From Crash to Byzantine Consensus with $2f + 1$ Processes *

Giuliana Santos Veronese[1], Miguel Correia[1], Lau Cheuk Lung[2]
[1]*Universidade de Lisboa, Faculdade de Ciências, LASIGE*
[2]*Dep. de Informática e Estatística, Centro Tecnológico, Universidade Federal de Santa Catarina*
giuliana@lasige.di.fc.ul.pt, mpc@di.fc.ul.pt, lau.lung@inf.ufsc.br

Introduction *Consensus* is an important distributed computing problem that consists in making a set of processes agree on one of the values that each one of them proposes. Consensus in the asynchronous Byzantine message-passing model (the model we consider in the abstract) has been shown to require $n \geq 3f + 1$ processes to be solvable in several variations of the basic system model, where f is the maximum number of faulty processes. Reducing the ratio n/f is important both theoretically, since achieving lower bounds has been always a goal in distributed computing, and in practice, as reducing the number of processes reduces the cost of a real system. Recently a few solutions to implement Byzantine fault-tolerant state machine replication, which involves solving consensus, with only $2f + 1$ replicas have appeared [3, 2]. This reduction from $3f + 1$ to $2f + 1$ is possible by extending the system model with a trusted/trustworthy component that constrains the power of faulty processes to have certain behaviors. These components have been called *wormholes* [6].

State machine replication consists in replicating a service in a set of n servers, f of which may be faulty. Correia et al. use a wormhole called TTCB to help define an order for the execution of the clients' requests with only $2f + 1$ servers [3]. More recently, Chun et al. used an attested append-only memory (A2M) abstraction with the same purpose [2].

The main objective of this work is to contribute to a better understanding of the problem of consensus with only $2f + 1$ processes. Despite these important results, the problem of solving asynchronous Byzantine consensus with only $2f + 1$ processes is still far from being well understood. There are several reasons for this state of affairs: the related works we cited are only two and very recent; they solve consensus but have the solution for this problem submerged in the complications of a larger problem (state machine replication); they are based on special-purpose, reasonably complex, abstractions/wormholes (TTCB, A2M) that researchers other than the authors of those papers are probably not familiarized with.

*This work was supported by the EC through Alban scholarship E05D057126BR, by the FCT through the Multiannual and the CMU-Portugal Programmes, and by CAPES/FCT through project TISD.

$2f + 1$ **Reliable Broadcast with a Wormhole** The reliable broadcast problem consists basically in making all correct processes *deliver* the same messages [1]. Furthermore, if the process that *broadcasts* the message is correct, then all correct processes deliver the message, and no two messages with the same identifier are delivered by any correct process. Bracha presented a reliable broadcast algorithm that needs $n \geq 3f + 1$ processes [1]. He does not provide a proof that $3f + 1$ is the minimum number of processes but it is simple to understand that the algorithm can not work with $n = 2f + 1$.

With a wormhole it becomes possible to do reliable broadcast with $n = 2f + 1$. Consider that there is a set of trusted/trustworthy wormholes $\{w_1, ...w_n\}$ and that process p_j has access exclusively to wormhole w_j. Each wormhole w_j has a public-private key pair. The private key K_{rj} is known only by w_j and is used to produce digital signatures. Every correct process knows the correct public key K_{uj} of every wormhole w_j, so it can verify signatures produced by the wormholes using their private keys. The wormholes provide a single service that can be abstracted as a function that is called by the processes: $\sigma \leftarrow sign_j(id, msg)$ (for wormhole w_j). The function takes as parameters a message identifier id and a message msg. It returns either the signature σ of (id, msg) or $\perp$. The signature is returned if $id > id'$, where id' is the identifier given as parameter in the

Algorithm 1 Reliable broadcast algorithm

```
Function RELIABLE_BROADCAST($id, msg$)
Task T1:
1: $\sigma \leftarrow sign_i(id, msg)$
2: $\forall j \neq i$ : SEND INITIAL$(i, id, msg)_\sigma$ to $p_j$
Task T2: {exec only once per msg broadcasted}
3: when (message INITIAL$(j, id, msg)_\sigma$ or ECHO$(j, id, msg, \sigma)$ is received)
   and (verify$(id, msg, \sigma, K_{uj})$) do
4:     $\forall k \neq j$ : SEND ECHO$(j, id, msg, \sigma)$ to $p_k$
5:     RETURN$(id, msg)$
6: end when
```

previous call to the function; otherwise $\perp$ is returned. This service is very simple but it precludes a faulty process from obtaining two different messages with the same identifier correctly signed.

Algorithm 1 uses this service to solve reliable broadcast with any number of faulty processes. It is a variation of Bracha's but it has one less communication step due to the use of the wormhole and requires no bounds on the number of faulty processes.

Solvability of $2f+1$ Consensus We do not present a consensus algorithm designed from scratch, but modify Mostefaoui and Raynal's crash fault-tolerant consensus algorithm (*MR-Consensus* for short) [5]. Algorithm 2 is the modified algorithm, *2FBC*.

Some of the modifications to MR-Consensus are obvious and do not need much discussion: reliable channels are substituted by *authenticated reliable channels* and message disseminations are substituted by the *reliable broadcast primitive* (lines 5, 8). The identifier of a message is composed by the process identifier, the message type and the round number. Another modification is that we use Bracha's *message validation* mechanism to prevent some of the attacks that might be done by faulty processes [1]. In several places the algorithm only takes into account messages that are valid (lines 7, 9, 10, 14). Informally, a message is said to be *valid* if it is justified by the messages previously received by the process.

Another difference of 2FBC in relation to MR-Consensus is line 9. In MR-Consensus, processes *wait until* they receive messages from $n-f$ processes, but there is a crucial difference between the crash and the Byzantine fault models: while in the crash fault model (MR-Consensus) all messages in that $n-f$ quorum are sent by processes that follow the algorithm, in the Byzantine fault model (2FBC) f of those messages can be sent by faulty processes. In fact, in the worst case, with $n = 2f+1$ and f Byzantine processes, in every round that quorum of $n-f$ messages contains $f+1$ messages, f of which sent by Byzantine processes. To deal with this problem, line 9 must "know about" *all* processes before continuing, i.e., we need an *eventually perfect muteness FD* – $\Diamond\mathcal{MP}_\mathcal{A}$ [4]. Line 9 waits for messages from $n-f$ processes, but also either for messages or to suspect of the rest of the processes. This ensures that eventually p_i receives messages from all correct processes, as there is a time after which correct processes are not suspected by any correct process (eventual strong $\mathcal{A}$-accuracy). This is also the reason why we need a stronger FD than previous Byzantine consensus algorithms, that require only eventual *weak* $\mathcal{A}$-accuracy. While those algorithms require only that the coordinator is eventually not suspected, 2FBC requires that eventually no correct process is suspected, i.e., eventual *strong* $\mathcal{A}$-accuracy.

Final Remarks This work shows that it is possible to implement a $2f+1$ asynchronous Byzantine consensus algorithm using simple wormholes and an eventually perfect muteness failure detector. This is an interesting result due to practical importance of reducing the number of processes in real fault-tolerant systems.

Algorithm 2 2FBC Byzantine consensus algorithm

Function 2FBC_CONSENSUS(v_i)

Task T1:

```
1: $r_i \leftarrow 0$   {$r_i$ is $p_i$'s round number}
2: $est_i \leftarrow v_i$   {$est_i$ is $p_i$'s current estimate of the value to be decided}
3: while true do
4:    $c_i \leftarrow (r_i \; mod \; n) + 1$;   $r_i \leftarrow r_i + 1$   {$c_i =$ current coordinator}
      {———————- phase 1: coordinator to all ———————- }
5:    if $(c_i = i)$ then RELIABLE_BROADCAST PHASE1$(r_i, est_i)$ end if
6:    wait until (message PHASE1$(r_i, -)$ is received from $p_{c_i}$ or $p_{c_i}$ is suspected by $p_i$'s FD module)
7:    if (valid message PHASE1$(r_i, v)$ received from $p_{c_i}$) then $aux_i \leftarrow v$ else $aux_i \leftarrow \perp$ end if
      {——————————- phase 2: all to all ——————————- }
8:    RELIABLE_BROADCAST PHASE2$(r_i, aux_i)$
9:    wait until (valid messages PHASE2$(r_i, -)$ are received from at least $n - f$ processes)
              and ($\forall j$ : valid message PHASE2$(r_i, -)$ is received from $p_j$ or $p_j$ is suspected by $p_i$'s FD module)
10:   $\forall j$ : if (valid message PHASE2$(r_i, v)$ received) then $R_i[j] \leftarrow v$ else $R_i[j] \leftarrow \perp$ end if
11:   if $(\exists v : \#_v(R_i) \geq n - f)$ then $est_i \leftarrow v$; $\forall j \neq i$ : SEND DECISION$(r_i, est_i)$ to $p_j$; RETURN$(est_i)$ else
12:       if $(\exists v : \#_v(R_i) \geq n - 2f)$ then $est_i \leftarrow v$ end if end if
13: end while
```

Task T2:

```
14: when valid message DECISION$(r, est)$ is received do
15:    $\forall j \neq i$ : SEND DECISION$(r, est)$ to $p_j$; RETURN$(est)$
16: end when
```

References

[1] G. Bracha. An asynchronous $\lfloor(n-1)/3\rfloor$-resilient consensus protocol. In *Proceedings of the 3rd ACM Symposium on Principles of Distributed Computing*, pages 154–162, Aug. 1984.

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

[3] M. Correia, N. F. Neves, and P. Verissimo. How to tolerate half less one Byzantine nodes in practical distributed systems. In *Proceedings of the 23rd IEEE Symposium on Reliable Distributed Systems*, pages 174–183, Oct. 2004.

[4] A. Doudou, B. Garbinato, and R. Guerraoui. Tolerating arbitrary failures with state machine replication. In H. B. Diab and A. Y. Zomaya, editors, *Dependable Computing Systems Paradigms, Performance Issues, and Applications*, chapter 2. Wiley, 2005.

[5] A. Mostefaoui and M. Raynal. Solving consensus using Chandra-Toueg's unreliable failure detectors: A general quorum-based approach. In *Proceedings of the 13th International Symposium on Distributed Computing*, pages 49–63, 1999.

[6] P. Verissimo. Travelling through wormholes: A new look at distributed systems models. *SIGACT News*, 37(1):66–81, 2006.