Efficient Byzantine Fault Tolerance

Supplemental Material

Giuliana Santos Veronese, Miguel Correia *Member, IEEE,*
Alysson Neves Bessani, Lau Cheuk Lung and Paulo Verissimo, *Fellow, IEEE*

A. MinBFT Correctness

This section sketches proofs of the correctness of MinBFT. We have to prove that the *safety* property is always satisfied (i.e., that all servers execute the same requests in the same order) and the same for *liveness* (i.e., that all clients' requests are eventually executed).

A.1 Safety

The proof that MinBFT satisfies the safety properties is the following.

Lemma 1 *In a view v, if a correct server executes an operation o with sequence number i, no correct server will execute o with sequence number $i' \neq i$.*

Proof: If a correct server s executes o with sequence number i, it must have received $f+1$ valid COMMIT messages for $\langle o, i\rangle$ from a quorum Q of servers. The proof is by contradiction. Suppose there is another correct server s' that executes o with sequence number $i' > i$. By the MinBFT algorithm, this can only happen if it receives $f+1$ valid COMMIT messages for $\langle o, i'\rangle$, from a quorum Q' of $f+1$ servers. Since $n = 2f+1$ and $|Q|+|Q'| = 2f+2 \geq 2f+1$, there must be at least one server r (called *intersection server*) that sends COMMIT messages both for $\langle o, i\rangle$ and $\langle o, i'\rangle$. Assuming that the primary on view v is the server p, we have to consider four cases:

1) *the primary and the intersection server are correct*: in this case it is trivial to see that the primary will not generate two UIs for the same request operation o, so the intersection server will never send two valid COMMIT messages for $\langle o, i\rangle$ and $\langle o, i'\rangle$.
2) *the primary is correct and the intersection server is faulty*: The (faulty) intersection server would only be able to send valid COMMIT messages for $\langle o, i\rangle$ and $\langle o, i'\rangle$ if these messages contained $UI_p = \langle i, H(o)\rangle_p$ and $UI'_p = \langle i', H(o)\rangle_p$, respectively. Since the primary is correct, it will never invoke `createUI` for the same operation o twice and consequently it will never produce these two UIs.
3) *the primary is faulty and the intersection server is correct*: Now the primary will create PREPARE messages containing $UI_p = \langle i, H(o)\rangle_p$ and $UI'_p = \langle i', H(o)\rangle_p$ and send them to the intersection server in order to try to make it send COMMIT messages both for $\langle o, i\rangle$ and $\langle o, i'\rangle$. However, due to the verification of the seq field of the request operations, which is part of o, the intersection replica will not accept the second PREPARE for the same operation o, issued by client c, because $o.seq = V_{req}[c]$, and the servers only accept operations from clients if their sequence number is greater than their previous number (i.e., if $o.seq > V_{req}[c]$).
4) *the primary and the intersection server are faulty*: Now both the primary and the intersection server are in collusion to make two servers execute the same operation with different numbers. Suppose that the intersection server r sends $\langle \text{COMMIT}, v, r, s, UI_p, UI_r\rangle$ message to s for $\langle o, i\rangle$ and $\langle \text{COMMIT}, v, r, s', UI'_p, UI'_r\rangle$ message to s' for $\langle o, i'\rangle$. Suppose $UI_p = \langle i, H(o)\rangle_p$ and $UI'_p = \langle i', H(o)\rangle_p$ such that $i < i'$. In this case, there are two cases to consider:
   a) s' executed some operation with sequence number i. In this case its sequence of operations does not contain a "hole", but since the USIG service does not allow the primary p to generate two UIs for different message with the same sequence number, the operation with sequence number i must be o (the same executed by s), and thus s' will not execute o again with i' because $o.seq \leq V_{req}[c]$;
   b) s' did not execute some operation with sequence number i. In this case the server will only execute o with sequence number i' if it executed all operations with sequence number $< i'$, and since it did not execute any operation with sequence number i, it will halt waiting for this operation to be executed.

Consequently, it is not possible for two clients to executes the same operation with different sequence number in view v. $\blacksquare_{Lemma\ 1}$

• Giuliana Santos Veronese, Miguel Correia, Alysson Bessani, Paulo Verissimo are with the Departmento de Informática, Faculdade de Ciências, Universidade de Lisboa, Campo Grande, C6, Lisboa, Portugal. Email: giuliana@lasige.di.fc.ul.pt, {bessani,mpc,pjv}@di.fc.ul.pt
• Lau Cheuk Lung is with Departamento de Informática e Estatística, Centro Tecnológico, Universidade Federal de Santa Catarina, Brazil. Email: lau.lung@inf.ufsc.br

Lemma 2 *If a correct server executes an operation o with sequence number i in a view v, no correct server will execute o with sequence number $i' \neq i$ in any view $v' > v$.*

Proof: If a correct server s executes o with sequence number i in a view v, it must have received $f+1$ valid COMMIT messages for $\langle o, i, v\rangle$ from a quorum Q of servers. The proof is again by contradiction. Suppose there is another correct server s' that executes o with sequence number $i' > i$ in a view $v' > v$. By the MinBFT algorithm, this can only happen if it receives $f+1$ valid COMMIT messages for $\langle o, i', v'\rangle$, from a quorum Q' of servers. Since $n = 2f+1$ and $|Q| + |Q'| = 2f+2 \geq 2f+1$, there must be at least one server r (called again *intersection server*) that sends COMMIT messages both for $\langle o, i, v\rangle$ and $\langle o, i', v'\rangle$.

Now let us prove that this leads to a contradiction. For simplicity we start by considering that $v' = v+1$, then we expand the proof for arbitrary values of v'.

First we show that the primary of the new view (p) must assert that o was accepted/executed before v', i.e., that it can not deny this fact. This assertion is done explicitly or implicitly in the new-view certificate V_{vc} that it sends in the NEW-VIEW message that starts view v': $\langle$NEW-VIEW$, p, v', V_{vc}, S, UI_i\rangle$. This certificate is composed by $f+1$ VIEW-CHANGE messages that p received from a quorum Q'' with that many servers, one of which must be correct. Consider a server $r \in Q''$ for which the VIEW-CHANGE message included in V_{vc} is $\langle$VIEW-CHANGE$, r, v', C_{latest}, O, UI_s\rangle$. We have to consider four cases:

1) *the primary p is correct and there is a correct server $r \in Q''$ that executed o*: if p is correct it inserts in V_{vc} $f+1$ VIEW-CHANGE messages, including the one that comes from r. There are two possibilities:
    a) *o was executed after the latest stable checkpoint*: r is correct so O contains the COMMIT message that r sent for o, therefore V_{vc} and S assert explicitly that o was executed.
    b) *o was executed before the latest stable checkpoint*: the execution of o is implicit in C_{latest} so V_{vc} asserts implicitly that o was executed.
2) *the primary p is correct but there is no correct server in Q'' that executed o*: In this situation at least one faulty server $r \in Q''$ accepted o because $|Q| + |Q''| = 2f+2 \geq 2f+1$. Again there are two possibilities:
    a) *o was executed after the latest stable checkpoint*: r might be tempted to not include the COMMIT message for o in O but if it did it p would not put the VIEW-CHANGE message from r in V_{vc}. The reason is that for not putting o in O r would have to do one of two things that would be detected by p: (1) if r executed a request o' after o, r might put the COMMIT message for o' in O but not the message for o, which would leave a "hole" in O that would be detected by p; (2) if r sent the COMMIT message for o with a UI with counter value cv, it might not put in O any COMMIT with $cv' \geq cv$, but that would be detected by p because r would have to sign the VIEW-CHANGE message with a UI with counter value $cv'' > cv + 1$. Therefore, for the VIEW-CHANGE message from r to be inserted in V_{vc} by p, r must include the COMMIT message for o in O, falling in case 1a above.
    b) *o was executed before the latest stable checkpoint*: in this situation the execution of o is implicit in the certificate of the latest stable checkpoint (see case 1b above). The faulty server r may attempt to put an older checkpoint in the VIEW-CHANGE message but p will never insert this message in V_{vc} because r would have to do one of the two detectable things pointed out in case 2a. Therefore, we fall in case 1b.
3) *the primary p is faulty but there is a correct server $r \in Q''$ that executed o*: in this case the faulty primary may attempt to modify the content of O that it inserts in V_{vc}. If it simply removes o from O it leaves a hole, which is detectable. If it removes o and all later messages this is also detectable because p can not forge a UI from r with the following counter value. If p tries to substitute the checkpoint certificate C_{latest} for an older one it also can not forge the UI. Even if it substitutes the checkpoint for an older checkpoint sent by r, this is detectable (servers known that r sent messages afterwards). This shows that these attacks are detectable so we have only to show that they are indeed detected. This is the case because when a correct server receives a NEW-VIEW message it checks the validity of V_{vc}. Therefore, a faulty primary can not tamper the content of the correct server VIEW-CHANGE message so we fall in case 1.
4) *the primary p is faulty and there is no correct server that executed o in Q''*: even if there is no correct server that executed o in Q'', there must be one faulty server $r \in Q''$ that executed o because $|Q| + |Q''| = 2f+2 \geq 2f+1$. For case 2 we already showed that r can not make the primary believe that it did not execute o. However, in this case the primary p is faulty so p can insert the VIEW-CHANGE message sent by r in V_{vc} anyway. However, this falls in case 3 because correct servers will validate V_{vc} when they receive the NEW-VIEW message from p. Therefore we end up falling in case 1.

This shows that the primary p of the new view v' must assert that o was accepted/executed before v' in the new-view certificate V_{vc}. Now we prove that no correct server will execute o with sequence number $i' \neq i$ in view v'. We have to consider two cases:

1) *the primary is correct*: as already shown, the primary must know that o was executed so it will never generate a second UI for the same request and correct servers will not send a COMMIT message for o in view v'.
2) *the primary is faulty*: in this case the primary can create a new PREPARE message containing $UI'_p = \langle i', H(o)\rangle_p$ and send it to the servers, say, to r.

However, r will verify the request number in the seq field of o and discover that $o.seq \leq V_{req}[c]$ meaning that the request was already executed, so it will not execute it again.

This proves the lemma for $v' = v + 1$. Now we have to expand for arbitrary values of v'. There are two cases:

1) *$v' = v + k$ but no requests were accepted in any view v'' such that $v' < v'' < v + k$*: this situation can be caused but an instability in the network that leads to several consecutive executions of the view change algorithm. It is trivial to understand that this case falls into the case of $v' = v + 1$ because nothing relevant happens in the views v''.
2) *the generic case where there are "real" views between v and v'*: an analysis of the proof for the case of $v' = v + 1$ shows that the information that is propagated from view v to $v + 1$ about requests that were executed is also propagated to later views, either explicitly in the O sets while there are no checkpoints, or implicitly in the checkpoints. This is the information used to prevent requests from being re-executed so this case falls into the case of $v' = v + 1$.

$\blacksquare_{Lemma\ 2}$

Theorem 1 *Let s be the correct server that executed more operations of all correct servers up to a certain instant. If s executed the sequence of operations $S = \langle o_1, ...o_i\rangle$, then all other correct servers executed this same sequence of operations or a prefix of it.*

Proof: Let $prefix(S, k)$ be a function that gets the prefix of sequence S containing the first k operations, with $prefix(S, 0)$ being the empty sequence. Let '.' be an operator that concatenates sequences.

Assume that the theorem is false, i.e., that there is a correct server s' that executed some sequence of operations S' that is not a prefix of S. More formally, assume that $prefix(S', j) = prefix(S, j - 1).\langle o'_j\rangle$ and $prefix(S', j - 1).\langle o_j\rangle = prefix(S, j)$, with $o'_j \neq o_j$. In this case o'_j is the j-th operation executed in s' and o_j is the j-th operation executed in s. Assume that o_j was executed in view v by s and o'_j was executed in view v' by s'. If $v = v'$, this contradicts Lemma 1, and if $v \neq v'$ it contradicts Lemma 2. Consequently, the theorem holds. $\blacksquare_{Theorem\ 1}$

A.2 Liveness

In the following we present the proof of liveness for the MinBFT algorithm. We say that an operation request issued by a client c *completes* when c receives the same response for the operation from at least $f + 1$ different servers. We define a *stable view* as a view in which the primary is correct and no timeouts expire at correct replicas.

Lemma 3 *During a stable view, an operation requested by a correct client completes.*

Proof: If the client c is correct it will send its operation o with a sequence number greater than any of its previous requests to all servers. Since, in a stable view the primary p is correct, it will generate an $UI = \langle i, H(o)\rangle_p$ and send it to all servers in a PREPARE message. A correct server will receive this message, verify the validity of UI by calling `verifyUI`, and send a COMMIT message for $\langle o, i\rangle$. Since there are at most f faulty servers on the system, there are at least $f + 1$ correct servers (the primary plus other f servers) that will produce these COMMIT messages and send them to all servers. When a correct server receives $f + 1$ COMMIT messages, it executes o[1]) and send a reply to the client c. When c receives $f + 1$ equal replies the operation completes, which must happen since there are $f + 1$ correct servers and all of them will produce the same result when executing o as their i-th operation. $\blacksquare_{Lemma\ 3}$

Lemma 4 *A view v eventually will be changed to a new view $v' > v$ if at least $f + 1$ correct servers request its change.*

Proof: To request a view change, a correct server s sends a $\langle$REQ-VIEW-CHANGE$, s, v, v'\rangle$ to all servers, where v is the current view number and $v' = v + 1$ the new view number. Consider that a quorum of $f + 1$ correct servers Q requests this view change from view v to view $v + 1$. The primary for the view is by definition $p \triangleq (v + 1)\ mod\ n$. There are two cases:

1) *the view is stable*: this means that all servers in Q receive the REQ-VIEW-CHANGE messages from each another. When one of these servers (s) receives the $f + 1$th of these messages, it sends to all other servers a message $\langle$VIEW-CHANGE$, s, v', C_{latest}, O, UI_s\rangle$. All the VIEW-CHANGE messages sent by servers in Q are received by all servers. The primary p for view $v + 1$ is correct so it sends a message $\langle$NEW-VIEW$, p, v', V_{vc}, S, UI_p\rangle$ to all servers. No timeouts expire (the view is stable) so all servers receive this message and the view changes to $v + 1$.
2) *the view is not stable*: this case can be divided in two cases:
   a) *p is faulty and does not send the* NEW-VIEW *message, or p is faulty and sends an invalid* NEW-VIEW *message that is discarded by all correct servers, or p is not faulty but the communication is slow and the timeout expires in all correct servers*: when the servers send a VIEW-CHANGE message they start a timer that expires after T_{vc} units of time. In this case this timeout expires at all correct servers, which start another view change.
   b) *p is faulty and sends the* NEW-VIEW *message but only to a quorum Q' with at least $f + 1$ servers but less than $f + 1$ correct, or p is correct and the same effect happens due to communication delays*: in this case faulty servers in Q' can follow the algorithm making the correct servers in Q' believe that the algorithm is running normally. More precisely, the servers in Q' can exchange

1. Since the primary p is correct, when it was elected it disseminated any pending requests of the previous view, and thus this server will not have holes in its sequence of operations and will be able to execute the request immediately.

PREPARE and COMMIT messages following the algorithm. At the correct servers that are not in Q', a timer will expire after T_{vc} units of time and these servers will send REQ-VIEW-CHANGE messages, but there will not be $f+1$ one of them so a view change will not happen. When faulty servers start to deviate from the algorithm, requests will stop being accepted, the correct servers in Q' will send REQ-VIEW-CHANGE messages and a view change will start.

In these last two cases (2a and 2b), when another view change starts the system can fall again in one of the cases 1 or 2. However, eventually the view will become stable, the system will fall in case 1 and the view will be changed to a new view $v' > v$. $\blacksquare_{Lemma\ 4}$

Theorem 2 *An operation requested by a correct client eventually completes.*

Proof: The proof comes from the previous lemmas. In stable views, operations requested by correct clients eventually complete (Lemma 3). If the view v is not stable, there are two possibilities:

1) *timers expire and at least $f+1$ correct servers request a view change*: in this case the view will be changed to a new view $v' > v$ (Lemma 4).
2) *less than $f+1$ correct servers request a view change*: this case is similar to case 2b of Lemma 4. If there is a quorum Q of at least $f+1$ servers that do not request the view change and that go on following the algorithm in view v, exchanging PREPARE and COMMIT messages, then the system will stay in view v and requests from correct clients will be executed. When there is no such a quorum or requests are not executed within T_{exec}, all correct servers request a view change and we fall in case 1.

Case 1 leads to a view change but the new view v' is not necessarily stable. The system model makes the assumption that the processing and communication delays do not grow indefinitely (Section 3) and in the algorithm T_{exec} is multiplied by two each time each time a new view change is needed (Section 4). Therefore, even if view changes happen successively, eventually there there will be a view v'' in which one of the following two cases is true:

1) *the primary is correct and no timeouts expire at correct replicas because T_{exec} is greater than the maximum delay observed*: the view is stable, so the operation is executed by Lemma 3.
2) *the primary is faulty*: in this case the primary can deviate from the algorithm causing timeout expiries and a new view change or follow the algorithm enough to avoid a view change. In either case, the view is not stable so we fall in cases 1 or 2 above. Eventually there will be a view in which the primary is correct because only a minority of the replicas are faulty by assumption, so this view will be stable.

$\blacksquare_{Theorem\ 2}$

B. MinZyzzyva Correctness

The two properties that we have to prove about MinZyzzyva are the same that were proved for Zyzzyva. These properties are defined from the point of view of what is observed by a client. Informally, a request is said to have complete if the client can use the reply to that request, i.e., if the client can be certain that the (speculative) execution of that request will not be rolled back. Formally, a request is said to have *complete* at a client if the client received $2f+1$ matching RESPONSE messages or $f+1$ matching LOCAL-COMMIT messages for the request. The properties that MinZyzzyva has to satisfy are:

Safety: If a request with sequence number seq and history h_{seq} completes, then any request that completes with a higher sequence number $seq' > seq$ has a history $h_{seq'}$ that includes h_{seq} as a prefix.

Liveness: Any request issued by a correct client eventually completes.

B.1 Safety

The proof that MinBFT satisfies the safety properties is the following.

Lemma 5 *In a view v, if a correct server executes an operation o with sequence number i, no correct server will execute o with sequence number $i' \neq i$.*

Proof: Despite the differences of MinZyzzyva and MinBFT, the proof of this lemma is similar to the proof of Lemma 1. The basic idea is that replicas process the primary messages by order of counter value in UI and the primary can not associate the same UI to two different requests due to the properties of the USIG service. $\blacksquare_{Lemma\ 5}$

Theorem 3 *If a request with sequence number seq and history h_{seq} completes, then any request that completes with a higher sequence number $seq' > seq$ has a history $h_{seq'}$ that includes h_{seq} as a prefix.*

Proof: Consider that the request o with sequence number seq completes in view v and o' with sequence number seq' in view v'. We have to consider two cases:

1) $v = v'$: the proof is by contradiction. Assume that $h_{seq'}$ does not include h_{seq} as a prefix. This is in contradiction with Lemma 5.
2) $v \neq v'$: suppose that $v' = v+1$ (the expansion for an arbitrary relation $v' > v$ is simple and similar to the one done in the proof of Lemma 2). First we have to show that the primary of the new view (p) must assert that o was completed before v', i.e., that it can not deny this fact. This assertion is done explicitly or implicitly in the new-view certificate V_{vc} that it sends in the NEW-VIEW message that starts view v'. However, the view change operation of MinZyzzyva is almost identical to the same operation of MinBFT so the proof is the same as the one made in the proof

of Lemma 2 and we skip it here. Then, we have to consider two cases:

a) *o' is the first request executed in view v'*: when the new view is installed the NEW-VIEW message serves as a commit certificate to the requests completed in view $v' - 1$. Therefore, $h_{seq'-1}$ is committed and is a prefix of $h_{seq'}$ (the theorem states that o' completes).
b) *o' is not the first request executed in view v'*: this is trivially proved by induction considering case 2a as the base case and using Lemma 5 to prove the induction step.

$\blacksquare_{Theorem\ 3}$

B.2 Liveness

Now we prove the liveness of MinZyzzyva.

Lemma 6 *During a stable view, an operation requested by a correct client completes.*

Proof: If the client c is correct it will send its operation o with a sequence number greater than any of its previous requests to all servers. Since, in a stable view the primary p is correct, it will generate an $UI = \langle i, H(o) \rangle_p$ and send it to all servers and to the client. A correct server will receive this message, verify the validity of UI by calling `verifyUI`, and send a RESPONSE message for $\langle o, i \rangle$ to the client. Since there are at most f faulty servers on the system, there are at least $f+1$ correct servers (the primary plus other f servers) that will produce these RESPONSE messages and send them to the client. There are two cases:

1) *no faulty servers*: the client receives $2f + 1$ RESPONSE messages and the operation completes.
2) *there are faulty servers*: the client receives between $f + 1$ (stable view) and $2f$ RESPONSE messages. When the timer expires the client sends a COMMIT message to the servers, all correct servers reply with a LOCAL-COMMIT message, and the operation completes.

$\blacksquare_{Lemma\ 6}$

Lemma 7 *A view v eventually will be changed to a new view $v' > v$ if at least $f + 1$ correct servers request its change.*

Proof: The view change operation of both algorithms is similar so this proof is similar to the proof of Lemma 4. $\blacksquare_{Lemma\ 7}$

Theorem 4 *An operation requested by a correct client eventually completes.*

Proof: The proof comes from the previous lemmas similarly to the proof of Theorem 2. In stable views, operations requested by correct clients eventually complete (Lemma 6). If the view v is not stable, there are two possibilities:

1) *at least $f + 1$ correct servers suspect of the primary and request a view change*: in this case the view will be changed to a new view $v' > v$ (Lemma 7).
2) *less than $f + 1$ correct servers suspect of the primary and request a view change*: this case is similar to case 2b of Lemma 4. If there is a quorum Q of at least $f+1$ servers that do not request the view change and that go on following the algorithm in view v, then the system will stay in view v and requests from correct clients will be executed. When there is no such a quorum or requests are not executed within the timeout, all correct servers request a view change and we fall in the previous case.

$\blacksquare_{Theorem\ 4}$