Leveraging Epidemic Quorum Protocols with Byzantine Fault Tolerance

André Nunes
Under supervision of João Pedro Barreto
Dep. Engenharia Informática, IST, Lisbon, Portugal

May 6, 2012

Abstract

Epidemic quorum protocols are known to achieve highly available agreement even when a quorum is not simultaneously connected. Therefore, they are a very interesting option for mobile and other weakly connected networks. However, every epidemic quorum system proposed so far neglects byzantine failures.

We propose a novel replication prutocol, called eBFT. To the best of our knowledge, eBFT is the first epidemic quorum protocol to tolerate byzantine replica failures. We simulate eBFT and quantify the overhead inflicted by adding byzantine fault tolerance to epidemic quorum protocols in eBFT. We show that, at the cost of doubling the number of exchanged messages, eBFT can deliver byzantine fault tolerance to epidemic quorum protocols.

Keywords: Replication, Byzantine, Quorum

1 Introduction

Many distributed systems rely on quorum systems for reliable agreement in scenarios where part of the system can be unavailable [1]. Quorum systems can be applied to a wide range of systems that require coordination, ranging from data replication protocols, distributed mutual exclusion, name servers, selective dissemination of data, to distributed access control and signatures [2].

In order to have a distributed system agree on a proposed candidate value, classical quorum protocols require that a quorum of live processes converges to that decision. The quorum of live processes is typically assumed to be simultaneously connected in the same network partition, which is not adequate in weakly connected networks, e.g. mobile ad hoc or sensor networks, where connected quorums are often the norm.

Epidemic quorum protocols (EQPs) were proposed to allow unconnected quorums to reach agreement on a value to be accepted [3]. This is done by running a finite number of elections, where each replica may vote for one proposed value. By epidemic propagation of votes, eventually each replica should be able to determine, from its local information, whether the system has agreed on a given value, or the current election has reached an inconclusive state and a new election needs to be done. EQPs, hence, are a strong tool for coordination in weakly connected networks.

Recent studies [4] show that the majority of IT software errors detected in commercial databases can lead them to byzantine behavior [5]. Furthermore, the same can happen due to malicious attacks [6]. Still, to the best of our knowledge, no EQP proposed so far is able to tolerate byzantine failures. In other words, should the replicas belonging to quorum protocols suffer malicious attacks or software errors, the EQP can take incorrect decisions.

Hence, we advocate that, in the same way that classical quorum systems evolved to more intricate variations that tolerate byzantine failures [5], the same direction ought to be followed regarding EQPs.

The main contribution of this paper is a novel replication protocol, called eBFT, which is the first EQP to support byzantine failures. eBFT allows agreement on a value to be accepted in unconnected quorums despite the existence of a limited number of replicas with byzantine behavior. Like any EQP,

eBFT uses elections and epidemic propagation of votes. However, eBFT tolerates byzantine replicas by imposing larger quorums than other EQPs and by digitally signing votes.

We quantify the cost inflicted by adding byzantine fault tolerance to EQPs with simulation results that compare eBFT with Keleher et al.'s non-byzantine fault-tolerant EQP [3]. We show that, at the cost of doubling the number of exchanged messages, eBFT can deliver byzantine fault tolerance to EQPs.

The remaining of this paper is organized as follows. Section 2 addresses related work on quorum protocols. Section 3 presents the system model assumed hereafter. Section 4 introduces eBFT. Section 5 presents experimental results. Finally, Section 6 draws some concluding remarks.

2 Related Work

Some state machine replication protocols [7] with byzantine fault tolerance have been proposed [8–10]. These protocols allow to model a service as a state machine and replicate it at a number of replicas, which contain a maximum limited number of byzantine replicas. Most of these protocols ensure the properties of linearity [11] and termination of client requests. The total minimum number of replicas to ensure these properties in the presence of f byzantine replicas is $3f + 1$.

Moniz et al. [12] proposed a protocol that tolerates a limited number of mobile byzantine replicas and uses a failure model which is based on the Santoro and Widmayer's communication failure model [13]. This model assumes the existence of dynamic and transitory failures in the communication channels. One characteristic of wireless networks is that the cost of transmitting a message to multiple replicas is the same as sending it to a single replica, since these replicas are within communication range.

The above mentioned protocols don't use epidemic propagation of information as the epidemic quorum protocols [3, 14] use, which we consider the most appropriate communication method for weakly connected wireless networks.

Keleher et al. [3] proposed a protocol that ensures consistency of the state of a replicated logic object in the presence of concurrent operations. The information of each replica is propagated among all replicas epidemically. This characteristic withdraws the requirement of the existence of simultaneously connected quorum, which is ideal to weakly connected networks.

To decide which operation is committed in the logical object, Keleher et al. uses elections. The candidates of each election are the replicas that purpose an operation. All replicas can vote in one and only one candidate. The candidate that wins the election has his proposed operation committed and the others candidates have their proposed operations aborted.

The logical object has a fixed total vote weight of 1 that is distributed among all replicas. The percentage of vote weight of each replica constitutes its vote power. The vote weight of each replica is variable, which allows the existence of replicas with more influence in the decision of the election's winner.

The voting is made in a decentralized manner. Through epidemic propagation of each replica vote information, a replica eventually will decide in a candidate to win a election based on its local information. A candidate wins the election when he has the plurality of votes. The information exchanged between the replicas are: (i) the result of finished elections; (ii) the known votes of the last or unfinished election.

Holliday et al. [14] proposed an EQP that address the multi-object case and can commit multiple compatible transactions in a single election. However, their protocol does not ensure one-copy serializability.

However the existent epidemic quorum protocols do not tolerate byzantine failures because if we assume the existence of a Byzantine replica, this replica could convey the wrong information to different replicas and provoke those replicas to take wrong decisions

3 System Model

For simplicity and without loss of generality we assume that the system replicates a single logical object. This system can be extended to a model where multiple objects are replicated in some group of replicas.

The total number of replicas includes a maximum limited number of byzantine replicas, f. The minimum total number of replicas, N, is equal to 3f+1 replicas. At the time that eBFT starts every replica in the system knows the value of N. However that value can decrease with the detection of byzantine replicas.

The pseudocode of eBFT protocol is at Algorithm 1 and Algorithm 2. Each replica maintains a state that includes:

- votes votes[N]: each position votes[n] contains the known vote of replica i, that is, a request op digitally signed by replica i, votes[i]= $[op]_i$.
- proof byzantines[]: each position byzantines[i] contains the proof of a byzantine replica i behavior, that is, a pair of votes at two different candidates (false vote), byzantine[i]={$[op]_n$, $[op']_n$}.
- N: total number of replicas;
- request: request received from client digitally signed, $[op]_c$;

Also, in our model we assume asynchronous communication and that every message exchanged in the system is digitally signed by message's sender. In our notation $[m]_x$ means that message m is digitally signed by sender x.

4 eBFT

We now introduce Epidemic Byzantine Fault Tolerance (eBFT), a replication algorithm tolerant to byzantine failures in replicas. If competing requests occur, eBFT tries to eventually decide a single request to eventually commit at all replicas, while the remaining concurrent requests eventually abort at all replicas. The algorithm propagates votes epidemically and uses elections to decide in which order the requests should be committed.

For simplicity, we describe the algorithm assuming a single election. It is straightforward to extend the described algorithm to multiple sequential elections, following Keleher et al.'s approach [3].

Each proposed request by a client is a candidate in the election and each correct replica votes only in one candidate. Considering the total weight of all votes equal to 1, the voting weight of each replica is equal to 1/N, where N is the total number of replicas. Like in other EQPs, each replica of eBFT can only vote for one candidate and this vote cannot be revoked. By epidemic propagation of voting information, each replica learns about the remaining replicas' votes. Each replica with the information of votes that it has locally, verifies if there exists some candidate with enough votes to be elected. If so, the replica decides in that candidate and commits the request correspondent to that candidate.

To win an election a candidate needs to obtain a plurality of votes. However, due to the existence of f byzantine replicas, f of the votes may be *false*. A false vote corresponds to when a byzantine replica votes for more than one candidate. For example consider that we have a byzantine replica B, two other correct replicas X and Y, and two candidates C_1 and C_2. A false vote happens if B tells X that B voted for C_1, and then B tells Y that B voted for C_2.

The key insight of eBFT is that, in order to tolerate f possible false votes, a candidate to be elected in a correct replica needs: a number of votes greater than the votes in any other candidate, plus the maximum number of false votes or byzantine replicas f, plus the number of replicas that have not voted. More precisely, a correct replica A elects a candidate C_1 if the following condition is true :

$$\forall_{C_i \neq C_1} \text{ votes}_A(C_1) > \text{votes}_A(C_i) + f + \text{unknown}_A$$

- $\text{votes}_A(C_i)$: number of votes known by A in candidate i;
- unknown_A: number of replicas, which their vote is unknown by A;

4.1 Voting

A correct replica only votes in a candidate when it didn't vote yet in other candidate and one of the follow conditions is fulfilled: (i) the received candidate is digitally signed by client(Algorithm 1, lines 11-18); (ii) the candidate was voted by a replica and that replica have the candidate digitally signed by the client(Algorithm 2, lines 29-30).

The candidate digitally signed by the client is necessary to prevent a correct replica from copying a candidate created by a byzantine replica. To ensure that a proposed candidate by a client reach at least one correct replica, the client sends its candidate to f+1 replicas that haven't vote in any candidate yet(Algorithm 1, lines 2-8). Therefore it is guaranteed the possibility of somewhere in time other correct replicas vote in the client's candidate.

4.2 Epidemic Propagation of information

When a replica finds another replica, they exchange information. That information consists in digitally signed votes and byzantine replicas' identification that each replica knows. Also, if the replica has voted in one candidate it sends that candidate digitally signed by the client (Algorithm 1, lines 21-23).

When a replica receives the information it verifies if there are: (i) unknown byzantine replicas (Algorithm 2, lines 3-7); (ii) unknown votes (Algorithm 2, lines 26-27); (iii) false votes. If some false vote is detected by a replica, the replica adds the responsible for that false vote to the known byzantine replicas(Algorithm 2, lines 13-18).

4.3 Detection of Byzantine Replicas

Each replica's vote is digitally signed with replica's private key and each replica knows all system replicas' public keys. Each vote's digital signature guarantees authenticity to the vote.

With this mechanism, the byzantine replicas' malicious actions are limited. A byzantine replica can't modify other correct replicas' vote or vote for others correct replicas. However, a byzantine replica can always vote for two or more candidates received from clients. With the votes' authenticity guarantee, this algorithm has the possibility to detect byzantine replicas that vote in different requests.

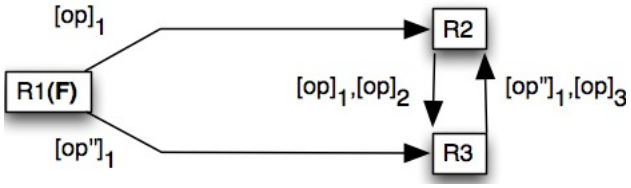

Figure 1: detection of byzantine replica

In the example of the Figure 4 we consider the communication between three replicas (R1, R2, R3) with R1 as byzantine replica and, R2 and R3 as correct replicas. There are two candidates op and op". R1 votes in two different candidates and sends one vote to R2 and the other different vote to R3. When R2 and R3 exchange voting information between them, they verify that R1 is byzantine replica. When any replica detects a byzantine replica it decreases in one value N and thus incrementing the vote power of each replica (Algorithm 2, lines 13-18).

4.4 Byzantine Clients

All messages, or requests, sent by the clients are authenticated with their private keys. The only possible action for the clients is to propose requests (candidates). This way, a byzantine client will never provoke different correct replicas to elect different candidates.

A possible attack from a byzantine client is the denial of service. A byzantine client can propose an elevated number of requests (candidates) and consequently decrease the probability of any candidate to be elected.

4.5 Minimum Total Number of Replicas

Assuming a maximum total number of f byzantine replicas, the minimum total number of replicas in the system, N, has great influence in the possibility of a candidate being elected. It's easy to show that if N isn't big enough it will be impossible to elect a candidate in any correct replica.

We considered that with N, it should be possible to elect a candidate when all the correct replicas vote in a single candidate and all byzantine replicas vote in another candidate. We found that the N value for this situation to be possible is 3f+1.

Algorithm 1 eBFT pseudocode, part 1

```
1: Executed when Client c proposes a request op
2: function propose(request op)
3: while n ≤ f + 1 do
4:    Choose accessible replica B
5:    if clientSend(B, [op]_c) then
6:       n++;
7:    end if
8: end while
9: ——————————————————————————————
10: Executed by replica B
11: function clientSend(replica B, authenticated_request [op]_c)
12: if B.votes[B] != ⊥ then
13:    return false;
14: else
15:    B.votes[B] = [op]_B;
16:    request = [op]_c
17:    return true;
18: end if
19: ——————————————————————————————
20: Executed by all replicas to exchange information with other replica
21: function sendInfo()
22: Replica S chooses an accessible replica B;
23: replicaSend(B, S.votes[], S.byzantines[], S.request);
```

In any possible case, there is no chance of two or more different candidates being elected. The only possibilities are: (i) no candidate elected in any correct replicas; (ii) some correct replicas elect the same candidate; (iii) all correct replicas elect the same candidate.

5 Evaluation

The primary goal of eBFT is to improve the ability of the system to make progress during times of low connectivity in the presence of a limited number of byzantine replicas. Thus it is essential for the system to have a good commit speed.

The main goal of this section is to evaluate the overhead of eBFT relatively to a non byzantine fault tolerant EQP. For that reason our simulator runs both eBFT and Keleher et al. [3]. The main difference between Keleher et al.'s and eBFT's implementations in our simulator is that Keleher et al. doesn't have byzantine replicas, f = 0. The consequences in Keleher et al.'s implementation are: (i) a request is proposed to only one replica chosen randomly; (ii) if a replica A didn't vote and it receives information of another replica B, which already voted in a request, then A copies the B's vote; (iii) a replica A decides in a candidate C_1 if C_1 has the relative majority of votes; $\forall_{C_i \neq C_1}$ $votes_A(C_1) > votes_A(C_i) + unknown_A$

We built a simulator using the same model as Keleher et al. [3]. In both protocols simulations time is broken into uniform intervals and a thread represents each replica. During each interval each replica randomly chooses other replica and sends its information. When all replicas communicate in an interval, they move to the next interval. The simulation for each set of parameters ends when the maximum number of rounds is reached. In all simulations when the maximum number of rounds is reached all replicas have already decided on the proposed request. All results were obtained by an average of 10 times with the same set of parameters.

Algorithm 2 eBFT pseudocode, part 2

```
1: Replica B receives votes and byzantine replicas known by replica S
2: function replicaSend(replica B, list votes[], list byzantines[], authenticated_request [op]_c)
3: for all byzantines[i] in byzantines[] do
4:    if !B.byzantines[].contains(byzantines[i]) ⋀ proofValidation(byzantines[i] then
5:       ignoreReplica(i);
6:       N - -;
7:       B.byzantines[i] = byzantines[i];
8:    end if
9: end for
10: for all votes[i] in votes[] do
11:    known_vote = false;
12:    for all B.votes[w] in B.votes[] do
13:       if i==w ⋀ ([op]_i!=[op']_w) then
14:          known_vote = true;
15:          B.byzantines[i] = {[op]_i, [op']_w}
16:          ignoreReplica(i);
17:          N - -;
18:          break;
19:       end if
20:       if i==w ⋀([op]_i==[op]_w) then
21:          known_vote = true;
22:          break;
23:       end if
24:    end for
25: end for
26: if !known_vote then
27:    B.votes[n] = votes[n];
28: end if
29: if B.votes[B]==⊥ ⋀ requestValidation(B.votes[S], [op]_c) then
30:    B.votes[B]= [op]_B
31: end if
32: request r = plurality(B.votes[]);
33: if if(r != ⊥) then
34:    commit(r);
35: end if
```

5.1 Results

In this section we show all the relevant results obtained with the simulations of eBFT protocol and Keleher et al.. All figures of this section plot both protocols in order to compare the results.

The first metric of interest is number of intervals needed for the first replica to commit a request versus the total number of replicas. This metric is important for applications where write requests are more relevant than read requests. In our protocol when the first replica commits a request then eventually somewhere in time all replicas will do the same. Thus for the above mentioned applications type, the most important is the time until the first replica commits.

Figure 5 plots the average number of intervals needed for the first replica to commit versus number of replicas.

The other metric of interest is number of intervals needed for all replicas to commit a request versus the total number of replicas. Figure 6 plots this metric.

Figure 7 plots the number of intervals needed by the n-th replica to commit with N=10. Figure 8 plots the same with N=100.

We simulated the existence of partitions in the network assigning to each replica a random partition in each interval. In this experiment only replicas in the same partition can communicate. Figure 9 plots the number of rounds needed for all replicas to commit a request versus the total number of replicas with the existence of two partitions. Figure 10 plots the same with the existence of twenty partitions.

We also simulated the existence of concurrent requests, where zero or one of the concurrent requests is committed. Figure 10 plots the percentage of commit versus the number of concurrent requests, with N = 10. The percentage of commit is the relation between the total number of experiments where a request was committed and the total number of experiments.

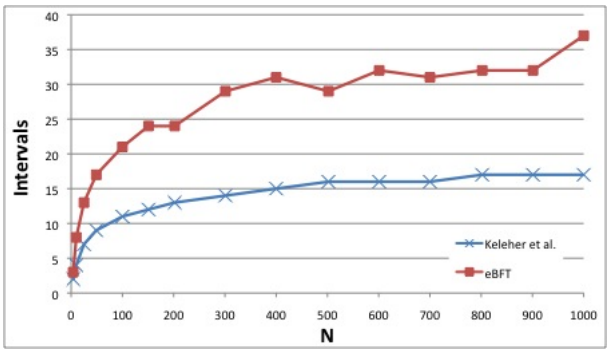

Figure 2: number of intervals for first replica to commit versus total number of replicas

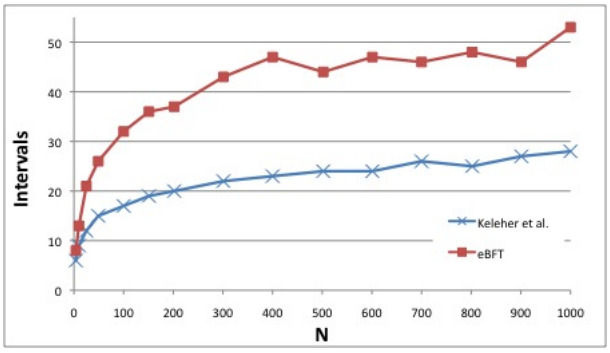

Figure 3: number of intervals for all replicas to commit versus total number of replicas

After we analyzed the results we verified that eBFT needs about twice the same time intervals as Keleher et al. for all replicas to commit a request or for the first replica to commit. There are two reasons for this. The first reason is that in eBFT a replica needs a quorum with more f votes to decide in a request than a replica in Keleher et al. needs. The second reason is that in eBFT the f byzantine replicas can just simply refuse to communicate with other replicas, which in the case of Keleher don't happen because all replicas are correct.

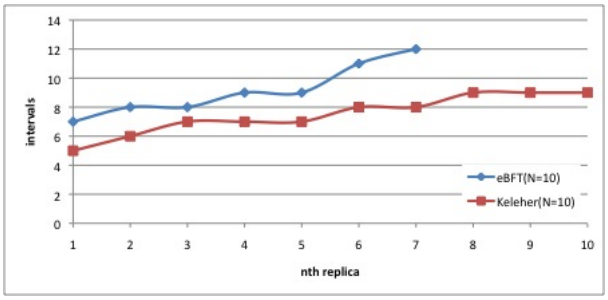

Figure 4: number of intervals for n-th replica to commit with N = 10

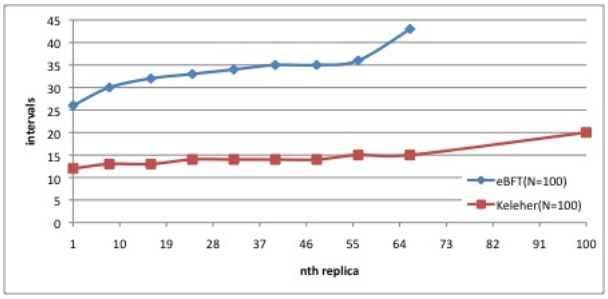

Figure 5: number of intervals for n-th replica to commit with N = 100

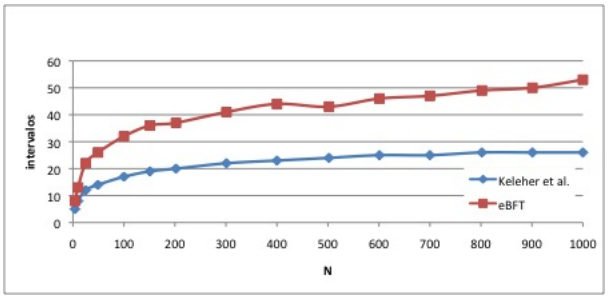

Figure 6: number of intervals for all replicas to commit with two partitions versus total number of replicas

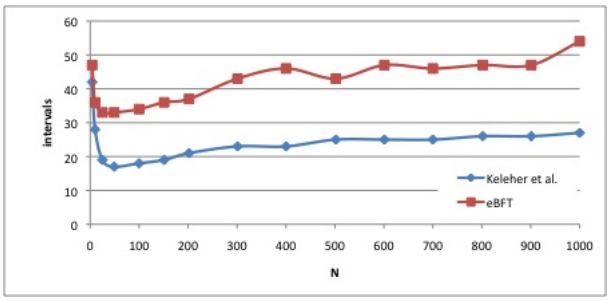

Figure 7: number of intervals for all replicas to commit with twenty partitions versus total number of replicas

Figure 10 shows that eBFT in the presence of concurrent requests is almost certain that all requests are aborted. The reason for this behavior is that when a client proposes a request, he tries to get f+1 different replicas to accept his request in order to have the guarantee that at least one correct replica accepts the request. In the case of Keleher et al. the client just needs to contact a single accessible replica.

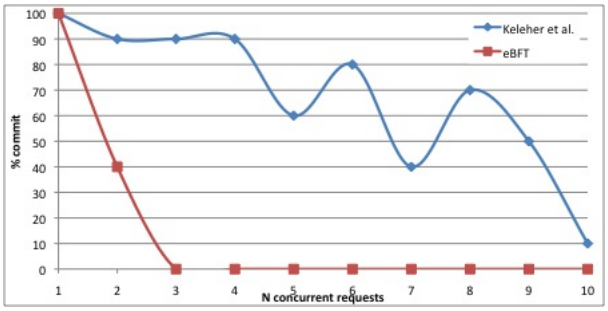

Figure 8: percentage of commit versus total number of concurrent requests with N =10

6 Conclusion

Epidemic quorum protocols are known to achieve highly available agreement even when a quorum is not simultaneously connected. Therefore, they are a very interesting option for mobile and other weakly connected networks.

In contrast, epidemic quorum protocols are based on epidemic vote propagation model that is known be much more appropriate to the availability challenges that weakly connected environments impose. However, to the best of our knowledge, no epidemic quorum protocol proposed so far is able to tolerate byzantine failures. Still, software bugs and malicious attacks are real threats to many applications and systems running in weakly connected environments.

We propose a novel replication protocol, called eBFT, that is the first epidemic quorum protocol to tolerate byzantine replica failures. We simulate eBFT and quantify the overhead inflicted by adding byzantine fault tolerance to epidemic quorum protocols in eBFT. We show that, at the cost of doubling the number of exchanged messages, eBFT can deliver byzantine fault tolerance to epidemic quorum protocols.

We believe that eBFT is a first step into the valuable direction of byzantine fault-tolerant epidemic quorum protocols. This work defines a first upper-bound on the cost of byzantine fault-tolerance in this kind of protocols.

References

[1] D. Peleg and A. Wool, "The availability of quorum systems," *Information and Computation*, vol. 123, no. 2, pp. 210–223, 1995. [Online]. Available: citeseer.ist.psu.edu/peleg93availability.html

[2] Y. Amir and A. Wool, "Evaluating quorum systems over the internet," in *Symposium on Fault-Tolerant Computing*, 1996, pp. 26–35. [Online]. Available: citeseer.ist.psu.edu/amir96evaluating.html

[3] P. J. Keleher, "Decentralized replicated-object protocols," in *Proceedings of the eighteenth annual ACM symposium on Principles of distributed computing*, ser. PODC '99. New York, NY, USA: ACM, 1999, pp. 143–151. [Online]. Available: http://doi.acm.org/10.1145/301308.301345

[4] I. Gashi, P. Popov, and L. Strigini, "Fault tolerance via diversity for off-the-shelf products: A study with sql database servers," vol. 4. Los Alamitos, CA, USA: IEEE Computer Society Press, October 2007, pp. 280–294. [Online]. Available: http://portal.acm.org/citation.cfm?id=1320305.1320887

[5] D. Malkhi and M. K. Reiter, "Survivable consensus objects," in *Proceedings of the The 17th IEEE Symposium on Reliable Distributed Systems*, ser. SRDS '98. Washington, DC, USA: IEEE Computer Society, 1998, pp. 271–. [Online]. Available: http://portal.acm.org/citation.cfm?id=829523.830975

[6] D. I. S. A. DISA, *Database security technical implementation guide - version 7, release 1.* White paper available at databasesecurity.com, October 2004.

[7] F. B. Schneider, "Implementing fault-tolerant services using the state machine approach: a tutorial," vol. 22. New York, NY, USA: ACM, December 1990, pp. 299–319. [Online]. Available: http://doi.acm.org/10.1145/98163.98167

[8] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Proceedings of the third symposium on Operating systems design and implementation*, ser. OSDI '99. Berkeley, CA, USA: USENIX Association, 1999, pp. 173–186. [Online]. Available: http://portal.acm.org/citation.cfm?id=296806.296824

[9] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong, "Zyzzyva: speculative byzantine fault tolerance," in *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, ser. SOSP '07. New York, NY, USA: ACM, 2007, pp. 45–58. [Online]. Available: http://doi.acm.org/10.1145/1294261.1294267

[10] J.-P. Martin and L. Alvisi, "Fastbyzantine paxos," University of Texas at Austin, Feb 2004.

[11] M. P. Herlihy and J. M. Wing, "Axioms for concurrent objects," in *Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, ser. POPL '87. New York, NY, USA: ACM, 1987, pp. 13–26. [Online]. Available: http://doi.acm.org/10.1145/41625.41627

[12] H. Moniz, N. Ferreira Neves, and M. Correia, "Turquois: Byzantine consensus in wireless ad hoc networks."

[13] N. Santoro and P. Widmayer, "Time is not a healer," in *Proceedings of the 6th Annual Symposium on Theoretical Aspects of Computer Science*. London, UK: Springer-Verlag, 1989, pp. 304–313. [Online]. Available: http://portal.acm.org/citation.cfm?id=646505.694026

[14] J. Holliday, R. Steinke, D. Agrawal, and A. El Abbadi, "Epidemic algorithms for replicated databases," vol. 15. Piscataway, NJ, USA: IEEE Educational Activities Department, September 2003, pp. 1218–1238. [Online]. Available: http://dx.doi.org/10.1109/TKDE.2003.1232274