Chapter 3

Byzantine Agreement

In order to make flying safer, researchers studied possible failures of various sensors and machines used in airplanes. While trying to model the failures, they were confronted with the following problem: Failing machines did not just crash, instead they sometimes showed arbitrary behavior before stopping completely. With these insights researchers modeled failures as arbitrary failures, not restricted to any patterns.

Definition 3.1 (Byzantine). A node which can have arbitrary behavior is called byzantine. This includes “anything imaginable”, e.g., not sending any messages at all, or sending different and wrong messages to different neighbors, or lying about the input value.

Remarks:

- Byzantine behavior also includes collusion, i.e., all byzantine nodes are being controlled by the same adversary.
- We assume that any two nodes communicate directly, and that no node can forge an incorrect sender address. This is a requirement, such that a single byzantine node cannot simply impersonate all nodes!
- We call non-byzantine nodes correct nodes.

Definition 3.2 (Byzantine Agreement). Finding consensus as in Definition 2.1 in a system with byzantine nodes is called byzantine agreement. An algorithm is f-resilient if it still works correctly with f byzantine nodes.

Remarks:

- As for consensus (Definition 2.1) we also need agreement, termination and validity. Agreement and termination are straight-forward, but what about validity?
3.1 Validity

Definition 3.3 (Any-Input Validity). The decision value must be the input value of any node.

Remarks:
- This is the validity definition we used for consensus, in Definition 2.1.
- Does this definition still make sense in the presence of byzantine nodes? What if byzantine nodes lie about their inputs?
- We would wish for a validity definition which differentiates between byzantine and correct inputs.

Definition 3.4 (Correct-Input Validity). The decision value must be the input value of a correct node.

Remarks:
- Unfortunately, implementing correct-input validity does not seem to be easy, as a byzantine node following the protocol but lying about its input value is indistinguishable from a correct node. Here is an alternative.

Definition 3.5 (All-Same Validity). If all correct nodes start with the same input $v$, the decision value must be $v$.

Remarks:
- If the decision values are binary, then correct-input validity is induced by all-same validity.
- If the input values are not binary, but for example from sensors that deliver values in $\mathbb{R}$, all-same validity is in most scenarios not really useful.

Definition 3.6 (Median Validity). If the input values are orderable, e.g. $v \in \mathbb{R}$, byzantine outliers can be prevented by agreeing on a value close to the median of the correct input values – how close depends on the number of byzantine nodes $f$.

Remarks:
- Is byzantine agreement possible? If yes, with what validity condition?
- Let us try to find an algorithm which tolerates 1 single byzantine node, first restricting to the so-called synchronous model.

Model 3.7 (synchronous). In the synchronous model, nodes operate in synchronous rounds. In each round, each node may send a message to the other nodes, receive the messages sent by the other nodes, and do some local computation.

Definition 3.8 (synchronous runtime). For algorithms in the synchronous model, the runtime is simply the number of rounds from the start of the execution to its completion in the worst case (every legal input, every execution scenario).
3.2 How Many Byzantine Nodes?

Algorithm 3.9 Byzantine Agreement with $f = 1$.

1: Code for node $u$, with input value $x$:

Round 1

2: Send $\text{tuple}(u, x)$ to all other nodes
3: Receive $\text{tuple}(v, y)$ from all other nodes $v$
4: Store all received $\text{tuple}(v, y)$ in a set $S_u$

Round 2

5: Send set $S_u$ to all other nodes
6: Receive sets $S_v$ from all nodes $v$
7: $T = \text{set of } \text{tuple}(v, y) \text{ seen in at least two sets } S_v, \text{ including own } S_u$
8: Let $\text{tuple}(v, y) \in T$ be the tuple with the smallest value $y$
9: Decide on value $y$

Remarks:

- Byzantine nodes may not follow the protocol and send syntactically incorrect messages. Such messages can easily be detected and discarded. It is worse if Byzantine nodes send syntactically correct messages, but with a bogus content, e.g., they send different messages to different nodes.

- Some of these mistakes cannot easily be detected: For example, if a Byzantine node sends different values to different nodes in the first round; such values will be put into $S_u$. However, some mistakes can and must be detected: Observe that all nodes only relay information in Round 2, and do not say anything about their own value. So, if a Byzantine node sends a set $S_v$ which contains a $\text{tuple}(v, y)$, this tuple must be removed by $u$ from $S_v$ upon receiving it (Line 6).

- Recall that we assumed that nodes cannot forge their source address; thus, if a node receives $\text{tuple}(v, y)$ in Round 1, it is guaranteed that this message was sent by $v$.

Lemma 3.10. If $n \geq 4$, all correct nodes have the same set $T$.

Proof. With $f = 1$ and $n \geq 4$ we have at least 3 correct nodes. A correct node will see every correct value at least twice, once directly from another correct node, and once through the third correct node. So all correct values are in $T$. If the Byzantine node sends the same value to at least 2 other (correct) nodes, all correct nodes will see the value twice, so all add it to set $T$. If the Byzantine node sends all different values to the correct nodes, none of these values will end up in any set $T$.

Theorem 3.11. Algorithm 3.9 reaches Byzantine agreement if $n \geq 4$.

Proof. We need to show agreement, any-input validity and termination. With Lemma 3.10 we know that all correct nodes have the same set $T$, and therefore
3.2. **HOW MANY BYZANTINE NODES?**

agree on the same minimum value. The nodes agree on a value proposed by any
node, so any-input validity holds. Moreover, the algorithm terminates after two
rounds.

**Remarks:**

- If \( n > 4 \) the byzantine node can put multiple values into \( T \).
- Algorithm 3.9 only provides any-input agreement, which is question-
able in the byzantine context. One can achieve allsame validity by
choosing the smallest value that occurs at least twice, if a value ap-
ppears at least twice.
- The idea of this algorithm can be generalized for any \( f \) and \( n >
3f \). In the generalization, every node sends in every of \( f + 1 \) rounds
all information it learned so far to all other nodes. In other words,
message size increases exponentially with \( f \).
- Does Algorithm 3.9 also work with \( n = 3 \)?

**Theorem 3.12.** Three nodes cannot reach byzantine agreement with allsame validity if one node among them is byzantine.

**Proof.** We have three nodes \( u, v, w \). In order to achieve allsame validity, a
correct node must decide on its own value if another node supports that value.
The third node might disagree, but that node could be byzantine. If correct
node \( u \) has input 0 and correct node \( v \) has input 1, the byzantine node \( w \) can
fool them by telling \( u \) that its value is 0 and simultaneously telling \( v \) that its
value is 1. This leads to \( u \) and \( v \) deciding on their own values, which results in
violating the agreement condition. Even if \( u \) talks to \( v \), and they figure out that
they have different assumptions about \( w \)'s value, \( u \) cannot distinguish whether
\( w \) or \( v \) is byzantine.

**Theorem 3.13.** A network with \( n \) nodes cannot reach byzantine agreement with \( f \geq n/3 \) byzantine nodes.

**Proof.** Let us assume (for the sake of contradiction) that there exists an algo-
rithm \( A \) that reaches byzantine agreement for \( n \) nodes with \( f \geq n/3 \) byzantine
nodes. With \( A \), we can solve byzantine agreement with 3 nodes. For simplicity,
we call the 3 nodes \( u, v, w \) supernodes.

Each supernode simulates \( n/3 \) nodes, either \( \lfloor n/3 \rfloor \) or \( \lceil n/3 \rceil \), if \( n \) is not
divisible by 3. Each simulated node starts with the input of its supernode. Now
the three supernodes simulate algorithm \( A \). The single byzantine supernode
simulates \( \lfloor n/3 \rfloor \) byzantine nodes. As algorithm \( A \) promises to solve byzantine
agreement for \( f \geq n/3 \), \( A \) has to be able to handle \( \lfloor n/3 \rfloor \) byzantine nodes.
Algorithm \( A \) guarantees that the correct nodes simulated by the correct two
supernodes will achieve byzantine agreement. So the two correct supernodes can
just take the value of their simulated nodes (these values have to be the same by
the agreement property), and we have achieved agreement for three supernodes,
one of them byzantine. This contradicts Lemma 3.12, hence algorithm \( A \) cannot
exist.
3.3 The King Algorithm

Algorithm 3.14 King Algorithm (for $f < n/3$)

1: $x = \text{my input value}$
2: for phase = 1 to $f + 1$ do
   3: Broadcast value($x$)
   4: if some value($y$) at least $n - f$ times then
   5: Broadcast propose($y$)
   6: end if
   7: if some propose($z$) received more than $f$ times then
   8: $x = z$
   9: end if
10: Let node $v_i$ be the predefined king of this phase $i$
11: The king $v_i$ broadcasts its current value $w$
12: if received strictly less than $n - f$ propose($x$) then
13: $x = w$
14: end if
15: end for

Lemma 3.15. Algorithm 3.14 fulfills the all-same validity.

Proof. If all correct nodes start with the same value, all correct nodes propose it in Round 2. All correct nodes will receive at least $n - f$ proposals, i.e., all correct nodes will stick with this value, and never change it to the king’s value. This holds for all phases. 

Lemma 3.16. If a correct node proposes $x$, no other correct node proposes $y$, with $y \neq x$, if $n > 3f$.

Proof. Assume (for the sake of contradiction) that a correct node proposes value $x$ and another correct node proposes value $y$. Since a good node only proposes a value if it heard at least $n - f$ value messages, we know that both nodes must have received their value from at least $n - 2f$ distinct correct nodes (as at most $f$ nodes can behave Byzantine and send $x$ to one node and $y$ to the other one). Hence, there must be a total of at least $2(n - 2f) + f = 2n - 3f$ nodes in the system. Using $3f < n$, we have $2n - 3f > n$ nodes, a contradiction.

Lemma 3.17. There is at least one phase with a correct king.

Proof. There are $f + 1$ phases, each with a different king. As there are only $f$ Byzantine nodes, one king must be correct.

Lemma 3.18. After a round with a correct king, the correct nodes will not change their values $v$ anymore, if $n > 3f$. 
3.4. LOWER BOUND ON NUMBER OF ROUNDS

Proof. If all correct nodes change their values to the king’s value, all correct nodes have the same value. If some correct node does not change its value to the king’s value, it received a proposal at least \( n - f \) times, therefore at least \( n - 2f \) correct nodes broadcasted this proposal. Thus, all correct nodes received it at least \( n - 2f > f \) times (using \( n > 3f \)), therefore all correct nodes set their value to the proposed value, including the correct king. Note that only one value can be proposed more than \( f \) times, which follows from Lemma 3.16. With Lemma 3.15, no node will change its value after this round. \( \square \)

Theorem 3.19. Algorithm 3.14 solves byzantine agreement.

Proof. The king algorithm reaches agreement as either all correct nodes start with the same value, or they agree on the same value latest after the phase where a correct node was king according to Lemmas 3.17 and 3.18. Because of Lemma 3.15 we know that they will stick with this value. Termination is guaranteed after \( 3(f + 1) \) rounds, and all-same validity is proved in Lemma 3.15. \( \square \)

Remarks:

- Algorithm 3.14 requires \( f + 1 \) predefined kings. We assume that the kings (and their order) are given. Finding the kings indeed would be a byzantine agreement task by itself, so this must be done before the execution of the King algorithm.

- Do algorithms exist which do not need predefined kings? Yes, see Section 3.5.

- Can we solve byzantine agreement (or at least consensus) in less than \( f + 1 \) rounds?

3.4 Lower Bound on Number of Rounds

Theorem 3.20. A synchronous algorithm solving consensus in the presence of \( f \) crashing nodes needs at least \( f + 1 \) rounds, if nodes decide for the minimum seen value.

Proof. Let us assume (for the sake of contradiction) that some algorithm \( A \) solves consensus in \( f \) rounds. Some node \( u_1 \) has the smallest input value \( x \), but in the first round \( u_1 \) can send its information (including information about its value \( x \)) to only some other node \( u_2 \) before \( u_1 \) crashes. Unfortunately, in the second round, the only witness \( u_2 \) of \( x \) also sends \( x \) to exactly one other node \( u_3 \) before \( u_2 \) crashes. This will be repeated, so in round \( f \) only node \( u_{f+1} \) knows about the smallest value \( x \). As the algorithm terminates in round \( f \), node \( u_{f+1} \) will decide on value \( x \), all other surviving (correct) nodes will decide on values larger than \( x \). \( \square \)
Remarks:

- A general proof without the restriction to decide for the minimum value exists as well.
- Since byzantine nodes can also just crash, this lower bound also holds for byzantine agreement, so Algorithm 3.14 has an asymptotically optimal runtime.
- So far all our byzantine agreement algorithms assume the synchronous model. Can byzantine agreement be solved in the asynchronous model?

3.5 Asynchronous Byzantine Agreement

Algorithm 3.21 Asynchronous Byzantine Agreement (Ben-Or, for $f < n/10$)

1: $x_i \in \{0, 1\} \quad$ < input bit
2: $r = 1 \quad$ < round
3: decided = false
4: Broadcast $\texttt{propose}(x_i, r)$
5: repeat
6: Wait until $n - f$ $\texttt{propose}$ messages of current round $r$ arrived
7: if at least $n/2 + 3f + 1$ $\texttt{propose}$ messages contain same value $x$ then
8: $x_i = x$, decided = true
9: else if at least $n/2 + f + 1$ $\texttt{propose}$ messages contain same value $x$ then
10: $x_i = x$
11: else
12: choose $x_i$ randomly, with $Pr[x_i = 0] = Pr[x_i = 1] = 1/2$
13: end if
14: $r = r + 1$
15: Broadcast $\texttt{propose}(x_i, r)$
16: until decided (see Line 8)
17: decision = $x_i$

Lemma 3.22. Let a correct node choose value $x$ in Line 10, then no other correct node chooses value $y \neq x$ in Line 10.

Proof. For the sake of contradiction, assume that both 0 and 1 are chosen in Line 10. This means that both 0 and 1 had been proposed by at least $n/2 + 1$ out of $n - f$ correct nodes. In other words, we have a total of at least $2 \cdot n/2 + 2 = n + 2 > n - f$ correct nodes. Contradiction! □

Theorem 3.23. Algorithm 3.21 solves binary byzantine agreement as in Definition 3.2 for up to $f < n/10$ byzantine nodes.

Proof. First note that it is not a problem to wait for $n - f$ propose messages in Line 6, since at most $f$ nodes are byzantine. If all correct nodes have the same input value $x$, then all (except the $f$ byzantine nodes) will propose the same value $x$. Thus, every node receives at least $n - 2f$ propose messages containing $x$. Observe that for $f < n/10$, we get $n - 2f > n/2 + 3f$ and the nodes will
decide on $x$ in the first round already. We have established all-same validity!
If the correct nodes have different (binary) input values, the validity condition
becomes trivial as any result is fine.

What about agreement? Let $u$ be the first node to decide on value $x$ (in
Line 8). Due to asynchrony another node $v$ received messages from a different
subset of the nodes, however, at most $f$ senders may be different. Taking
into account that byzantine nodes may lie (send different propose messages to
different nodes), $f$ additional propose messages received by $v$ may differ from
those received by $u$. Since node $u$ had at least $n/2 + 3f + 1$ propose messages
with value $x$, node $v$ has at least $n/2 + f + 1$ propose messages with value $x$. Hence every correct node will propose $x$ in the next round, and then decide on
$x$.

So we only need to worry about termination: We have already seen that
as soon as one correct node terminates (Line 8) everybody terminates in the
next round. So what are the chances that some node $u$ terminates in Line 8?
Well, we can hope that all correct nodes randomly propose the same value (in
Line 12). Maybe there are some nodes not choosing randomly (entering Line 10
instead of 12), but according to Lemma 3.22 they will all propose the same.

Thus, at worst all $n − f$ correct nodes need to randomly choose the same bit,
which happens with probability $2^{-(n−f)+1}$. If so, all correct nodes will send
the same propose message, and the algorithm terminates. So the expected running
time is exponential in the number of nodes $n$ in the worst case.

Remarks:

- This Algorithm is a proof of concept that asynchronous byzantine
  agreement can be achieved. Unfortunately this algorithm is not useful
  in practice, because of its runtime.

- Note that for $f \in O(n)$, the probability for some node to terminate
  in Line 8 is greater than some positive constant. Thus, the Ben-Or
  algorithm terminates within expected constant number of rounds for
  small values of $f$.

- For a long time, there was no algorithm with subexponential runtime.
  The currently fastest algorithm has an expected runtime of $O(n^{2.5})$
  but only tolerates $f \leq 1/500n$ many byzantine nodes. This algorithm
  works along the lines of the shared coin algorithm; additionally nodes
  try to detect which nodes are byzantine.

Chapter Notes

The project which started the study of byzantine failures was called SIFT and
was founded by NASA [WLG+78], and the research regarding byzantine agree-
ment started to get significant attention with the results by Pease, Shostak, and
Lamport [PSL80, LSP82]. In [PSL80] they presented the generalized version
of Algorithm 3.9 and also showed that byzantine agreement is unsolvable for
$n \leq 3f$. The algorithm presented in that paper is nowadays called Exponential
Information Gathering (EIG), due to the exponential size of the messages.
There are many algorithms for the byzantine agreement problem. For example, the Queen Algorithm \[BG89\] which has a better runtime than the King algorithm \[BGP89\], but tolerates less failures. That byzantine agreement requires at least \(f + 1\) many rounds was shown by Dolev and Strong \[DS83\], based on a more complicated proof from Fischer and Lynch \[FL82\].

While many algorithms for the synchronous model have been around for a long time, the asynchronous model is a lot harder. The only results were by Ben-Or and Bracha. Ben-Or \[Ben83\] was able to tolerate \(f < n/5\). Bracha \[BT85\] improved this tolerance to \(f < n/3\). The first algorithm with a polynomial expected runtime was found by King and Saia \[KS13\]. However, their approach required exponential computation time for each node. In the follow-up paper \[KS14\], same authors improved the local computation time to be polynomial in expectation at the expense of a smaller tolerance \(f < 0.000028n\).

Nearly all developed algorithms only satisfy all-same validity. There are a few exceptions, e.g., correct-input validity \[FG03\], available if the initial values are from a finite domain, or median validity \[SW15\] if the input values are orderable.

Before the term byzantine was coined, the terms Albanian Generals or Chinese Generals were used in order to describe malicious behavior. When the involved researchers met people from these countries they moved – for obvious reasons – to the historic term byzantine \[LSP82\].

Hat tip to Peter Robinson for noting how to improve Algorithm \[3.9\] to all-same validity. This chapter was written in collaboration with Barbara Keller.

Bibliography


