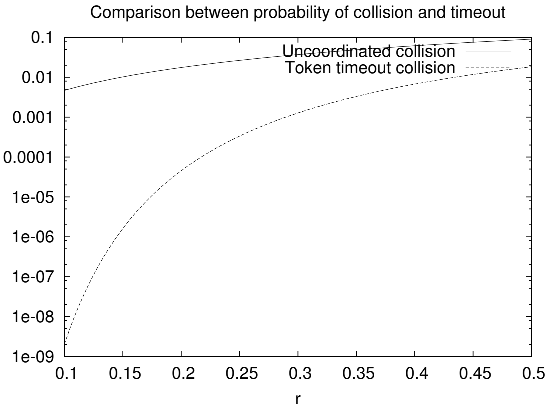

Fig. 2. Comparison of probability of collision as a function of $r = \frac{n\tau}{\delta}$ (logarithmic scale on y axis)

our solution: this negatively impacts fault tolerance features, although this impact is limited to the occurrence of timeout events, when the centralized knowledge is used to introduce a re-routing operation.

It is therefore of interest to quantify the frequency of these events. And, since the delay between successive visits has exponential distribution, the probability of a timeout (during a two-way session) is:

$$P(w > \delta) = e^{-\lambda\delta} = e^{-\frac{\delta}{n\tau}} \tag{3}$$

A collision occurs when two or more timeouts are triggered during the same interval of width τ. In that case one of the two has to be postponed, likely to the successive token passing operation. We compute a first order approximation of this event as:

$$P_{coll} = P(w > \delta)^2 = e^{-2\frac{\delta}{n\tau}} \tag{4}$$

We are now at the point of being able to compare from an analytic point of view the coordinated and the uncoordinated operation, though the nature of the critical events in the system for which we have analytically studied the frequency is slightly different. In the case of the uncoordinated network, collisions may deteriorate clock synchronization in an hardly predictable way. In the case of the application of our token based solution, collisions are rapidly compensated after a short lapse, incurring in the worst case a limited inaccuracy of a clock.

Also the comparison between the two probabilities is in favor of the token passing scheme as clearly shown in Figure 2: the probability of collision during a two-way session for the token passing scheme is orders of magnitude better than that of the for an uncoordinated operation. However the implementation of a token passing scheme adds complexity: here following is a summary of its operation on each slave:

```
History, Neighbors: stacks;
forever
  wait TwoWaySession;
  observe TwoWaySession
    passing token from J to I;
  push J into History;
  if ( LocalId == I )
    push History[10] into Neighbors;
    Next=random(Neighbors);
    perform TwoWaySession
      passing token from I to Next;
```

The algorithm is executed each time a new two-way session is observed. Such operation entails a token passing operation, and each slave in the system records the source and destination of the token. The source of the token is stored in a local stack (History). If the destination of the token corresponds to the local identifier the operation continues pushing the last element of History into the local cache of Neighbors. Next one Neighbors is selected at random as the destination of the token, and the two-way session is executed.

The master clock executes a distinguished algorithm:

```
WaitTime[Slave]: array;
forever
  wait TwoWaySession;
  observe TwoWaySession
    passing token from J to I;
  if ( (T=max(WaitTime)) > delta )
    I = select Slave with WaitTime=T;
  WaitTime[I] = 0;
  perform TwoWaySession
    passing token from J to I;
```

As a general rule it is transparent (i.e. stateless) with respect to the token passing operation: only when one of the slaves is starving, its identifier is replaced as the destination of the token, using information stored in the master clock.

A. Other operations

We consider two separate events: the *leave* event, when the peer that receives the token does not show signs of life, and the *join* event, when a new peer enters the membership. We note that both of them are equally observed by all peers, and this greatly simplifies the task of implementing a consistent response on each slave clock.

The *leave* event can be effectively coped with removing the identifier of the leaving peer from all lists, and by re-sending the un-received token.

The *join* event may be implemented using a *bullish* approach: as soon as a new slave clock joins the membership, it seizes the token. This is performed with the intervention of the Master, which re-routes the token to the joining slave. In addition, the Master will provide the joining slave with a random neighborhood. This operation, as well as the notification of the new slave, is assumed to occur in the background, using a communication channel distinct from the one used for the IEEE-1588 protocol.

With these two simple rules in place, the event of token replication never occurs. However, in the unlikely case this event occurs for unforeseen reasons, the broadcast nature of the media simplifies the task of removing the tokens, leaving to the master the task of reintroducing only one.

III. Implementation of the a token passing operation

The *token passing operation* is implemented as part of the two-way session: the slave clock that sends the Delay_Req message includes in the message the identifier of a proposed destination of the token. All clocks in the network, included the master clock, observe both the identifier of the source and that of the proposed destination of the token. Only occasionally the master clock may alter the proposed destination of the token, by indicating a different one in the Delay_Resp message. This implements the re-route operation introduced above.

Note that the token is not represented by any sort of data structure, neither inside a slave clock nor in a piece of communication. The property of *holding the token* is simply embedded in the control flow of the slave clock. The token is (virtually) passed from the slave clock that sends the Delay_Req message to the slave clock indicated in the Delay_Resp packet.

We have a limited space available to embed a slave identifier in the IEEE-1588 messages of concern: 5 octets and one 4 bits nibble are reserved for future use in the common message header. We envision a solution based on the MAC address associated to slave interfaces. Our solution uses 3 octets in Delay_Req and Delay_Resp message, and one bit in the Delay_Resp, that we call *re-route flag*.

A MAC address is composed of 6 octets, that may be encoded in two different ways. In case the MAC is left in its factory settings (universally administered), the three most significant octets encode the manufacturer of the network card, and are not very useful to uniquely identify a card. The three least significant octets contain an identifier which is unique for a given manufacturer. Otherwise the MAC address can be rewritten with an identifier locally unique (locally administered).

We use primarily the least significant octets for our protocol: they are included in the Delay_Req message of the slave that sends the token. The master that observes the Delay_Req follows a more complex algorithm that we split into four cases, starting from the most frequent one:

1) *if no reroute is needed, and the three octets uniquely identify a slave* it must set to 0 the *reroute* bit, and may include the 3 most significant octets in the Delay_Resp.
2) *if no reroute is needed, and the three octets do not uniquely identify a slave* it must set to 0 the *reroute* flag, and must include in the Delay_Resp the 3 most significant octets of of the MAC address of one of those sharing the least significant three octets. A reasonable choice is the one of the least recently synchronized slave.
3) *if it reroutes the token* it must set to 1 the *reroute* flag, and must include the 3 least significant octets in the Delay_Resp 3 octets.
4) *if a collision is detected* it must send in sequence two Delay_Resp messages, respectively containing the most significant and the least significant triplets. We call this a *two-step recovery*.

Based on such algorithm, a unique destination is selected with high probability:

1) *if the reroute flag is 0* the two triplets in the Delay_Req and Delay_Resp form the MAC address of the destination;
2) *if the reroute flag is 1* only the starving slave or the joining one may be the target, and they know about their state.

A low level collision occurs in the case there are several slaves joining or starving, and they have identical least significant triplets. It is a quite marginal case, and the collision is equally observed by all slaves and by the master: they will engage into the *two step recovery*.

Note that, in the case of an organization that decides to override the MAC addresses assigned by the manufacturer, and assign instead *locally administered addresses*, the algorithm is simplified if the three least significant bytes uniquely identify a slave.

IV. Fault tolerance issues

The failure of a slave D is detected when the token is passed from S to D, but D does not perform the successive two-way session. In that case the same S will resend the token. If both S and D fail, the master will backup with a *two step recovery*.

The failure of the master is an issue only as long as its intervention is required to backup some of the infrequent events listed above. As a general rule, especially in a network with *locally administered addresses*, the failure of the master has no immediate effect on the token passing algorithm. However, in case the identifier in a Delay_Req is ambiguous, a collision will occur and the slaves will start waiting for a *two step recovery* from the master. In the long run, the system will suffer from the absence of rerouting activity: due to this fact, some slave may starve. In this case we envision a leave and re-join of the slave, which will complete when a new master becomes operational.

A master that is restarted with a damaged list of slaves should be provided at startup with at least one correct MAC address of a slave, in order to run the *two-step recovery*: the list of slave addresses will be partially recovered during the operation by observing the MAC addresses visited by the token. However the restart of some slave clock may be required.

The event of *noisy* failures should be addressed using tools not illustrated in this paper. In case of medium jamming, fault tolerance relies on fault containment features of the physical layer.

Malicious failures of a slave clock are easy to detect and can be ignored by the master clock and by other peers, but may cause collisions: also in this case, we envision a low level deactivation of the port serving the failed device.

V. Verification

The algorithm illustrated in the previous sections is rather detailed, and is therefore very difficult to verify its correctness. In order to give some evidence of its correctness, we have used a number of different techniques:

- in section II we give analytical results proving that the idea of randomly circulating a token to achieve coordination instead of detecting collisions is well founded
- in section III we make a *by case* analysis to explain how to encode slave identifiers into Delay_Req and Delay_Resp messages;
- in section IV we give a *by case* analysis to partially justify fault tolerance properties, but without evaluating statistical properties.

There is still one relevant detail that has not been conveniently covered: it is the algorithm used to implement the random graph. This algorithm is quite complex, and combines the replacement of one element in the cache during each two-way session with the effects of *token rerouting* in case of timeout of one slave.

On one side, our attempts to give an analytic model for the composition of the neighbor's cache have been unsuccessful. On the other, a *by case analysis* is prevented by the size of the input space.

Therefore we conducted a simulation of limited scope, aimed at exploring the effects of the overlay random graph: we concentrate on the basic figure of our token circulation algorithm, the *return time*.

The simulation has been run with the round-trip time τ corresponding to the simulation time unit, in a system of 1000 slave clocks, with variable values of δ. We used an *ad hoc* simulator written in Perl, and each simulation collects 10^5 return time samples. Internal queues for the management of the random neighborhood have 10 positions. The code of the simulator (about 100 lines) is available at `http://code.google.com/p/ispcs2009/`.

The first simulation is for a system with a $r = 0.2$ corresponding (with the above constants) to a value of $\delta = 5 * 10^3$ time units (figure 3). We observe that the distribution of the return time reasonably approximates the theoretical slope: the final spike at $5 * 10^3$ time units is due to the timeout on the return time. It corresponds to a probability of 1.6%, which is the rate of intervention of the timeout. The collision rate is 0.027%, which compares favorably with the 1.7% rate of the uncoordinated scheme, but is more than the expected 0.0045% from the theoretical model (see figure 2).

We conclude that the limited number of neighbors significantly deviates the right queue of the distribution of the return time from the expected distribution, that is computed for a complete mesh network. The resulting collision probability is nonetheless significantly better than the corresponding collision probability in the uncoordinated case.

If we select a smaller value for r, we move in a region where collisions are less likely to occur, and therefore the application of a coordination algorithm is less relevant. In figure (figure 4) we observe the results of the simulation for $r = 0.1$, corresponding to a δ of $10 * 10^3$ time units. The deviation from the full mesh distribution is more pronounced: the intervention rate of the timeout is 0.24%, with a collision rate of 0.0006%, instead of order of 10^{-9}. But the comparison with the uncoordinated collision rate, around 0.47%, is persistently favorable.

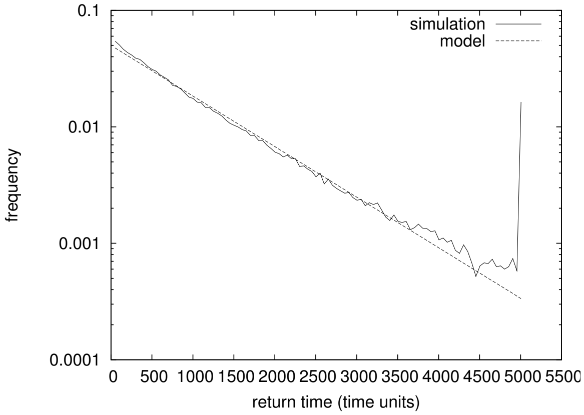

Fig. 3. Frequency of return time values for $r = 0.2$: comparison between model and simulation (logarithmic scale on y axis)

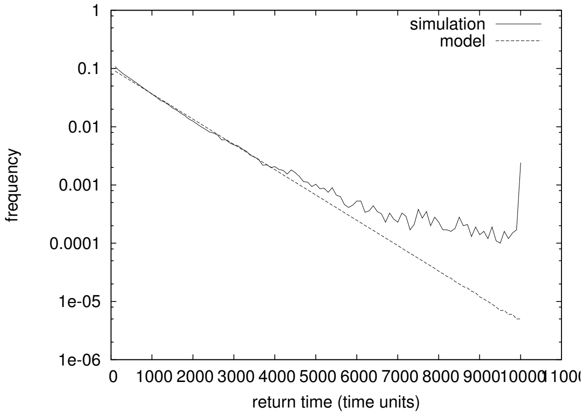

Fig. 4. Frequency of return time values for $r = 0.1$: comparison between model and simulation (logarithmic scale on y axis)

If we move in a region with higher collision rate, with a $r = 0.4$, we observe an opposite trend, but with similar results. Simulation results tightly approximate the full mesh, and the collision rate compares favorably with the collision rate without coordination: 1.5%, against 6.2%.

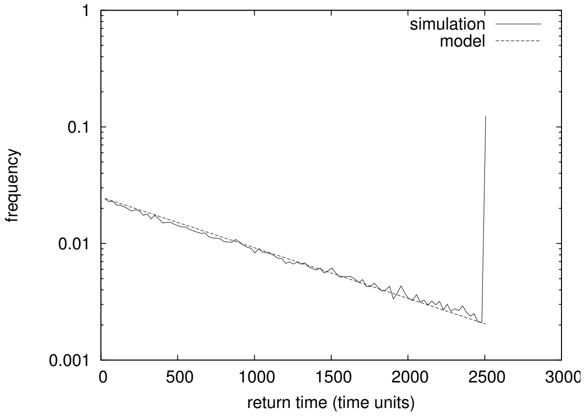

Fig. 5. Frequency of return time values for $r = 0.4$: comparison between model and simulation (logarithmic scale on y axis)

r	0.1	0.2	0.4
Intervention	0.2410%	1.6350%	12.3420%
Full mesh collision	$2.06E-9$	0.0045%	0.6738%
Token collision	0.0006%	0.0267%	1.5232%
Uncoord. collision	0.4679%	1.7523%	6.1552%

TABLE I
RELEVANT VALUES USED FOR EVALUATION

The figures above are summarized in Table I. The rows contain, for the value of r indicated in the first row, the percentage of two-way sessions that exhibit the named features:

- **Intervention**, the occurrence of token wait timeout;
- **Full mesh collision**, the simultaneous occurrence of more than one token wait timeout with a full mesh overlay network;
- **Token collision**, the simultaneous occurrence of more than one token wait timeout with the overlay graph generated by the algorithm;
- **Uncoord. collision**, exposure to frame collision in an uncoordinated broadcast medium.

We derive two conclusions from our discussion:

- the collision rate is lower using the token based algorithm. We recall that we did not take into account that the loss of accuracy produced by a low level collision is hardly predictable, while a collision using the token based algorithm delays of a few τs the execution of a two-way session.
- the approximation introduced by the *full mesh* hypothesis becomes imprecise for systems with a low collision rate.

The increment of the length of the queues used to represent the overlay random network improves the performance, since figures tend to get closer to the *full mesh* approximation.

VI. CONCLUSIONS AND APPLICATION NOTES

This paper proves that the introduction in the IEEE-1588 protocol of a token passing protocol for the coordination of two-way sessions is

- convenient, since it reduces the impact and the frequency of collision events and
- feasible, since its introduction does not alter the structure of protocol messages;

The interest for the protocol is justified by the existence of collisions among distinct two-way sessions: when the occurrence of this event is not an issue, either because its effects are irrelevant for the application, or because it is extremely infrequent, the introduction of our protocol is pointless.

However, we know that typical applications of the IEEE-1588 protocol have critical timing requirements, where inaccuracies due to collisions do matter, and the network is heavily used for application related data transfer. Some sort of traffic engineering solution (e.g. see [1]) may be introduced to avoid collisions between data traffic and IEEE-1588 packets, but the coordination of IEEE-1588 traffic remains an issue.

The paper identifies the variable that determines whether our solution is of interest or not: the r parameter, that represents the density of two-way sessions. A low value disencourages the introduction of a coordination, unless under extremely demanding reliability requirements.

When the introduction of a coordination scheme is of interest, we prove that our solution, attains two important targets:

- reduces and makes predictable the impact of a collision;
- significantly reduces the probability of collision.

We prove the above results using an analytical model, and introducing simulation where we are unable to give an analytic model.

We clearly consider the paper as a starting point. One direction for future research is the identification of relevant use cases: this task requires specific expertise in each of the numberless application fields of IEEE-1588. The presentation at the IEEE-1588 Workshop is a milestone on this way.

REFERENCES

[1] Astrit Ademaj and Hermann Kopetz. Time-triggered ethernet and IEEE 1588 clock synchronization. In *International IEEE Symposium on Precision Clock Synchronization (ISPCS)*, 2007.

[2] Ziv Bar-Yossef, Roy Friedman, and Gabriel Kliot. RaWMS - random walk based lightweight membership service for wireless a-hoc networks. *ACM Transactions on Computer Systems*, 26(2):66, June 2008.

[3] Augusto Ciuffoletti. The wandering token: Congestion avoidance of a shared resource. In Z. Nemeth P. Kacsuk, T. Fahringer, editor, *Distributed and Parallel Systems*, pages 3–12. Springer, 2007.

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

[5] IEEE Std 1588-2008. *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, July 2008.

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

[7] Chongning Na, Dragan Obradovic, Ruxandra Lupas Scheiterer, Guinter Steindl, and Franz-Josef Goetz. Synchronization performance of the precision time protocol. In *International IEEE Symposium on Precision Clock Synchronization (ISPCS)*, pages 25–32, 2007.