International Journal of Computer Applications (0975 - 8887)
Volume 43- No.21, April 2011

An Improved Byzantine Agreement Algorithm for Synchronous Systems with Mobile Faults

Nazreen Banu, Samia Souissi, Taisuke Izumi and Koichi Wada
Graduate School of Engineering,
Nagoya Institute of Technology, Gokiso-cho,
Showa-ku, Nagoya, Aichi, 466-8555, Japan.
Phone: +81-52-735-5438, Fax: +81-52-735-5408

Email: nazreentech@yahoo.co.in (or) nazreentech@phaser.elcom.nitech.ac.jp
{souissi.samia,t-izumi,wada}@nitech.ac.jp

ABSTRACT
We study the problem of Byzantine agreement in synchronous systems where malicious agents can move from one process to another and try to corrupt them. This model is known as mobile Byzantine faults. In a previous result [10], Garay has shown that $n > 6t$ (n is the total number of processes, and t is the number of mobile faults) is sufficient to solve this problem even in the presence of strong agents. These agents can move at full speed (in the sense that each agent can take a movement in every round) and can make corrupted processes forget that they run the algorithm (as a result, after recovery a process must learn the current state of computation including the code from other processes). Many following results [3] have improved the above result but with some additional assumptions such as a corrupted process must recover and learn the current state of computation before another process can fail instead of it. The question, whether the result of Garay can be improved without any additional assumption, remains open. In this paper, we answer this question by providing an algorithm MBA that works with $n > 4t$.

General Terms
Distributed Systems, Fault Tolerance Algorithms

Keywords
Synchronous Systems, Agreement(Consensus) Problem, Mobile Byzantine Adversary

1. INTRODUCTION
The *distributed consensus (agreement) problem* is a fundamental and important problem that has to be solved in designing fault-tolerant distributed systems. This problem can be stated informally as: how to ensure that a set of distributed processes achieve agreement on a value or an action despite a number of faulty processes. This problem is interesting both in theoretical and practical aspects. From theoretical point of view, the significance of the consensus problem derives from several other distributed systems problems being reducible or equivalent to it. Examples are atomic broadcast [6, 8, 9], non-blocking atomic commit [11], group membership [11] and state machine replication[18]. From a system perspective, replication of components and services is an important paradigm that can be employed for information protection in critical applications, and consensus plays a fundamental role in many replication algorithms. Some example solutions based on these ideas are: Bessani et al. [2] and Yin et al. [21] use replication to implement fault- and intrusion-tolerant firewall devices; Cachin et al. [4] and Castro and Liskvo [5], Chun et al. [7] and Veronese et al. [20] propose replication algorithms to implement highly resilient services, like data historians or DNS (essential for the Internet).

Algorithms that solve consensus vary much depending on the assumptions that are made about the system. This paper considers the systems that experience *Byzantine faults* [13], more destructive faults which do not put any constraints on how processes fail. Algorithms based on this system model are expected to work correctly no matter how faulty processes behave. Byzantine consensus algorithms are highly essential for practical systems to deal with malicious attacks or situations where faults are difficult to characterize.

The Byzantine consensus problem can be informally stated in terms of three properties: each process proposes a value, and the non-faulty processes have to decide (*Termination property*) on a

common output value (*Agreement property*) that has to be related to the set of input values (*Validity property*). However, in many situations, reaching a common agreement among the correct processes in the presence of moving malicious agents, such as mobile viruses, is highly required. This problem is referred to as the mobile Byzantine agreement(consensus) problem. Reischuk [15] worked on a model with malicious faults covering a fraction of the network, and designed a Byzantine agreement protocol which tolerates them as long as they remain immobile for a given interval of time. However, the model which we focus in this paper was initiated by Ostrovsky and Yung [14]. They used randomization and information theoretic security methods to provide mobile virus protection at run time. They worked on means to tolerate mobile viruses which can be viewed as transient faults. They used the notion of a mobile adversary. In the static adversary model, a constant fraction of processes may be faulty and the static adversary is not allowed to move faults once it has chosen a set of faulty processes. However, in the case of mobile adversary, the adversary is said to be infinitely powerful and can inject and distribute faults into the system at a constant rate in every round. Unlike its static counterpart the mobile adversary is allowed to change the position of the faults. We can assume that only a constant fraction of processes may be infected in a round. But, we also allow every process to be infected in some round. We also need the capacity to reboot the infected machine on detection of fault. Another requirement is the property that fault detection can proceed at the same rate as infection. This can be justified for a general setting by the results of Kepphart and White [12] . Ostrovsky et al. [14] were able to show that a weak fault detection capacity is sufficient to make computation robust in this scenario.

Garay [10] investigated the mobile fault environment for solutions to the problem of Byzantine agreement. He studied the power of disruption of malicious agents as a function of the speed with which they can traverse the network (called roaming pace). Specifically, the roaming pace ρ denotes the minimal amount of time (i.e., the number of rounds) that has to elapse between the time at which an agent leaves a process, and the time at which it starts to corrupt another process. For example, $\rho = 3$ means that an agent takes at least 3 rounds to hop from one process to another. In particular, Garay [10] proposed two protocols for solving Byzantine agreement problem in synchronous systems with mobile adversary. He assumed that at least one process remains uncorrupted for $O(n)$ rounds, since a discouraging impossibility result [16] proved that consensus is not solvable without restrictions even with a single mobile failure. The first protocol is for agents that move at full speed (i.e., $\rho = 1$). It requires $n > 6t$. The second protocol assumes $n > 4t$, but it deals with agents that move at half-speed (i.e., $\rho = 2$). These protocols run in $O(n)$ communication rounds.

Later on, Burhman et al. [3] proposed an optimal Byzantine consensus algorithm with $n > 3t$ for full speed agents. However, they added the assumptions that agents can move from one process to another only through messages (i.e., the migration of the agents is possible only during the send operations), and a faulty process must recover and learn the current state of computation before another process can fail instead of it. In a recent work, Biely et al. [1] proposed a mobile Byzantine agreement algorithm for partially synchronous systems for $n > 3t$, with the same assumption that a faulty process must recover and learn the current state of computation before another process can fail instead of it. However, this algorithm guarantees termination only when all faulty processes have recovered. In a different work, Schmid et al. [17] presented impossibility results and lower bounds on the required number of processes and rounds for synchronous systems under mobile link failure model.

The question that remains open is, can the result of Garay [10]($n > 6t$) be improved without adding any additional assumptions? In other words, what is the required number of processes to solve mobile Byzantine agreement problem in synchronous systems where malicious agents move at full speed and corrupt new processes before the previously corrupted processes recover?

This paper answers this question by providing an algorithm *MBA* that solves mobile Byzantine agreement problem in Garay's model with $n > 4t$.

The remainder of this paper is structured as follows. Section 2., introduces the system model and the mobile Byzantine agreement problem we are interested in. Section 3., provides our algorithm MBA that solves this problem under the requirement of $n > 4t$ and proves its correctness. Section 4. presents the discussion, and the conclusion is given in Section 5..

2. PRELIMINARIES

2.1 System Model

We consider a distributed system that consists of n processes numbered from 1 to n. Each process communicates with each other process by sending messages over a reliable link where neither message loss, duplication nor corruption oc-

curs. Our system is synchronous. This means that its execution is organized by a sequence of rounds during which each process can send messages to other processes, receive messages, and perform some local computation. Also, a message that is sent in some round is necessarily received within in the same round.

We assume that the system is interfered by a powerful computationally unbounded adversary which can inject up to t malicious agents into the system. These agents can move from processes to processes at full speed (means, it takes an agent at least one round to migrate from one process to another) and corrupt them in a dynamic fashion. In the worst case, they do erase the local memories of the processes. Because of the mobility of agents, any process can be corrupted during the course of the algorithm. However, we assume that at least one process remains uncorrupted for $O(n)$ rounds. Since each agent can corrupt one process at a round, the total number of corrupted processes in any round is at most t. A corrupted process may behave arbitrarily, which means that even it is allowed not to follow the deployed algorithm. In addition, we assume that a corrupted process can recover and rejoin the on-going execution after the corrupting agent left it. The semantics of rejoining is as follows: Let us assume that an agent leaves a process p in round r; Process p recovers in round $r+1$; After recovery, it learns the code and the current state of computation from other processes by receiving messages that were sent in round $r+1$; Then, p starts participating in the on-going execution from round $r+2$.

We refer the processes that are corrupted in the current round as *faulty or infected*, and the processes that were faulty in the previous round, but no longer as *cured*. Also, we use the term *correct* to refer the processes that are neither faulty nor cured in the current round.

2.2 Problem Definition

In mobile Byzantine agreement problem, each correct process p has an initial value v_p from the set $\mathcal{V}$ of all possible initial values, and decides a value v according to the following rules.

- **Termination:** Each non-faulty process eventually irreversibly decides a value v.
- **Agreement:** The non-faulty processes decide on the same value.
- **Unanimity:** If all non-faulty processes have the same initial value v, then no non-faulty process decides a value different from v.
- **Consistency maintenance:** Once agreement is reached among currently noninfected processes, it must be maintained among the (possibly different) noninfected processes.

Note that, any algorithm that solves the above problem is responsible not only for reaching the agreement, but for preserving the agreement among all correct processes forever. This is required, since even if agreement is reached at some point, the mobile agents can move to corrupt the correct processes and make the agreement disappear.

2.3 Notations

Let $\mathcal{V}$ be an ordered set of all possible proposal values. We introduce the default value $\perp$ such that $\perp \notin \mathcal{V}$ and $\perp < min(\mathcal{V})$. Let I be an vector in $(\mathcal{V} \cup \{\perp\})^n$. The number of occurrences of a value v in I is denoted by $\#_v(I)$.

3. AN IMPROVED MOBILE BYZANTINE AGREEMENT ALGORITHM

3.1 Algorithm *MBA*

In this subsection, we present a mobile Byzantine agreement algorithm **MBA** for synchronous systems. The algorithm is described in Fig. 1. It requires $n > 4t$ and at least one process remains uncorrupted for at least $3n$ rounds. This algorithm consists of phases, each phase is made of three rounds, namely, *Proposal round, Voting round,* and *Coordinator round* during which the processes exchange messages. Each message consists of values and a message tag (such as *PROP*, *VOTE* and *ECHO*) that indicates the name of the round in which it is sent. These messages also include the algorithmic code and the current state of computation, etc., that help the recovering processes to correctly reintegrate into the on-going execution, but they are omitted for the sake of clarity. Remember that, in our model the cured processes do not send messages but they can receive messages. The three rounds are detailed as follows, where in the description, just 'process' means correct or cured one.

- *Proposal round:* The aim of this round is to end up in a situation where there is a single value v ($v \neq \perp$), such that each process adopts either v or $\perp$. To attain this goal, each process sends a message $< PROP, val >$ to provide the other processes with its value *val* and stores the received values in vector PV. Note that, since cured processes (if any) are silent, the entries that correspond to them in PV contain $\perp$. A process p adopts a value v if only if it appears at least $n-2t$ times in PV_p and the sum of the number of occurrences of v and $\perp$ in PV_p is

3.2 Correctness

Algorithm MBA

```
1 :  val ← v
2 :  Begin
3 :    for all s from 1 to ∞ do
4 :      coord-accept ← True
                  ———— Proposal round of phase s ————
5 :      PV ← [⊥, ⊥, ..., ⊥]
6 :      Send (PROP, val) to all
7 :      for all i do: if a message (PROP, v) is received from process i then PV[i] ← v
8 :      if (∃v ≠⊥: #_v(PV) ≥ n − 2t and #_v(PV) + #_⊥(PV) ≥ n − t) then
9 :        val ← v
10 :     else
11 :       val ← ⊥
12 :     end if
                  ———— Voting round of phase s ————
13 :     SV ← [⊥, ⊥, ..., ⊥]
14 :     Send (VOTE, val) to all
15 :     for all i do: if a message (VOTE, v) is received from process i then SV[i] ← v
16 :     if (∃v ≠⊥: #_v(SV) > 2t) then
17 :       val ← v
18 :       coord-accept ← False
19 :     else if (∃v ≠⊥: #_v(SV) > t) then
20 :       val ← v
21 :     else
22 :       val ← ⊥
23 :     end if
                  ———— Coordinator round of phase s ————
24 :     EV ← [⊥, ⊥, ..., ⊥][⊥, ⊥, ..., ⊥]
25 :     Send (ECHO, SV) to all
26 :     for all i do: if a message (ECHO, sv) is received from process i then EV[i] ← sv
27 :     if I am cured then
28 :       Reconstruct()
29 :     coord ← s mod n;
30 :     if ( ∃v ≠⊥: #_v(EV[coord]) > t) then coord-val ← v else coord-val ←⊥ end if
31 :     if coord-accept = True then
32 :       val ← max{coord-val, min{V}}
33 :   end for
34 :  End
```

Figure 1: **Algorithm MBA**

at least $n-t$. Since any two sets of $n-t$ values will intersect at a correct process's value, it is ensured that there exists a single value v and all processes that do not adopt $\perp$ will adopt the same value v.

- *Voting round:* In this round, each process checks for the existence of a unique value v. If such a value v exists, it adopts v and chooses not to adopt the coordinator value in the next round. To do this, each process sends its value val in a VOTE message $< VOTE, val >$ to all processes and stores the received values in a vector SV. If more than $2t$ of the votes are for some value v, then it adopts v and chooses not to accept the coordinator value by setting its variable *coord-accept* as "$False$". Otherwise, if more than t of the votes are for v, it adopts v, else it adopts $\perp$. In these cases, it chooses to accept the coordinator value in the following round.

- *Coordinator round:* The aim of this round is to try to make all processes have the same value. In this round, each process echoes the votes that it has received in the voting round through an ECHO message $< ECHO, SV >$ in order to help the cured processes to reconstruct their variables. Since all correct processes send their vectors, each cured process can compute the correct values of its variables by reconstructing the previous voting round using the procedure *Reconstruct*, given in Fig. 2, and each process correctly computes the current coordinator value. Then, only the processes that chose to accept the coordinator value adopt the coordinator value, the remaining processes omit that value . The secret hidden here is, when the coordinator is correct, all processes adopt the same value v regardless of whether they accept or ignore the coordinator value.

3.2 Correctness

In the following, for a given phase l, let $N_c^v(l)$ and $N_{cu}^v(l)$ respectively be the number of correct processes and the number of cured processes that adopt v ($v \neq \perp$) at the end of the phase l.

Lemma 1 (Agreement) The non-faulty processes decide on the same value.

Proof We have to show that there exists a phase l in which all correct and cured processes adopt

```
Procedure Reconstruct()
35 :    Begin
36 :      for all i do
37 :        if ∃v ≠⊥ such that |{j|EV[j][i] = v}| ≥ n − 2t then
38 :          SV[i] ← v
39 :        else
40 :          SV[i] ← ⊥
41 :        end if
42 :      end for
43 :      if (∃v ≠⊥: #_v(SV) > 2t) then
44 :        val ← v
45 :        coord-accept ← False
46 :      else if (∃v ≠⊥: #_v(SV) > t) then
47 :        val ← v
48 :      else
49 :        val ← ⊥
50 :      end if
51 :    End
```

Figure 2: **Procedure Reconstruct**

the same value v, i.e., the condition $N_c^v(l) + N_{cu}^v(l) \geq n - t$ holds.

Remember that, we have assumed that at least one process, say k, remains uncorrupted for at least $3n$ rounds. Let l be the phase in which process k is the coordinator. Since, k is correct, it will send the same vector SV_k to all processes. There are two cases to consider at line 31 of algorithm *MBA*:

- (Case 1) *coord-accept* is $True$ for each process p (it means p chooses to adopt the coordinator value). Since all processes get the same vector SV_k from k, they compute the same value v at line 32 and adopt it. Hence, $N_c^v(l) + N_{cu}^v(l) \geq n - t$ holds. The lemma follows.

- (Case 2) *coord-accept* is $False$ for some process p (it means that p does not adopt the coordinator value while other processes adopt that value). Since *coord-accept* is $False$ at p, it follows that there exists some value v such that $\#_v(SV_p) > 2t$, and hence p adopts v. Note that, in voting round, if two correct processes vote for $v \notin \perp$ and $v' \notin \perp$ respectively, then $v = v'$. Also, since there are at most t Byzantine processes, and cured processes do not cast vote, it is guaranteed that at the coordinator process k, $\#_v(SV_k) > t$ and $\#_{v'}(SV_k) \leq t$ holds for any value v' ($v' \notin \{v, \perp\}$). Hence, at each process, the coordinator value is v. Consequently, all processes (correct and cured) have the same value v regardless of whether they accept or ignore the coordinator value. The lemma follows.

□

Lemma 2 (Consistency maintenance) Let $n \geq 4t + 1$. If at the end phase l, $N_c^v(l) + N_{cu}^v(l) \geq n - t$ holds for some value v, then $N_c^v(m) + N_{cu}^v(m) \geq n - t$ holds at all phases m ($m > l$).

Proof Assume that at the end of phase l, $N_c^v(l) + N_{cu}^v(l) \geq n - t$ holds for some value v. In the proposal round of phase $l + 1$, at most t new faults can occur. Hence, at least $n - 2t$ processes certainly send v. Note that, for each new fault there is a cured process, and cured processes do not send any messages (by definition). Hence, at any process p, the entries that correspond to cured processes in PV_p contain $\perp$. As a result, the condition $\#_v(PV_p) \geq n - 2t$ and $\#_v(PV_p) + \#_\perp(PV_p) \geq n - t$ holds. Also, since there are at most t faulty processes, $\#_{v'}(PV_p) \leq t$ for any value v' ($v' \notin \{v, \perp\}$). Hence, all correct and cured processes (at least $n-t$ processes) adopt v.

In the voting round, since at most t new faults can occur, at least $n - 2t \geq 2t + 1$ processes can vote for v. Hence, p can collect at least $n - 2t$ votes for v (i.e., $\#_v(SV_p) > 2t$). Since there are at most t Byzantine faults, p can get at most t votes for any value $v'(v' \notin \{v, \perp\})$. As a result, each correct and cured process (at least $n-t$) adopts v and choose not to adopt coordinator value in the next round.

Among these processes, at most t processes can become faulty in the coordinator round and the remaining (at least $n - 2t$) processes have v and do not adopt the coordinator value. Again, for each new fault there is a cured process. Since, all correct processes broadcast their vote vector SV, each cured process p correctly reconstructs the previous voting round using procedure *Reconstruct*, and finds $\#_v(SV_p) > 2t$. Therefore, it adopts v and and ignores the coordinator value. As a result, at the end of phase $l + 1$, $N_c^v(l+1) + N_{cu}^v(l+1) \geq n-t$ holds. The situation repeats itself, and the lemma holds. □

REFERENCES

Lemma 3 (Unanimity) If all non-faulty processes have the same initial value v, then no non-faulty process decides a value different from v.

Proof All non-faulty processes (at least n-t) have the same initial value v at the beginning of phase1. This statement can be interpreted as at least n-t processes have the same value v at the end of phase0. By using the same arguments as in lemma 2, we can show that at the end of phase1, $N_c^v(1) + N_{cu}^v(1) \geq n - t$ holds and this situation continues in all the following phases. Hence, the lemma holds. □

Lemma 4 (Termination) Each non-faulty process eventually irreversibly decides a value.

Proof From lemma 1, we can prove that all processes (correct and cured) certainly decide on the same value v ($v \in \mathcal{V}$) at the end of the correct coordinator phase l ($l \leq n$). Then, from lemma 2, we can show that the decision value will not change after phase l. Hence, the lemma holds. □

Theorem 1 Let $n > 4t$. The algorithm MBA described in Fig. 1 solves the mobile Byzantine agreement problem.

Proof The proof follows from lemmas 1, 2, 3, and 4. □

4. DISCUSSION

We conjecture that we can not improve the resilience more than $n > 4t$, since in a previous result [19], Thambidurai et al. have proved that, even in static failure models (strong models), $n > 4t$ is the necessary assumption to solve consensus with t benign failures and t Byzantine failures. Note that, benign failures are less severe than Byzantine failures. An example for benign failures is omission failure. When a process fails by omission, it forgets to send or receive messages. In our model, we may consider the cured processes as omission failures since they do not send any messages. Hence, we can say that our model also suffers from t benign and t Byzantine failures. Moreover, our model is a weak model the sense that the set of faulty processes dynamically changes in every round.

5. CONCLUSION

In this paper, we studied the Byzantine agreement problem with mobile faults. We have presented an algorithm *MBA* for synchronous systems where there are t malicious agents that move at full speed to corrupt and destroy the memory of processes. Our algorithm improves the result of Garay[10], namely it requires $n > 4t$ in stead of $n > 6t$[10].

Table 1: **Performance comparison of Algorithm MBA with previous works.**

	Garay's model	**Weaker model (message moving agent)**
Garay et al.[10]	$n > 6t$	–
Burhman et al.[3]	–	$n > 3t$
Algorithm MBA	$n > 4t$	–

Table 1 compares our algorithm MBA with previous ones. One drawback of our algorithm is, it requires $3n$ communication rounds while the algorithm of Garay[10] requires $2n$ rounds.

REFERENCES

[1] M. Biely and M. Hulte: "Consensus When All Processes may be Byzantine for Some Time", *In Proc. 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS '09)*, vol.5873, pp.120–132, 2009.

[2] A.N Bessani and P. Sousa and M. Correia and N.F. Neves and P. Verissimo: "The CRUTIAL way of critical infrastructure protection", *IEEE Security and Privacy*, vol. 6(6), pp. 44-51, 2008.

[3] H. Buhrman, J. A. Garay and J.Hoepman: "Optimal Resiliency Against Mobile Faults", *In Proc. 25th International Symposium on Fault-Tolerant Computing (FTCS'95)*, 1995.

[4] C. Cachin and K. Kursawe and F. Petzold and V. Shoup: "Secure and efficient asynchronous broadcast protocols (extended abstract)", *Kilian, J., editor, Advances in Cryptology: CRYPTO 2001*, vol.2139 of LNCS, pp. 524-541, 2001.

[5] M. Castro and B. Liskov: "Practical Byzantine fault tolerance and proactive recovery", *ACM Transactions on Computer Systems*, vol.20(4), pp. 398-461, 2002.CMS07,

[6] T. Chandra and S. Toueg: "Unreliable failure detectors for reliable distributed systems", *Journal of the ACM*, vol.43(2), pp.225-267, 1996.

[7] B.G. Chun and P. Maniatis and S. Shenker and J. Kubiatowicz: "Attested append-only memory: making adversaries stick to their word", *In Proc. of the 21st ACM Symposium on Operating Systems Principles*, pp. 189-204, 2007.

REFERENCES

[8] M. Correia and N.F. Neves and P. Verissimo: "From consensus to atomic broadcast: Time-free Byzantine-resistant protocols without signatures", *"Computer Journal*, vol. 41(1), pp. 82-96, 2006.

[9] V. Hadzilacos and S. Toueg: "A modular approach to fault-tolerant broadcasts and related problems", *Technical Report TR94-1425*, Cornell University, Department of Computer Science, 1994.

[10] J.A. Garay: "Reaching (and Maintaining) Agreement in the Presence of Mobile Faults", *In Proc. 8th International Workshop on Distributed Algorithms*, LNCS. No.857, pp.253–264, 1994.

[11] R. Guerraoui and A. Schiper: "The generic consensus service", *IEEE Transactions on Software Engineering*, vol. 27(1),pp. 29-41, 2001.

[12] Kepphart and White: "Directed graph epidemiological models of computer viruses", *IEEE symposium on Security and Privacy*, 1991.

[13] L. Lamport and R. E. Shostak and M. C. Pease: "The Byzantine Generals Problem", *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol.4(3), pp. 382–401, 1982.

[14] R. Ostrovsky and M. Yung: "How to withstand mobile attacks" *In Proc. of tenth annual ACM symposium on Principles of distributed computing(PODC'91)*, 1991.

[15] R. Reischuk: "A new solution for Byzantine generals problem", *Information and Control*, vol. 64, 1985.

[16] N.Santoro and P.Widmayer: "Time is not a healer", *In Proc. 6th Annual Symposium on Theor. Aspects of Computer Science(STACS89)*, LNCS.No.349, pp.304-313, 1989.

[17] U. Schmid, B. Weiss and I. Keidar: "Impossibility Results and Lower Bounds for Consensus under Link Failures", *SIAM Journal on Computing*", vol.38, pp. 1912–1951, 2009.

[18] F.B. Schneider: "Implementing faul-tolerant services using the state machine approach: A tutorial", *ACM Computing Surveys*, vol.22(4), pp. 299-319, 1990.

[19] P. Thambidurai and You-Keun Park: "Interactive Consistency with Multiple Failure Modes", *In Proc. Reliable Distributed Systems*, pp.93–100, 1988.

[20] G.S. Veronese and M. Correia and A.N. Bessani and L.C. Lung: "Highly-resilient services for critical infrastructures", *In Proc. of the Embedded Systems and Communications Security Workshop*, 2009.

[21] J. Yin and J. Martin and A. Venkataramani and L. Alvisi and M. Dahlin: "Separating agreement from execution for Byzantine fault-tolerant services", *In Proc. of the 19th ACM Symposium on Operating Systems Principles*, pp. 253-267, 2003.