Chapter 4

Authenticated Agreement

Byzantine nodes are able to lie about their inputs as well as received messages. Can we detect certain lies and limit the power of byzantine nodes? Possibly, the authenticity of messages may be validated using signatures?

4.1 Agreement with Authentication

Definition 4.1 (Signature). *If a node never signs a message, then no correct node ever accepts that message. We denote a message* $\texttt{msg}(x)$ *signed by node* u *with* $\texttt{msg}(x)_u$*.*

Remarks:

- Algorithm 4.2 shows an agreement protocol for binary inputs relying on signatures. We assume there is a designated "primary" node p. The goal is to decide on p's value.

Algorithm 4.2 Byzantine Agreement using Authentication

```
    Code for primary p:
if input is 1 then
    broadcast $\texttt{value}(1)_p$
    decide 1 and terminate
else
    decide 0 and terminate
end if
    Code for all other nodes v:
for all rounds $i \in 1, \dots, f+1$ do
    $S$ is the set of accepted messages $\texttt{value}(1)_u$.
    if $|S| \geq i$ and $\texttt{value}(1)_p \in S$ then
        broadcast $S \cup \{\texttt{value}(1)_v\}$
        decide 1 and terminate
    end if
end for
decide 0 and terminate
```

Theorem 4.3. *Algorithm 4.2 can tolerate $f < n$ byzantine failures while terminating in $f + 1$ rounds.*

Proof. Assuming that the primary p is not byzantine and its input is 1, then p broadcasts $\texttt{value}(1)_p$ in the first round, which will trigger all correct nodes to decide for 1. If p's input is 0, there is no signed message $\texttt{value}(1)_p$, and no node can decide for 1.

If primary p is byzantine, we need all correct nodes to decide for the same value for the algorithm to be correct. Let us assume that p convinces a correct node v that its value is 1 in round i with $i < f + 1$. We know that v received i signed messages for value 1. Then, v will broadcast $i + 1$ signed messages for value 1, which will trigger all correct nodes to also decide for 1. If p tries to convince some node v late (in round $i = f + 1$), v must receive $f + 1$ signed messages. Since at most f nodes are byzantine, at least one correct node u signed a message $\texttt{value}(1)_u$ in some round $i < f + 1$, which puts us back to the previous case. □

Remarks:

- The algorithm only takes $f + 1$ rounds, which is optimal as described in Theorem 3.20.
- Using signatures, Algorithm 4.2 solves consensus for any number of failures! Does this contradict Theorem 3.12? Recall that in the proof of Theorem 3.12 we assumed that a byzantine node can distribute contradictory information about its own input. If messages are signed, correct nodes can detect such behavior – a node u signing two contradicting messages proves to all nodes that node u is byzantine.
- Does Algorithm 4.2 satisfy any of the validity conditions introduced in Section 3.1? No! A byzantine primary can dictate the decision value. Can we modify the algorithm such that the correct-input validity condition is satisfied? Yes! We can run the algorithm in parallel for $2f + 1$ primary nodes. Either 0 or 1 will occur at least $f + 1$ times, which means that one correct process had to have this value in the first place. In this case, we can only handle $f < \frac{n}{2}$ byzantine nodes.
- In reality, a primary will usually be correct. If so, Algorithm 4.2 only needs two rounds! Can we make it work with arbitrary inputs? Also, relying on synchrony limits the practicality of the protocol. What if messages can be lost or the system is asynchronous?
- Zyzzyva uses authenticated messages to achieve state replication, as in Definition 1.8. It is designed to run fast when nodes run correctly, and it will slow down to fix failures!

4.2 Zyzzyva

Definition 4.4 (View)**.** *A view V describes the current state of a replicated system, enumerating the $3f + 1$ replicas. The view V also marks one of the replicas as the primary p.*

Definition 4.5 (Command)**.** *If a client wants to update (or read) data, it sends a suitable command c in a* `Request` *message to the primary p. Apart from the command c itself, the* `Request` *message also includes a timestamp t. The client signs the message to guarantee authenticity.*

Definition 4.6 (History)**.** *The history h is a sequence of commands $c_1, c_2, \ldots$ in the order they are executed by Zyzzyva. We denote the history up to c_k with h_k.*

Remarks:

- In Zyzzyva, the primary p is used to order commands submitted by clients to create a history h.
- Apart from the globally accepted history, node u may also have a local history, which we denote as h^u or h^u_k.

Definition 4.7 (Complete command)**.** *If a command completes, it will remain in its place in the history h even in the presence of failures.*

Remarks:

- As long as clients wait for the completion of their commands, clients can treat Zyzzyva like one single computer even if there are up to f failures.

In the Absence of Failures

Algorithm 4.8 Zyzzyva: No failures

```
1: At time $t$ client $u$ wants to execute command $c$
2: Client $u$ sends request R = Request($c$,$t$)$_u$ to primary $p$
3: Primary $p$ appends $c$ to its local history, i.e., $h^p = (h^p, c)$
4: Primary $p$ sends OR = OrderedRequest($h^p, c,$ R)$_p$ to all replicas
5: Each replica $r$ appends command $c$ to local history $h^r = (h^r, c)$ and checks
   whether $h^r = h^p$
6: Each replica $r$ runs command $c_k$ and obtains result $a$
7: Each replica $r$ sends Response($a$,OR)$_r$ to client $u$
8: Client $u$ collects the set $S$ of received Response($a$,OR)$_r$ messages
9: Client $u$ checks if all histories $h^r$ are consistent
10: if $|S| = 3f + 1$ then
11:     Client $u$ considers command $c$ to be complete
12: end if
```

Remarks:

- Since the client receives $3f+1$ consistent responses, all correct replicas have to be in the same state.
- Only three communication rounds are required for the command c to complete.

- Note that replicas have no idea which commands are considered complete by clients! How can we make sure that commands that are considered complete by a client are actually executed? We will see in Theorem 4.23.
- Commands received from clients should be ordered according to timestamps to preserve the causal order of commands. We will discuss this in Section 4.3.
- There is a lot of optimization potential. For example, including the entire command history in most messages introduces prohibitively large overhead. Rather, old parts of the history that are agreed upon can be truncated. Also, sending a hash value of the remainder of the history is enough to check its consistency across replicas.
- What if a client does not receive $3f + 1$ `Response`$(a,$`OR`$)_r$ messages? A byzantine replica may omit sending anything at all! In practice, clients set a timeout for the collection of `Response` messages. Does this mean that Zyzzyva only works in the synchronous model? Yes and no. We will discuss this in Lemma 4.26 and Lemma 4.27.

Byzantine Replicas

Algorithm 4.9 Zyzzyva: Byzantine Replicas (append to Algorithm 4.8)

```
1: if $2f+1 \leq |S| < 3f+1$ then
2:     Client $u$ sends Commit$(S)_u$ to all replicas
3:     Each replica $r$ replies with a LocalCommit$(S)_r$ message to $u$
4:     Client $u$ collects at least $2f+1$ LocalCommit$(S)_r$ messages and considers
       $c$ to be complete
5: end if
```

Remarks:

- If replicas fail, a client u may receive less than $3f + 1$ consistent responses from the replicas. Client u can only assume command c to be complete if all correct replicas r eventually append command c to their local history h^r.

Definition 4.10 (Commit Certificate)**.** *A commit certificate S contains $2f+1$ consistent and signed* `Response`$(a,$`OR`$)_r$ *messages from $2f + 1$ different replicas r.*

Remarks:

- The set S is a commit certificate which proves the execution of the command on $2f + 1$ replicas, of which at least $f + 1$ are correct. This commit certificate S must be acknowledged by $2f + 1$ replicas before the client considers the command to be complete.
- Why do clients have to distribute this commit certificate to $2f + 1$ replicas? We will discuss this in Theorem 4.21.

- What if $|S| < 2f + 1$, or what if the client receives $2f + 1$ messages but some have inconsistent histories? Since at most f replicas are byzantine, the primary itself must be byzantine! Can we resolve this?

Byzantine Primary

Definition 4.11 (Proof of Misbehavior)**.** *Proof of misbehavior of some node can be established by a set of contradicting signed messages.*

Remarks:

- For example, if a client u receives two `Response`$(a,$`OR`$)_r$ messages that contain inconsistent `OR` messages signed by the primary, client u can proof that the primary misbehaved. Client u broadcasts this proof of misbehavior to all replicas r which initiate a view change by broadcasting a `IHatePrimary`$_r$ message to all replicas.

Algorithm 4.12 Zyzzyva: Byzantine Primary (append to Algorithm 4.9)

```
if $|S| < 2f + 1$ then
    Client $u$ sends the original R = Request$(c,t)_u$ to all replicas
    Each replica $r$ sends a ConfirmRequest(R)$_r$ message to $p$
    if primary $p$ replies with OR then
        Replica $r$ forwards OR to all replicas
        Continue as in Algorithm 4.8, Line 5
    else
        Replica $r$ initiates view change by broadcasting IHatePrimary$_r$ to all
        replicas
    end if
end if
```

Remarks:

- A faulty primary can slow down Zyzzyva by not sending out the `OrderedRequest` messages in Algorithm 4.8, repeatedly escalating to Algorithm 4.12.
- Line 5 in the Algorithm is necessary to ensure liveness. We will discuss this in Theorem 4.27.
- Again, there is potential for optimization. For example, a replica might already know about a command that is requested by a client. In that case, it can answer without asking the primary. Furthermore, the primary might already know the message `R` requested by the replicas. In that case, it sends the old `OR` message to the requesting replica.

Safety

Definition 4.13 (Safety)**.** *We call a system safe if the following condition holds: If a command with sequence number j and a history h_j completes, then for any command that completed earlier (with a smaller sequence number $i < j$), the history h_i is a prefix of history h_j.*

Remarks:

- In Zyzzyva a command can only complete in two ways, either in Algorithm 4.8 or in Algorithm 4.9.
- If a system is safe, complete commands cannot be reordered or dropped. So is Zyzzyva so far safe?

Lemma 4.14. *Let c_i and c_j be two different complete commands. Then c_i and c_j must have different sequence numbers.*

Proof. If a command c completes in Algorithm 4.8, $3f + 1$ replicas sent a `Response`$(a,$`OR`$)_r$ to the client. If the command c completed in Algorithm 4.9, at least $2f + 1$ replicas sent a `Response`$(a,$`OR`$)_r$ message to the client. Hence, a client has to receive at least $2f + 1$ `Response`$(a,$`OR`$)_r$ messages.

Both c_i and c_j are complete. Therefore there must be at least $2f+1$ replicas that responded to c_i with a `Response`$(a,$`OR`$)_r$ message. But there are also at least $2f + 1$ replicas that responded to c_j with a `Response`$(a,$`OR`$)_r$ message. Because there are only $3f + 1$ replicas, there is at least one correct replica that sent a `Response`$(a,$`OR`$)_r$ message for both c_i and c_j. A correct replica only sends one `Response`$(a,$`OR`$)_r$ message for each sequence number, hence the two commands must have different sequence numbers. □

Lemma 4.15. *Let c_i and c_j be two complete commands with sequence numbers $i < j$. The history h_i is a prefix of h_j.*

Proof. As in the proof of Lemma 4.14, there has to be at least one correct replica that sent a `Response`$(a,$`OR`$)_r$ message for both c_i and c_j.

A correct replica r that sent a `Response`$(a,$`OR`$)_r$ message for c_i will only accept c_j if the history for c_j provided by the primary is consistent with the local history of replica r, including c_i. □

Remarks:

- A byzantine primary can cause the system to never complete any command. Either by never sending any messages or by inconsistently ordering client requests. In this case, replicas have to replace the primary.

View Changes

Definition 4.16 (View Change)**.** *In Zyzzyva, a view change is used to replace a byzantine primary with another (hopefully correct) replica. View changes are initiated by replicas sending* `IHatePrimary`$_r$ *to all other replicas. This only happens if a replica obtains a valid proof of misbehavior from a client or after a replica fails to obtain an* `OR` *message from the primary in Algorithm 4.12.*

Remarks:

- How can we safely decide to initiate a view change, i.e. demote a byzantine primary? Note that byzantine nodes should not be able to trigger a view change!

Algorithm 4.17 Zyzzyva: View Change Agreement

```
1: All replicas continuously collect the set $H$ of IHatePrimary$_r$ messages
2: if a replica $r$ received $|H| > f$ messages or a valid ViewChange message then
3:     Replica $r$ broadcasts ViewChange$(H,h^r,S)_r$
4:     Replica $r$ stops participating in the current view
5:     Replica $r$ switches to the next primary "$p = p + 1$"
6: end if
```

Remarks:

- The $f + 1$ IHatePrimary$_r$ messages in set H prove that at least one correct replica initiated a view change. This proof is broadcast to all replicas to make sure that once the first correct replica stopped acting in the current view, all other replicas will do so as well.
- S is the most recent commit certificate that the replica with support at least $2f+1$, obtained in the ending view as described in Algorithm 4.9. S will be used to recover the correct history before the new view starts. The local histories h^r are included in the ViewChange$(H,h^r,S)_r$ message such that commands that completed after a correct client received $3f + 1$ responses from replicas can be recovered as well.
- In Zyzzyva, a byzantine primary starts acting as a normal replica after a view change. In practice, all machines eventually break and rarely fix themselves after that. Instead, one could consider to replace a byzantine primary with a fresh replica that was not in the previous view.

Algorithm 4.18 Zyzzyva: View Change Execution

```
1: The new primary $p$ collects the set $C$ of ViewChange$(H,h^r,S)_r$ messages
2: if new primary $p$ collected $|C| \geq 2f + 1$ messages then
3:     New primary $p$ sends NewView$(C)_p$ to all replicas
4: end if

5: if a replica $r$ received a NewView$(C)_p$ message then
6:     Replica $r$ recovers new history $h_{\texttt{new}}$ as shown in Algorithm 4.22
7:     Replica $r$ broadcasts ViewConfirm$(h_{\texttt{new}})_r$ message to all replicas
8: end if

9: if a replica $r$ received $2f + 1$ ViewConfirm$(h_{\texttt{new}})_r$ messages then
10:     Replica $r$ accepts $h^r = h_{\texttt{new}}$ as the history of the new view
11:     Replica $r$ starts participating in the new view
12: end if
```

Remarks:

- Analogously to Lemma 4.15 commit certificates are ordered. For two commit certificates S_i and S_j with sequence numbers $i < j$, the history h_i certified by S_i is a prefix of the history h_j certified by S_j.
- Zyzzyva collects the most recent commit certificate and the local history of $2f + 1$ replicas. This information is distributed to all replicas, and used to recover the history for the new view.
- If a replica does not receive the $\texttt{NewView}(C)_p$ and $\texttt{ViewConfirm}(h_{\texttt{new}})_r$ message in time, it triggers another view change by boradcasting $\texttt{IHatePrimary}_r$ to all other replicas.
- How is the history recovered exactly? It seems that the set of histories included in C can be messy. How can we be sure that complete commands are not reordered or dropped?

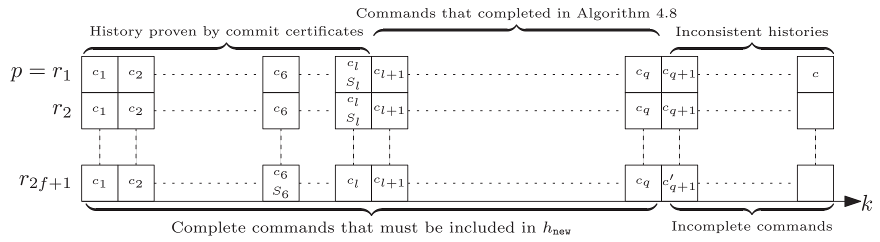
Commands that completed in Algorithm 4.8
History proven by commit certificates
Inconsistent histories
$p = r_1$
r_2
r_{2f+1}
c_1
c_2
c_6
c_l
S_l
c_{l+1}
c_q
c_{q+1}
c
S_6
c'_{q+1}
k
Complete commands that must be included in $h_{\texttt{new}}$
Incomplete commands

Figure 4.19: Structure of the data reported by different replicas in C. Starting with a consistent history, there can be a number of commit certificates S_i. Commands up to the last commit certificate S_l were completed in either Algorithm 4.8 or Algorithm 4.9. After the last commit certificate S_l there may be commands that completed at a correct client (c_l through c_q) in Algorithm 4.8. These commands c_{l+1} through c_q are at least consistent in all histories h^r reported by correct replicas, i.e., by at least $f + 1$ correct replicas. After that, there may be commands that are not reported consistently by the correct replicas. These commands cannot have completed at a correct client since there is no commit certificate available.

Remarks:

- Because c_l may have completed at a correct client, all commands up to the most recent commit certificate S_l have to be carried over into the new view. These are easy to recover since S_l includes the entire history up to sequence number l. But how can we be sure to find the most recent commit certificate?
- Also, there can be commands that complete in Algorithm 4.8, after the last commit certificate! Can we safely recover these commands from C?

Lemma 4.20. *The globally most recent commit certificate S_l is included in C.*

Proof. Any two sets of $2f+1$ replicas share at least one correct replica. Hence, at least one correct replica which acknowledged the most recent commit certificate S_l also sent a $\texttt{LocalCommit}(S_l)_r$ message that is in C. □

Lemma 4.21. *Any command and its history that completes after S_l has to be reported in C at least $f+1$ times.*

Proof. A command c can only complete in Algorithm 4.8 after S_l. Hence, $3f+1$ replicas sent a $\texttt{Response}(a,\texttt{OR})_r$ message for c. C includes the local histories of $2f+1$ replicas of which at most f are byzantine. As a result, c and its history is consistently found in at least $f+1$ local histories in C. □

Algorithm 4.22 Zyzzyva: History Recovery

```
1: $C$ = set of $2f+1$ $\texttt{ViewChange}(H,h^r,S)_r$ messages in $\texttt{NewView}(C)_p$
2: $R$ = set of replicas included in $C$
3: $S_l$ = most recent commit certificate reported in $C$
4: $k = l+1$, next sequence number
5: $h_{\texttt{new}}$ = history $h_l$ contained in $S_l$
6: **while** command $c_k$ exists in $C$ **do**
7:    **if** $c_k$ reported by more than $f+1$ replicas in $R$ **then**
8:       Remove replicas from $R$ that do not support $c_k$
9:       $h_{\texttt{new}} = (h_{\texttt{new}}, c_k)$
10:   **end if**
11:   $k = k+1$
12: **end while**
13: return $h_{\texttt{new}}$
```

Remarks:

- Commands up to S_l are included into the history.
- If at least $f+1$ replicas share a consistent history after that, also the commands after that are included.
- It may be that a command c that did not complete at a client is included into the new history $h_{\texttt{new}}$. For example, C may contain $f+2$ correct replicas of which one may be unaware of c. However, replicas can use the timestamp included in the command to avoid duplicate execution; if a client u re-issues command c again, and c is already in the history, the correct replicas will return the result of the previous execution.
- Can we be sure that all commands that completed at a correct client are carried over into the new view?

Lemma 4.23. *If a command c is considered complete by a client, command c remains in its place in the history even if the view changes.*

Proof. We have shown in Lemma 4.21 that the most recent commit certificate is contained in C, and hence any command that terminated in Algorithm 4.9 is included in the new history after a view change. What about commands that completed in Algorithm 4.8, after the last commit certificate? According to Lemma 4.21 they also have enough support of $f+1$ correct replicas. Commands can only complete after the most recent commit certificate if the client received $3f+1$ $\texttt{Response}(a,\texttt{OR})_r$ messages. This means that every correct replica will include such a command into their local history h^r. Therefore, the request must be supported by at least $f+1$ replicas in the set of $2f+1$ `ViewChange` messages collected by the primary. Such commands are carried over into the new history as described in Algorithm 4.22. □

Theorem 4.24. *Zyzzyva is safe even during view changes.*

Proof. Complete commands are not reordered within a view as described in Lemma 4.15. Also, no complete command is lost or reordered during a view change as shown in Lemma 4.23. Hence, Zyzzyva is safe. □

Liveness

Definition 4.25 (Liveness)**.** *We call a system live if every command eventually completes.*

Lemma 4.26. *Zyzzyva is live during periods of synchrony if the primary is correct and a command is requested by a correct client.*

Proof. The client receives a $\texttt{Response}(a,\texttt{OR})_r$ message from all correct replicas. If it receives $3f+1$ messages, the command completes immediately. If the client receives fewer than $3f+1$ messages, it will at least receive $2f+1$ of them, since there are at most f byzantine replicas. All correct replicas will answer the clients $\texttt{Commit}(S)_u$ message with a correct $\texttt{LocalCommit}(S)_r$ message after which the command completes. □

Lemma 4.27. *If, during a period of synchrony, a request does not complete in Algorithm 4.8 or Algorithm 4.9, a view change occurs.*

Proof. If a command does not complete for a sufficiently long time, the client will resend the $\texttt{R} = \texttt{Request}(c,t)_u$ message to all replicas. If a replica's $\texttt{ConfirmRequest}(\texttt{R})_r$ message is not answered in time by the primary, it broadcasts an $\texttt{IHatePrimary}_r$ message. If a correct replica gathers $f+1$ $\texttt{IHatePrimary}_r$ messages, the view change is initiated. If no correct replica collects more than f $\texttt{IHatePrimary}_r$ messages, at least one correct replica received a valid $\texttt{OrderedRequest}(h^p, c, \texttt{R})_p$ message from the primary which it forwards to all other replicas. In that case, the client is guaranteed to receive at least $2f+1$ $\texttt{Response}(a,\texttt{OR})_r$ messages from the correct replicas and can complete the command by assembling a commit certificate. □

Remarks:

- If the newly elected primary is byzantine, the view change may never terminate. However, we can detect if the new primary does not assemble C correctly as all contained messages are signed. If the primary refuses to assemble C, replicas initiate another view change with after a timeout.

4.3 Strong consistency