Consensus and Reliable Broadcast
Broadcast

ользователям обмена сообщениями по сети. Могут быть организованы различные виды сообщений: влайне, врлые, всле...
Broadcast

If a process sends a message $m$, then every process eventually delivers $m$
Broadcast

If a process sends a message \( m \), then every process eventually delivers \( m \)

How can we adapt the spec for an environment where processes can fail? And what does “fail” mean?
A hierarchy of failure models

Crash
A hierarchy of failure models
A hierarchy of failure models

- Fail-stop
- Send Omission
- Receive Omission
- Crash
A hierarchy of failure models

Fail-stop → Crash
Send Omission → Receive Omission
General Omission

Crash

Send Omission
Receive Omission
General Omission
A hierarchy of failure models

Fail-stop → Crash
Send Omission → Receive Omission
General Omission

benign failures
A hierarchy of failure models

- Fail-stop
- Crash
- Send Omission
- Receive Omission
- General Omission
- Arbitrary failures with message authentication
A hierarchy of failure models

- Fail-stop
- Crash
- Send Omission
- Receive Omission
- General Omission
- Arbitrary failures with message authentication
- Arbitrary (Byzantine) failures

Benign failures
Reliable Broadcast

Validity
If the sender is correct and broadcasts a message $m$, then all correct processes eventually deliver $m$.

Agreement
If a correct process delivers a message $m$, then all correct processes eventually deliver $m$.

Integrity
Every correct process delivers at most one message, and if it delivers $m$, then some process must have broadcast $m$. 
## Terminating Reliable Broadcast

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Validity</strong></td>
<td>If the sender is correct and broadcasts a message ( m ), then all correct processes eventually deliver ( m )</td>
</tr>
<tr>
<td><strong>Agreement</strong></td>
<td>If a correct process delivers a message ( m ), then all correct processes eventually deliver ( m )</td>
</tr>
<tr>
<td><strong>Integrity</strong></td>
<td>Every correct process delivers at most one message, and if it delivers ( m \neq SF ), then some process must have broadcast ( m )</td>
</tr>
<tr>
<td><strong>Termination</strong></td>
<td>Every correct process eventually delivers some message</td>
</tr>
</tbody>
</table>
Consensus

Validity
If all processes that propose a value propose $v$, then all correct processes eventually decide $v$

Agreement
If a correct process decides $v$, then all correct processes eventually decide $v$

Integrity
Every correct process decides at most one value, and if it decides $v$, then some process must have proposed $v$

Termination
Every correct process eventually decides some value
Properties of send(m) and receive(m)

Benign failures:

Validity If \( p \) sends \( m \) to \( q \), and \( p, q \), and the link between them are correct, then \( q \) eventually receives \( m \)

Uniform* Integrity For any message \( m \), \( q \) receives \( m \) at most once from \( p \), and only if \( p \) sent \( m \) to \( q \)

* A property is uniform if it applies to both correct and faulty processes
Properties of send($m$) and receive($m$)

Arbitrary failures:

**Integrity**  For any message $m$, if $p$ and $q$ are correct then $q$ receives $m$ at most once from $p$, and only if $p$ sent $m$ to $q$
Questions, Questions...

- Are these problems solvable at all?
- Can they be solved independent of the failure model?
- Does solvability depend on the ratio between faulty and correct processes?
- Does solvability depend on assumptions about the reliability of the network?
- Are the problems solvable in both synchronous and asynchronous systems?
- If a solution exists, how expensive is it?
Plan

Synchronous Systems
- Consensus for synchronous systems with crash failures
- Lower bound on the number of rounds
- Reliable Broadcast for arbitrary failures with message authentication
- Lower bound on the ratio of faulty processes for Consensus with arbitrary failures
- Reliable Broadcast for arbitrary failures

Asynchronous Systems
- Impossibility of Consensus for crash failures
- Failure detectors
- PAXOS
Model

- Synchronous Message Passing
  - Execution is a sequence of rounds
  - In each round every process takes a step
    - sends messages to neighbors
    - receives messages sent in that round
    - changes its state

- Network is fully connected (an $n$-clique)

- No communication failures
A simple Consensus algorithm

Process $p_i$:

Initially $V = \{v_i\}$

To execute $\text{propose}(v_i)$

1: send $\{v_i\}$ to all

$\text{decide}(x)$ occurs as follows:

2: for all $j, 0 \leq j \leq n-1, j \neq i$ do

3: receive $S_j$ from $p_j$

4: $V := V \cup S_j$

5: decide $\min(V)$
An execution
An execution
An execution
An execution

Suppose $v_1 = v_3 = v_4$ at the end of round 1. Can $p_3$ decide?
An execution

Suppose $v_1 = v_3 = v_4$ at the end of round 1. Can $p_3$ decide?
An execution

Suppose $v_1 = v_3 = v_4$ at the end of round 1. Can $p_3$ decide?
An execution

Suppose $v_1 = v_3 = v_4$ at the end of round 1. Can $p_3$ decide?

$v_1 = v_3 = v_4$
An execution

Suppose $v_1 = v_3 = v_4$ at the end of round 1.
Can $p_3$ decide?
An execution

Suppose $v_1 = v_3 = v_4$ at the end of round 1. Can $p_3$ decide?
Suppose $v_1 = v_3 = v_4$ at the end of round 1. Can $p_3$ decide?
Echoing values

A process that receives a proposal in round 1, relays it to others during round 2.
Echoing values

A process that receives a proposal in round 1, relays it to others during round 2.

Suppose $p_3$ hasn’t heard from $p_2$ at the end of round 2. Can $p_3$ decide?
Echoing values

A process that receives a proposal in round 1, relays it to others during round 2.

Suppose $p_3$ hasn’t heard from $p_2$ at the end of round 2. Can $p_3$ decide?
Echoing values

A process that receives a proposal in round 1, relays it to others during round 2.

Suppose $p_3$ hasn't heard from $p_2$ at the end of round 2. Can $p_3$ decide?
Echoing values

A process that receives a proposal in round 1, relays it to others during round 2.

Suppose $p_3$ hasn’t heard from $p_2$ at the end of round 2. Can $p_3$ decide?
Echoing values

A process that receives a proposal in round 1, relays it to others during round 2.

Suppose $p_3$ hasn't heard from $p_2$ at the end of round 2. Can $p_3$ decide?
Echoing values

A process that receives a proposal in round 1, relays it to others during round 2.

Suppose $p_3$ hasn’t heard from $p_2$ at the end of round 2. Can $p_3$ decide?
What is going on

A correct process $p^*$ has not received all proposals by the end of round $i$. Can $p^*$ decide?

Another process may have received the missing proposal at the end of round $i$ and be ready to relay it in round $i + 1$. 


Dangerous Chains

Dangerous chain
The last process in the chain is correct, all others are faulty

\[
p_0 \\ p_1 \\ p_2 \\ p_{i-1} \\ p_i \\ p^* \]

rounds 3...i − 1

round i
Living dangerously

How many rounds can a dangerous chain span?

- $f$ faulty processes
- at most $f+1$ nodes in the chain
- spans at most $f$ rounds

It is safe to decide by the end of round $f+1$!
The Algorithm

Code for process $p_i$:

Initially $V = \{v_i\}$

To execute $\text{propose}(v_i)$

round $k$, $1 \leq k \leq f + 1$

1: send $\{v \in V : p_i \text{ has not already sent } v\}$ to all

2: for all $j$, $0 \leq j \leq n-1$, $j \neq i$ do

3: receive $S_j$ from $p_j$

4: $V := V \cup S_j$

$\text{decide}(x)$ occurs as follows:

5: if $k = f + 1$ then

6: decide $\text{min}(V)$
Termination and
Integrity

Initially \( V = \{v_i\} \)

To execute propose(\( v_i \))

round \( k, 1 \leq k \leq f+1 \)

1: \( \text{send } \{v \in V : p_i \text{ has not already sent } v\} \text{ to all} \)

2: \( \text{for all } j, 0 \leq j \leq n - 1, j \neq i \text{ do} \)

3: \( \text{receive } S_j \text{ from } p_j \)

4: \( V := V \cup S_j \)

\( \text{decide}(x) \) occurs as follows:

5: \( \text{if } k = f + 1 \text{ then} \)

6: \( \text{decide } \min(V) \)

Termination
Termination and Integrity

Initially $V = \{v_i\}$

To execute $\text{propose}(v_i)$

round $k$, $1 \leq k \leq f + 1$

1: send $\{v \in V : p_i \text{ has not already sent } v\}$ to all

2: for all $j$, $0 \leq j \leq n - 1$, $j \neq i$ do

3: receive $S_j$ from $p_j$

4: $V := V \cup S_j$

$\text{decide}(x)$ occurs as follows:

5: if $k = f + 1$ then

6: decide $\min(V)$

Termination

Every correct process

reach round $f + 1$

Decides on $\min(V)$ which is well defined
Termination and Integrity

Initially \( V = \{v_i\} \)

To execute \( \text{propose}(v_i) \)
  round \( k, 1 \leq k \leq f + 1 \)
1:  \( \text{send} \{v \in V : p_i \text{ has not already sent } v\} \text{ to all} \)
2:  \( \text{for all } j, 0 \leq j \leq n-1, j \neq i \text{ do} \)
3:  \( \text{receive } S_j \text{ from } p_j \)
4:  \( V := V \cup S_j \)

decide(x) occurs as follows:
5:  \( \text{if } k = f + 1 \text{ then} \)
6:  \( \text{decide min}(V) \)

Termination

Every correct process
- reaches round \( f + 1 \)
- Decides on \( \min(V) \) --- which is well defined

Integrity

At most one value:

Only if it was proposed:
Termination and Integrity

Initially \( V = \{v_i\} \)

To execute \text{propose}(v_i)

round \( k, 1 \leq k \leq f + 1 \)

1: send \( \{v \in V : p_i \text{ has not already sent } v\} \) to all

2: for all \( j, 0 \leq j \leq n - 1, j \neq i \) do

3: receive \( S_j \) from \( p_j \)

4: \( V := V \cup S_j \)

decide(\( x \)) occurs as follows:

5: if \( k = f + 1 \) then

6: decide \( \min(V) \)

**Termination**

Every correct process

- reaches round \( f + 1 \)
- Decides on \( \min(V) \) — which is well defined

**Integrity**

At most one value:
- one decide, and \( \min(V) \) is unique

Only if it was proposed:
**Termination and Integrity**

**Termination**

Every correct process

- reaches round $f + 1$
- Decides on $\min(V)$ --- which is well defined

**Integrity**

1. Initially $V = \{v_i\}$
2. To execute `propose(v_i)`
   - round $k$, $1 \leq k \leq f + 1$
   - send $\{v \in V : p_i \text{ has not already sent } v\}$ to all
3. for all $j$, $0 \leq j \leq n - 1$, $j \neq i$ do
4. receive $S_j$ from $p_j$
5. $V := V \cup S_j$

`decide(x)` occurs as follows:

- if $k = f + 1$ then
- decide $\min(V)$

---

**At most one value:**

- one decide, and $\min(V)$ is unique

**Only if it was proposed:**

- To be decided upon, must be in $V$ at round $f+1$
  - if value $= v_i$, then it is proposed in round 1
  - else, suppose received in round $k$. By induction:
    - $k = 1$:
      - by Uniform Integrity of underlying send and receive, it must have been sent in round 1
      - by the protocol and because only crash failures, it must have been proposed
    - Induction Hypothesis: all values received up to round $k = j$ have been proposed
  - $k = j+1$
    - sent in round $j+1$ (Uniform Integrity of send and synchronous model)
    - must have been part of $V$ of sender at end of round $j$
    - by protocol, must have been received by sender by end of round $j$
    - by induction hypothesis, must have been proposed
Validity

Initially \( V = \{v_i\} \)

To execute \texttt{propose}(\(v_i\))

round \( k, 1 \leq k \leq f + 1 \)

1: send \( \{v \in V: p_i \text{ has not already sent } v\} \) to all

2: for all \( j, 0 \leq j \leq n - 1, j \neq i \) do

3: receive \( S_j \) from \( p_j \)

4: \( V := V \cup S_j \)

decide(\(x\)) occurs as follows:

5: if \( k = f + 1 \) then

6: decide \( \min(V) \)
Validity

Initially $V = \{v_i\}$

To execute $\text{propose}(v_i)$

round $k, 1 \leq k \leq f+1$

1: send $\{v \in V : p_i \text{ has not already sent } v\}$ to all

2: for all $j, 0 \leq j \leq n-1, j \neq i$ do

3: receive $S_j$ from $p_j$

4: $V := V \cup S_j$

decide($x$) occurs as follows:

5: if $k = f+1$ then

6: decide $\min(V)$

Suppose every process proposes $v^*$

Since only crash model, only $v^*$ can be sent

By Uniform Integrity of send and receive, only $v^*$ can be received

By protocol, $V = \{v^*\}$

$\min(V) = v^*$

$\text{decide}(v^*)$
Agreement

Lemma 1
For any $r \geq 1$, if a process $p$ receives a value $v$ in round $r$, then there exists a sequence of processes $p_0, p_1, \ldots, p_r$ such that $p_r = p$, $p_0$ is $v$'s proponent, and in each round $p_{k-1}$ sends $v$ and $p_k$ receives it. Furthermore, all processes in the sequence are distinct.

Proof
By induction on the length of the sequence
Agreement

Initially $V = \{v_i\}$

To execute $\text{propose}(v_i)$

round $k$, $1 \leq k \leq f + 1$

1: send $\{v \in V : p_i \text{ has not already sent } v\}$ to all
2: for all $j$, $0 \leq j \leq n - 1$, $j \neq i$ do
3: receive $S_j$ from $p_j$
4: $V := V \cup S_j$

decide($x$) occurs as follows:
5: if $k = f + 1$ then
6: decide $\min(V)$

Lemma 2:

In every execution, at the end of round $f + 1$, $V_i = V_j$ for every correct processes $p_i$ and $p_j$
Agreement

Initially $V = \{v_i\}$

To execute $\text{propose}(v_i)$

round $k$, $1 \leq k \leq f + 1$

1: send $\{v \in V : p_i \text{ has not already sent } v\}$ to all

2: for all $j$, $0 \leq j \leq n - 1$, $j \neq i$ do

3: receive $S_j$ from $p_j$

4: $V := V \cup S_j$

$\text{decide}(x)$ occurs as follows:

5: if $k = f + 1$ then

6: decide $\text{min}(V)$

Lemma 2:

In every execution, at the end of round $f + 1$, $V_i = V_j$ for every correct processes $p_i$ and $p_j$

Agreement follows from Lemma 2, since $\text{min}$ is a deterministic function
Agreement

Proof:

- Show that if a correct $p$ has $x$ in its $V$ at the end of round $f+1$, then every correct $p$ has $x$ in its $V$ at the end of round $f+1$

 Initially $V = \{v_i\}$

To execute $\text{propose}(v_i)$

  round $k$, $1 \leq k \leq f+1$

1: send $\{v \in V : p_i \text{ has not already sent } v\}$ to all

2: for all $j$, $0 \leq j \leq n-1$, $j \neq i$ do

3: receive $S_j$ from $p_j$

4: $V := V \cup S_j$

$\text{decide}(x)$ occurs as follows:

5: if $k = f+1$ then

6: decide $\text{min}(V)$

Lemma 2:

In every execution, at the end of round $f+1$, $V_i = V_j$ for every correct processes $p_i$ and $p_j$

Agreement follows from Lemma 2, since $\text{min}$ is a deterministic function
Agreement

Proof:

• Show that if a correct $p$ has $x$ in its $V$ at the end of round $f+1$, then every correct $p$ has $x$ in its $V$ at the end of round $f+1$

• Let $r$ be earliest round $x$ is added to the $V$ of a correct $p$. Let that process be $p^*$

• If $r \leq f$, then $p^*$ sends $x$ in round $r+1 \leq f+1$; every correct process receives $x$ and adds $x$ to its $V$ in round $r+1$

Lemma 2:

In every execution, at the end of round $f+1$, $V_i = V_j$ for every correct processes $p_i$ and $p_j$

Agreement follows from Lemma 2, since $\text{min}$ is a deterministic function

Initially $V = \{v_i\}$

To execute $\text{propose}(v_i)$

round $k, 1 \leq k \leq f+1$

1: send $\{v \in V : p_i \text{ has not already sent } v\}$ to all
2: for all $j, 0 \leq j \leq n-1, j \neq i$ do
3: receive $S_j$ from $p_j$
4: $V := V \cup S_j$

decide($x$) occurs as follows:
5: if $k = f+1$ then
6: decide $\text{min}(V)$
Agreement

Lemma 2:
In every execution, at the end of round $f + 1$, $V_i = V_j$ for every correct processes $p_i$ and $p_j$

Agreement follows from Lemma 2, since min is a deterministic function

Proof:
• Show that if a correct $p$ has $x$ in its $V$ at the end of round $f + 1$, then every correct process has $x$ in its $V$ at the end of round $f + 1$
• Let $r$ be earliest round $x$ is added to the $V$ of a correct $p$. Let that process be $p^*$
• If $r \leq f$, then $p^*$ sends $x$ in round $r + 1 \leq f + 1$; every correct process receives $x$ and adds $x$ to its $V$ in round $r + 1$
• What if $r = f + 1$?

Initially $V = \{v_i\}$

To execute propose($v_i$)
• round $k, 1 \leq k \leq f + 1$
  1: send $\{v \in V : p_i \text{ has not already sent } v\}$ to all
  2: for all $j, 0 \leq j \leq n - 1, j \neq i$ do
  3: receive $S_j$ from $p_j$
  4: $V := V \cup S_j$

decide($x$) occurs as follows:
  5: if $k = f + 1$ then
  6: decide min($V$)
Agreement

Lemma 2:
In every execution, at the end of round \( f + 1 \), \( V_i = V_j \) for every correct processes \( p_i \) and \( p_j \).

Agreement follows from Lemma 2, since \( \text{min} \) is a deterministic function.

Proof:
- Show that if a correct \( p \) has \( x \) in its \( V \) at the end of round \( f + 1 \), then every correct \( p \) has \( x \) in its \( V \) at the end of round \( f + 1 \).
- Let \( r \) be earliest round \( x \) is added to the \( V \) of a correct \( p \). Let that process be \( p^* \).
- If \( r \leq f \), then \( p^* \) sends \( x \) in round \( f + 1 \); every correct process receives \( x \) and adds \( x \) to its \( V \) in round \( r + 1 \).
- What if \( r = f + 1 \)?
  - By Lemma 1, there exists a sequence of distinct processes \( p_0, \ldots, p_{f+1} = p^* \).
  - Consider processes \( p_0, \ldots, p_f \).
  - \( f + 1 \) processes; only \( f \) faulty.
  - one of \( p_0, \ldots, p_f \) is correct, and adds \( x \) to its \( V \) before \( p^* \) does it in round \( r \).

CONTRADICTION!
Terminating Reliable Broadcast

Validity
If the sender is correct and broadcasts a message $m$, then all correct processes eventually deliver $m$.

Agreement
If a correct process delivers a message $m$, then all correct processes eventually deliver $m$.

Integrity
Every correct process delivers at most one message, and if it delivers $m \neq SF$, then some process must have broadcast $m$.

Termination
Every correct process eventually delivers some message.
TRB for benign failures

Sender in round 1:
1: send m to all

Process p in round $k$, $1 \leq k \leq f+1$
1: if delivered m in round $k-1$ and $p \neq$ sender then
2: send m to all
3: halt
4: receive round k messages
5: if received m then
6: deliver(m)
7: if $k = f+1$ then halt
8: else if $k = f+1$
9: deliver(SF)
10: halt

Terminates in $f+1$ rounds

How can we do better?
find a protocol whose round complexity is proportional to $t$ – the number of failures that actually occurred – rather than to $f$ – the max number of failures that may occur
Early stopping: the idea

Suppose processes can detect the set of processes that have failed by the end of round $i$.

Call that set $\text{faulty}(p, i)$.

If $|\text{faulty}(p, i)| < i$ there can be no active dangerous chains, and $p$ can safely deliver SF.
Early Stopping: The Protocol

Let \( \text{faulty}(p, k) \) be the set of processes that have failed to send a message to \( p \) in any round \( 1, \ldots, k \).

1: if \( p = \text{sender} \) then \( \text{value} := m \) else \( \text{value} := ? \)

Process \( p \) in round \( k, 1 \leq k \leq f + 1 \)

2: \( \text{send value to all} \)
3: if \( \text{value} \neq ? \) and delivered \( m \) in round \( k - 1 \) then halt
4: \( \text{receive round } k \text{ values from all} \)
5: \( \text{faulty}(p, k) := \text{faulty}(p, k - 1) \cup \{ q \mid p \text{ received no value from } q \text{ in round } k \} \)
6: if received value \( v \neq ? \) then
7: \( \text{value} := v \)
8: \( \text{deliver value} \)
9: else if \( k = f + 1 \) or \( |\text{faulty}(p, k)| < k \) then
10: \( \text{value} := \text{SF} \)
11: \( \text{deliver value} \)
12: if \( k = f + 1 \) then halt
Termination

Let faulty(p, k) be the set of processes that have failed to send a message to p in any round 1, . . . , k.

1: if p = sender then value := m else value := ?

Process p in round k, 1 ≤ k ≤ f + 1

2: send value to all
3: if value ≠ ? and delivered m in round k − 1 then halt
4: receive round k values from all
5: faulty(p, k) := faulty(p, k − 1) ∪ {q | p
received no value from q in round k}
6: if received value v ≠ ? then
7: value := v
8: deliver value
9: else if k = f + 1 or |faulty(p, k)| < k then
10: value := SF
11: deliver value
12: if k = f + 1 then halt
Termination

Let $\text{faulty}(p, k)$ be the set of processes that have failed to send a message to $p$ in any round $1, \ldots, k$.

1. if $p = \text{sender}$ then value := $m$ else value := ?

Process $p$ in round $k$, $1 \leq k \leq f + 1$

2. send value to all
3. if value $\neq ?$ and delivered $m$ in round $k - 1$ then halt
4. receive round $k$ values from all
5. $\text{faulty}(p, k) := \text{faulty}(p, k - 1) \cup \{q \mid p\}
\text{received no value from } q \text{ in round } k\}$
6. if received value $v \neq ?$ then
   7. value := $v$
   8. deliver value
9. else if $k = f + 1$ or $|\text{faulty}(p, k)| < k$ then
   10. value := SF
   11. deliver value
   12. if $k = f + 1$ then halt

- If in any round a process receives a value, then it delivers the value in that round.
- If a process has received only “?” for $f + 1$ rounds, then it delivers SF in round $f + 1$. 
Validity

Let $\text{faulty}(p, k)$ be the set of processes that have failed to send a message to $p$ in any round $1, \ldots, k$.

1: \hspace{1cm} if $p = \text{sender}$ then value := $m$ else value := $?$

Process $p$ in round $k, 1 \leq k \leq f+1$

2: \hspace{1cm} send value to all
3: \hspace{1cm} if value $\neq ?$ and delivered $m$ in round $k-1$ then halt
4: \hspace{1cm} receive round $k$ values from all
5: \hspace{1cm} $\text{faulty}(p, k) := \text{faulty}(p, k - 1) \cup \{q \mid p$ received no value from $q$ in round $k\}$
6: \hspace{1cm} if received value $v \neq ?$ then
7: \hspace{1cm} value := $v$
8: \hspace{1cm} deliver value
9: \hspace{1cm} else if $k = f+1$ or $|\text{faulty}(p, k)| < k$ then
10: \hspace{1cm} value := $\text{SF}$
11: \hspace{1cm} deliver value
12: \hspace{1cm} if $k = f+1$ then halt
Validity

Let $\text{faulty}(p, k)$ be the set of processes that have failed to send a message to $p$ in any round $1, \ldots, k$.

1: if $p = \text{sender}$ then $\text{value} := m$ else $\text{value} := ?$

Process $p$ in round $k, 1 \leq k \leq f + 1$

2: send $\text{value}$ to all
3: if $\text{value} \neq ?$ and delivered $m$ in round $k - 1$ then halt
4: receive round $k$ values from all
5: $\text{faulty}(p, k) := \text{faulty}(p, k - 1) \cup \{q \mid p$ received no value from $q$ in round $k\}$
6: if received value $v \neq ?$ then
   7: $\text{value} := v$
   8: deliver value
9: else if $k = f + 1$ or $|\text{faulty}(p, k)| < k$ then
   10: $\text{value} := \text{SF}$
   11: deliver value
12: if $k = f + 1$ then halt

- If the sender is correct then it sends $m$ to all in round 1
- By Validity of the underlying send and receive, every correct process will receive $m$ by the end of round 1
- By the protocol, every correct process will deliver $m$ by the end of round 1
Lemma 1

For any \( r \geq 1 \), if a process \( p \) delivers \( m \neq SF \) in round \( r \), then there exists a sequence of processes \( p_0, p_1, \ldots, p_r \) such that \( p_0 = \text{sender}, p_r = p \), and in each round \( k, 1 \leq k \leq r \), \( p_{k-1} \) sent \( m \) and \( p_k \) received it. Furthermore, all processes in the sequence are distinct, unless \( r = 1 \) and \( p_0 = p_1 = \text{sender} \).

Lemma 2:

For any \( r \geq 1 \), if a process \( p \) sets value \( m \) to \( SF \) in round \( r \), then there exist some \( j \leq r \) and a sequence of distinct processes \( q_j, q_{j+1}, \ldots, q_r = p \) such that \( q_j \) only receives “?” in rounds 1 to \( j \), \( |\text{faulty}(q_j, j)| < j \), and in each round \( k, j+1 \leq k \leq r \), \( q_{k-1} \) sends \( SF \) to \( q_k \) and \( q_k \) receives \( SF \).
Lemma 3:
It is impossible for $p$ and $q$, not necessarily correct or distinct, to set value in the same round $r$ to $m$ and SF, respectively.
Let \( \text{faulty}(p, k) \) be the set of processes that have failed to send a message to \( p \) in any round \( 1, \ldots, k \).

1. if \( p = \text{sender} \) then \( \text{value} := m \) else \( \text{value} := ? \)

Process \( p \) in round \( k, 1 \leq k \leq f + 1 \)

2. send \( \text{value} \) to all
3. if \( \text{value} \neq ? \) and delivered \( m \) in round \( k - 1 \) then halt
4. receive round \( k \) values from all
5. \( \text{faulty}(p, k) := \text{faulty}(p, k - 1) \cup \{ q \mid p \text{ received no value from } q \text{ in round } k \} \)
6. if received value \( v \neq ? \) then
7. \( \text{value} := v \)
8. deliver \( v \)
9. else if \( k = f + 1 \) or \( |\text{faulty}(p, k)| < k \) then
10. \( \text{value} := \text{SF} \)
11. deliver \( \text{value} \)
12. if \( k = f + 1 \) then halt

**Lemma 3:**
It is impossible for \( p \) and \( q \), not necessarily correct or distinct, to set value in the same round \( r \) to \( m \) and \( \text{SF} \), respectively.

**Proof**

By contradiction

Suppose \( p \) sets value = \( m \) and \( q \) sets value = \( \text{SF} \)

By Lemmas 1 and 2 there exist \( p_0, \ldots, p_r \)

\( q_j, \ldots, q_r \)

with the appropriate characteristics

Since \( q_j \) did not receive \( m \) from process \( p_{k - 1} \) \( 1 \leq k \leq j \) in round \( k \)

\( q_j \) must conclude that \( p_0, \ldots, p_{j - 1} \) are all faulty processes

But then, \( |\text{faulty}(q_j, j)| \geq j \)

**CONTRADICTION**
Let $\text{faulty}(p, k)$ be the set of processes that have failed to send a message to $p$ in any round $1, \ldots, k$

1: if $p = \text{sender}$ then value := $m$ else value := ?

Process $p$ in round $k, 1 \leq k \leq f + 1$

2: send value to all
3: if value $\neq ?$ and delivered $m$ in round $k - 1$ then halt
4: receive round $k$ values from all
5: $\text{faulty}(p, k) := \text{faulty}(p, k - 1) \cup \{ q \mid p$ received no value from $q$ in round $k \}$
6: if received value $v \neq ?$ then
7: value := $v$
8: deliver value
9: else if $k = f + 1$ or $|\text{faulty}(p, k)| < k$ then
10: value := SF
11: deliver value
12: if $k = f + 1$ then halt
Let $\text{faulty}(p, k)$ be the set of processes that have failed to send a message to $p$ in any round $1, \ldots, k$.

1: if $p = \text{sender}$ then $\text{value} := m$ else $\text{value} := ?$

Process $p$ in round $k, 1 \leq k \leq f + 1$

2: send value to all
3: if value $\neq ?$ and delivered $m$ in round $k - 1$ then halt
4: receive round $k$ values from all
5: $\text{faulty}(p, k) := \text{faulty}(p, k - 1) \cup \{q \mid p$ received no value from $q$ in round $k\}$
6: if received value $v \neq ?$ then
7: value := $v$
8: deliver value
9: else if $k = f + 1$ or $|\text{faulty}(p, k)| < k$ then
10: value := SF
11: deliver value
12: if $k = f + 1$ then halt

**Proof**

If no correct process ever receives $m$, then every correct process delivers SF in round $f + 1$.

Let $r$ be the earliest round in which a correct process delivers value $\neq \text{SF}$

$r \leq f$

□ By Lemma 3, no (correct) process can set value differently in round $r$
□ In round $r + 1 \leq f + 1$, that correct process sends its value to all
□ Every correct process receives and delivers the value in round $r + 1 \leq f + 1$

$r = f + 1$

□ By Lemma 1, there exists a sequence $p_0, \ldots, p_{f+1}$

$= p_r$ of distinct processes
□ Consider processes $p_0, \ldots, p_f$

$f + 1$ processes; only $f$ faulty
one of $p_0, \ldots, p_f$ is correct—let it be $p_c$
To send $v$ in round $c + 1$, $p_c$ must have set its value to $v$ and delivered $v$ in round $c < r$

CONTRADICTION
Let $\text{faulty}(p, k)$ be the set of processes that have failed to send a message to $p$ in any round $1, \ldots, k$.

1: if $p = \text{sender}$ then value := $m$ else value := ?

Process $p$ in round $k, 1 \leq k \leq f+1$

2: send value to all
3: if value $\neq ?$ and delivered $m$ in round $k-1$ then halt
4: receive round $k$ values from all
5: $\text{faulty}(p, k) := \text{faulty}(p, k-1) \cup \{q \mid p$ received no value from $q$ in round $k\}$
6: if received value $v \neq ?$ then
7: value := $v$
8: deliver value
9: else if $k = f+1$ or $|\text{faulty}(p, k)| < k$ then
10: value := SF
11: deliver value
12: if $k = f+1$ then halt
Integrity

Let \( \text{faulty}(p, k) \) be the set of processes that have failed to send a message to \( p \) in any round \( 1, \ldots, k \).

1. if \( p = \text{sender} \) then value := \( m \) else value := ?

Process \( p \) in round \( k, 1 \leq k \leq f + 1 \)

2. send value to all
3. if value \# ? and delivered \( m \) in round \( k - 1 \) then halt
4. receive round \( k \) values from all
5. \( \text{faulty}(p, k) := \text{faulty}(p, k - 1) \cup \{q \mid p \) received no value from \( q \) in round \( k \}\}
6. if received value \( v \# ? \) then
7. value := \( v \)
8. deliver value
9. else if \( k = f + 1 \) or \( |\text{faulty}(p, k)| < k \) then
10. value := \( \text{SF} \)
11. deliver value
12. if \( k = f + 1 \) then halt

- At most one \( m \)
- Failures are benign, and a process executes at most one deliver event before halting
- If \( m \neq \text{SF} \), only if \( m \) was broadcast
- From Lemma 1 in the proof of Agreement
A Lower Bound

Theorem

There is no algorithm that solves the consensus problem in fewer than $f+1$ rounds in the presence of $f$ crash failures, if $n \geq f + 2$

We consider a special case ($f = 1$) to study the proof technique
Views

Let $\alpha$ be an execution. The view of process $p_i$ in $\alpha$, denoted by $\alpha|p_i$, is the subsequence of computation and message receive events that occur in $p_i$ together with the state of $p_i$ in the initial configuration of $\alpha$. 
Let $\alpha$ be an execution. The view of process $p_i$ in $\alpha$, denoted by $\alpha|p_i$, is the subsequence of computation and message receive events that occur in $p_i$ together with the state of $p_i$ in the initial configuration of $\alpha$. 

![Diagram showing processes $p_1$, $p_2$, $p_3$, $p_4$ and their interactions.]}
**Definition** Let $\alpha_1$ and $\alpha_2$ be two executions of consensus and let $p_i$ be a correct process in both $\alpha_1$ and $\alpha_2$.

$\alpha_1$ is similar to $\alpha_2$ with respect to $p_i$, denoted $\alpha_1 \sim_{p_i} \alpha_2$ if

$$\alpha_1|p_i = \alpha_2|p_i$$
Similarity

**Definition** Let $\alpha_1$ and $\alpha_2$ be two executions of consensus and let $p_i$ be a correct process in both $\alpha_1$ and $\alpha_2$.

$\alpha_1$ is similar to $\alpha_2$ with respect to $p_i$, denoted $\alpha_1 \sim_{p_i} \alpha_2$, if

$$\alpha_1|p_i = \alpha_2|p_i$$

**Note** If $\alpha_1 \sim_{p_i} \alpha_2$ then $p_i$ decides the same value in both executions.
Similarity

**Definition** Let \( \alpha_1 \) and \( \alpha_2 \) be two executions of consensus and let \( p_i \) be a correct process in both \( \alpha_1 \) and \( \alpha_2 \).

\( \alpha_1 \) is **similar** to \( \alpha_2 \) with respect to \( p_i \), denoted \( \alpha_1 \sim_{p_i} \alpha_2 \) if

\[
\alpha_1|_{p_i} = \alpha_2|_{p_i}
\]

**Note** If \( \alpha_1 \sim_{p_i} \alpha_2 \) then \( p_i \) decides the same value in both executions.

**Lemma** If \( \alpha_1 \sim_{p_i} \alpha_2 \) and \( p_i \) is correct, then \( \text{dec}(\alpha_1) = \text{dec}(\alpha_2) \)
Definition Let $\alpha_1$ and $\alpha_2$ be two executions of consensus and let $p_i$ be a correct process in both $\alpha_1$ and $\alpha_2$.

$\alpha_1$ is similar to $\alpha_2$ with respect to $p_i$, denoted $\alpha_1 \sim_{p_i} \alpha_2$ if

$$\alpha_1|p_i = \alpha_2|p_i$$

Note If $\alpha_1 \sim_{p_i} \alpha_2$ then $p_i$ decides the same value in both executions

Lemma If $\alpha_1 \sim_{p_i} \alpha_2$ and $p_i$ is correct, then $\text{dec}(\alpha_1) = \text{dec}(\alpha_2)$

The transitive closure of $\alpha_1 \sim_{p_i} \alpha_2$ is denoted $\alpha_1 \approx_{p_i} \alpha_2$.

We say that $\alpha_1 \approx \alpha_2$ if there exist executions $\beta_1, \beta_2, \ldots, \beta_{k+1}$ such that

$$\alpha_1 = \beta_1 \sim_{p_{i_1}} \beta_2 \sim_{p_{i_2}} \ldots, \sim_{p_{i_k}} \beta_{k+1} = \alpha_2$$
**Definition** Let $\alpha_1$ and $\alpha_2$ be two executions of consensus and let $p_i$ be a correct process in both $\alpha_1$ and $\alpha_2$. $\alpha_1$ is similar to $\alpha_2$ with respect to $p_i$, denoted $\alpha_1 \sim_{p_i} \alpha_2$ if

$$\alpha_1|p_i = \alpha_2|p_i$$

**Note** If $\alpha_1 \sim_{p_i} \alpha_2$ then $p_i$ decides the same value in both executions.

The transitive closure of $\alpha_1 \sim_{p_i} \alpha_2$ is denoted $\alpha_1 \approx \alpha_2$.

We say that $\alpha_1 \approx \alpha_2$ if there exist executions $\beta_1, \beta_2, \ldots, \beta_{k+1}$ such that

$$\alpha_1 = \beta_1 \sim_{p_{i_1}} \beta_2 \sim_{p_{i_2}} \ldots, \sim_{p_{i_k}} \beta_{k+1} = \alpha_2$$

**Lemma** If $\alpha_1 \approx \alpha_2$ then $\text{dec}(\alpha_1) = \text{dec}(\alpha_2)$

**Lemma** If $\alpha_1 \sim_{p_i} \alpha_2$ and $p_i$ is correct, then $\text{dec}(\alpha_1) = \text{dec}(\alpha_2)$
Single-Failure Case

There is no algorithm that solves consensus in fewer than two rounds in the presence of one crash failure, if $n \geq 3$
By contradiction

Consider a one-round execution in which each process proposes 0. What is the decision value?

Consider another one-round execution in which each process proposes 1. What is the decision value?

Show that there is a chain of similar executions that relate the two executions.

So what?
Definition

$\alpha^i$ is the execution of the algorithm in which

- no failures occur
- only processes $p_0, \ldots, p_{i-1}$ propose 1
\( \alpha^i \) is the execution of the algorithm in which
- no failures occur
- only processes \( p_0, \ldots, p_{i-1} \) propose 1
\[ \alpha_i \]

**Definition**

\[ \alpha_i \] is the execution of the algorithm in which

- no failures occur
- only processes \( p_0, \ldots, p_{i-1} \) propose 1
\[ \alpha^i \]

**Definition**

\( \alpha^i \) is the execution of the algorithm in which

- no failures occur
- only processes \( p_0, \ldots, p_{i-1} \) propose 1
Definition

\( \alpha^i \) is the execution of the algorithm in which

- no failures occur
- only processes \( p_0, \ldots, p_{i-1} \) propose 1
Definition

\( \alpha^i \) is the execution of the algorithm in which
- no failures occur
- only processes \( p_0, \ldots, p_{i-1} \) propose 1
**Definition**

$\alpha^i$ is the execution of the algorithm in which:

- no failures occur
- only processes $p_0, \ldots, p_{i-1}$ propose 1
Definition

$\alpha^i$ is the execution of the algorithm in which:
- no failures occur
- only processes $p_0, \ldots, p_{i-1}$ propose 1

Diagram:

- $\alpha^0$ with nodes $p_0, p_1, \ldots, p_{n-1}$
- $\alpha^i$ with nodes $p_0, p_1, \ldots, p_{n-1}$
\( \alpha_i \)'s

**Definition**

\( \alpha_i \) is the execution of the algorithm in which
- no failures occur
- only processes \( p_0, \ldots, p_{i-1} \) propose 1
**Definition**

$\alpha^i$ is the execution of the algorithm in which

- no failures occur
- only processes $p_0, \ldots, p_{i-1}$ propose 1
Adjacent $\alpha^i$'s are similar!

Starting from $\alpha^i$, we build a set of executions $\alpha^i_j$ where $0 \leq j \leq n-1$ as follows:

$\alpha^i_j$ is obtained from $\alpha^i$ after removing the messages that $p_i$ sends to the $j$-th highest numbered processors (excluding itself)
The executions
The executions

$p_0 \quad 1$

$p_{i-1} \quad 1$

$p_i \quad 0$

$p_{i+1} \quad 0$

$p_{n-1} \quad 0$

$\alpha_0^i$
The executions
The executions

\[
\begin{align*}
\alpha_0 & = p_0 \\
\alpha_1 & = p_0 \\
\end{align*}
\]
The executions

\[ p_0 \quad 1 \quad p_{i-1} \quad 1 \quad p_i \quad 0 \quad p_{i+1} \quad 0 \quad p_{n-1} \quad 0 \]

\[ \alpha^i \quad \alpha^i_0 \quad \alpha^i_1 \]
The executions
The executions

\[ p_0 \quad 1 \]
\[ p_{i-1} \quad 1 \]
\[ p_i \quad 0 \]
\[ p_{i+1} \quad 0 \]
\[ p_{n-1} \quad 0 \]
\[ \alpha_0^i \]

\[ p_0 \quad 1 \]
\[ p_{i-1} \quad 1 \]
\[ p_i \quad 0 \]
\[ p_{i+1} \quad 0 \]
\[ p_{n-1} \quad 0 \]
\[ \alpha_1^i \]

...
The executions

\[ p_0 \quad 1 \]
\[ p_{i-1} \quad 1 \]
\[ p_i \quad 0 \]
\[ p_{i+1} \quad 0 \]
\[ p_{n-1} \quad 0 \]

\[ \alpha_0 \]

\[ p_0 \quad 1 \]
\[ p_{i-1} \quad 1 \]
\[ p_i \quad 0 \]
\[ p_{i+1} \quad 0 \]
\[ p_{n-1} \quad 0 \]

\[ \alpha_1 \]

\[ \ldots \]

\[ p_0 \quad 1 \]
\[ p_{i-1} \quad 1 \]
\[ p_i \quad 0 \]
\[ p_{i+1} \quad 0 \]
\[ p_{n-1} \quad 0 \]

\[ \alpha_{n-1} \]
The executions

\[ p_0 \quad 1 \]
\[ p_{i-1} \quad 1 \]
\[ p_i \quad 0 \]
\[ p_{i+1} \quad 0 \]
\[ p_{n-1} \quad 0 \]

\( \alpha^i_0 \)

\[ p_0 \quad 1 \]
\[ p_{i-1} \quad 1 \]
\[ p_i \quad 0 \]
\[ p_{i+1} \quad 0 \]
\[ p_{n-1} \quad 0 \]

\( \alpha^i_1 \)

\[ p_0 \quad 1 \]
\[ p_{i-1} \quad 1 \]
\[ p_i \quad 0 \]
\[ p_{i+1} \quad 0 \]
\[ p_{n-1} \quad 0 \]

\( \alpha^i_{n-1} \)

\[ \ldots \]
Indistinguishability

\[ p_0 \quad 1 \]
\[ p_{i-1} \quad 1 \]
\[ p_i \quad 0 \]
\[ p_{i+1} \quad 0 \]
\[ p_{n-1} \quad 0 \]

\[ \alpha_i \]
\[ \alpha_0 \]
Indistinguishability

\[ p_i^{i-1} + p_i^{i+1} = p_i \]

\[ p_0^{i-1} + p_i^{i-1} = 1 \]

\[ p_i^{i+1} + p_i = 0 \]

\[ p_{n-1}^{i+1} + p_{n-1} = 0 \]
Indistinguishability

\[ p_0, p_{i-1}, p_i, p_{i+1}, p_{n-1} \]

\[ \alpha_i, \alpha_{i+1} \]

\[ \approx \]

\[ \approx \]
Indistinguishability
Indistinguishability

\[
\begin{align*}
\alpha_i^1 & \quad \alpha_i^2
\end{align*}
\]

\[
\begin{align*}
p_0 & \quad 1 \\
p_{i-1} & \quad 1 \\
p_i & \quad 0 \\
p_i+1 & \quad 0 \\
p_{n-1} & \quad 0
\end{align*}
\]
Indistinguishability

\[ p_0 \quad 1 \]
\[ p_{i-1} \quad 1 \]
\[ p_i \quad 0 \]
\[ p_{i+1} \quad 0 \]
\[ p_{n-1} \quad 0 \]

\[ \alpha^i \]
\[ \mathcal{N} \]
\[ \alpha^i_{n-1} \]
Indistinguishability

\[
p_0 \quad 1 \\
p_{i-1} \quad 1 \\
p_i \quad 0 \\
p_{i+1} \quad 0 \\
p_{n-1} \quad 0
\]

\[
\alpha^i \\
\lambda
\]

\[\alpha^i_{n-1}\]
Indistinguishability

\[ p_0 \quad 1 \quad \alpha_i \quad \beta_i^{n-1} \]
\[ p_{i-1} \quad 1 \quad \alpha_i \quad \beta_i^{n-1} \]
\[ p_i \quad 0 \quad \lambda \quad \beta_i^{n-1} \]
\[ p_{i+1} \quad 0 \quad \alpha_i \quad \beta_i^{n-1} \]
\[ p_{n-1} \quad 0 \quad \alpha_i \quad \beta_i^{n-1} \]
Indistinguishability

\[ p_0 \quad 1 \]

\[ p_{i-1} \quad 1 \]
\[ p_i \quad 0 \]
\[ p_{i+1} \quad 0 \]
\[ p_{n-1} \quad 0 \]

\[ \alpha_i \]
\[ \mathcal{U} \]
\[ \alpha_{n-1} \]

\[ \approx \]

\[ p_0 \quad 1 \]

\[ p_{i-1} \quad 1 \]
\[ p_i \quad 1 \]
\[ p_{i+1} \quad 0 \]
\[ p_{n-1} \quad 0 \]

\[ \beta_i \]
\[ \beta_{n-1} \]
Indistinguishability

\[ p_0 \quad 1 \]
\[ p_{i-1} \quad 1 \]
\[ p_i \quad 0 \]
\[ \alpha^i \]
\[ \beta^i \]
\[ p_{i+1} \quad 0 \]
\[ p_{n-1} \quad 0 \]

\[ p_0 \quad 1 \]
\[ p_{i-1} \quad 1 \]
\[ p_i \quad 1 \]
\[ \approx \]
\[ p_{i+1} \quad 0 \]
\[ p_{n-1} \quad 0 \]
\[ \alpha^i_{n-1} \approx \beta^i_{n-2} \]
Indistinguishability

$$\alpha^i \approx \beta^i_{n-3}$$
Indistinguishability
Indistinguishability

\[ p_i = \begin{cases} 1 & \text{if } i = 0 \\ 0 & \text{otherwise} \end{cases} \]

\[ p_{i+1} = \begin{cases} 1 & \text{if } i = n-1 \\ 0 & \text{otherwise} \end{cases} \]

\[ \alpha_i \approx \beta_0 \]
Indistinguishability

\[ p_0 \approx \alpha_i \approx p_0 \]

\[ p_{i-1} \quad 1 \]
\[ p_i \quad 0 \]
\[ p_{i+1} \quad 0 \]

\[ p_{n-1} \quad 0 \]

\[ \alpha_i \]

\[ \approx \]

\[ \alpha_{i+1} \]
Arbitrary failures with message authentication

- Fail-stop
- Crash
- Send Omission
- Receive Omission
- General Omission
- Arbitrary failures with message authentication
- Arbitrary (Byzantine) failures

- Process can send conflicting messages to different receivers
- Messages are signed with unforgeable signatures
Valid messages

A valid message $m$ has the following form:

in round 1:

$m : s_{id}$ (m is signed by the sender)

in round $r > 1$, if received by $p$ from $q$:

$m : p_1 : p_2 : \ldots : p_r$ where

- $p_1 = \text{sender}$; $p_r = q$
- $p_1, \ldots, p_r$ are distinct from each other and from $p$
- message has not been tampered with
AFMA: The Idea

- A correct process $p$ discards all non-valid messages it receives.
- If a message is valid,
  - it "extracts" the value from the message
  - it relays the message, with its own signature appended.
- At round $f+1$:
  - if it extracted exactly one message, $p$ delivers it
  - otherwise, $p$ delivers SF
AFMA: The Protocol

Initialization for process $p$:
if $p$ = sender and $p$ wishes to broadcast $m$ then
    extracted := relay := \{m\}

Process $p$ in round $k$, $1 \leq k \leq f+1$
for each $s \in$ relay
    send $s : p$ to all
receive round $k$ messages from all processes
relay := $\emptyset$
for each valid message received $s = m : p_1 : p_2 : \ldots : p_k$
    if $m \notin$ extracted then
        extracted := extracted $\cup \{m\}$
        relay := relay $\cup \{s\}$

At the end of round $f+1$
if $\exists m$ such that extracted = $\{m\}$ then
    deliver $m$
else deliver SF
Termination

Initialization for process $p$:

if $p = \text{sender}$ and $p$ wishes to broadcast $m$ then
  extracted := relay := \{m\}

Process $p$ in round $k$, $1 \leq k \leq f+1$

for each $s \in \text{relay}$
  send $s : p$ to all
receive round $k$ messages from all processes
relay := \{
for each valid message received $s = m : p_1 : p_2 : \ldots : p_k$
if $m \notin \text{extracted}$ then
  extracted := extracted $\cup \{m\}$
  relay := relay $\cup \{s\}$

At the end of round $f+1$
if $\exists m$ such that extracted $= \{m\}$ then
  deliver $m$
else deliver SF

In round $f+1$, every correct process delivers either $m$ or SF and then halts
Agreement

Initialization for process $p$:
   if $p$ = sender and $p$ wishes to broadcast $m$ then
      extracted := relay := \{m\}

Process $p$ in round $k$, $1 \leq k \leq f+1$
   for each $s \in$ relay
      send $s : p$ to all
   receive round $k$ messages from all processes
   relay := ()
   for each valid message received $s = m : p_1 : p_2 : \ldots : p_k$
      if $m \notin$ extracted then
         extracted := extracted $\cup \{m\}$
         relay := relay $\cup \{s\}$

At the end of round $f+1$
   if $\exists m$ such that extracted $= \{m\}$ then
      deliver $m$
   else deliver SF

Lemma. If a correct process extracts $m$, then every correct process eventually extracts $m$

Proof
Let $r$ be the earliest round in which some correct process extracts $m$. Let that process be $p$.
   • if $p$ is the sender, then in round 1 $p$ sends a valid message to all.
      All correct processes extract that message in round 1
   • otherwise, $p$ has received in round $r$ a message
      $m : p_1 : p_2 : \ldots : p_r$
   • Claim: $p_1, p_2, \ldots, p_r$ are all faulty
      - true for $p_1 = s$
      - Suppose $p_j, 1 \leq j \leq r$, were correct
      - $p_j$ signed and relayed message in round $j$
      - $p_j$ extracted message in round $j - 1$
         CONTRADICION
   • If $r \leq f$, $p$ will send a valid message
      $m : p_1 : p_2 : \ldots : p_r : p$
      in round $r + 1 \leq f + 1$ and every correct process will extract it in round $r + 1 \leq f + 1$
   • If $r = f + 1$, by Claim above, $p_1, p_2, \ldots, p_{f+1}$ faulty
      - At most $f$ faulty processes
      - CONTRADICTION
Validity

Initialization for process $p$:
  if $p$ = sender and $p$ wishes to broadcast $m$ then
    extracted := relay := \{m\}

Process $p$ in round $k$, $1 \leq k \leq f+1$
  for each $s \in$ relay
    send $s : p$ to all
  receive round $k$ messages from all processes
  relay := \{}
  for each valid message received $s = m : p_1 : p_2 : \ldots : p_k$
    if $m \not\in$ extracted then
      extracted := extracted $\cup \{m\}$
      relay := relay $\cup \{s\}$

At the end of round $f+1$
  if $\exists m$ such that extracted = \{m\} then
    deliver $m$
  else deliver SF
Validity

Initialization for process \( p \):
- if \( p = \) sender and \( p \) wishes to broadcast \( m \) then
  - extracted := relay := \{\( m \}\}

Process \( p \) in round \( k, 1 \leq k \leq f+1 \)
- for each \( s \in \text{relay} \)
  - send \( s : p \) to all
- receive round \( k \) messages from all processes
  - relay := \( \{\} \)
- for each valid message received \( s = m : p_1 : p_2 : \ldots : p_k \)
  - if \( m \notin \text{extracted} \) then
    - extracted := extracted \( \cup \{m\} \)
    - relay := relay \( \cup \{s\} \)

At the end of round \( f+1 \)
- if \( \exists m \) such that \( \text{extracted} = \{m\} \) then
  - deliver \( m \)
- else deliver SF

From Agreement and the observation that the sender, if correct, delivers its own message.
TRB for arbitrary failures

Fail-stop ←-→ Crash

Send Omission ←-→ Receive Omission

General Omission ←-→

Arbitrary failures with message authentication

Arbitrary (Byzantine) failures

Srikanth, T.K., Toueg S.
Simulating Authenticated Broadcasts to Derive Simple Fault-Tolerant Algorithms
Distributed Computing 2 (2), 80-94
AF: The Idea

- Identify the essential properties of message authentication that made AFMA work
- Implement these properties without using message authentication
AF: The Approach

- Introduce two primitives
  - \textit{broadcast}(p, m, i) (executed by \textit{p} in round \textit{i})
  - \textit{accept}(p, m, i) (executed by \textit{q} in round \textit{j} \geq \textit{i})
- Give axiomatic definitions of broadcast and accept
- Derive an algorithm that solves TRB for AF using these primitives
- Show an implementation of these primitives that does not use message authentication
Properties of broadcast and accept

**Correctness**  If a correct process $p$ executes broadcast $(p, m, i)$ in round $i$, then all correct processes will execute accept $(p, m, i)$ in round $i$.

**Unforgeability**  If a correct process $q$ executes accept $(p, m, i)$ in round $j \geq i$, and $p$ is correct, then $p$ did in fact execute broadcast $(p, m, i)$ in round $i$.

**Relay**  If a correct process $q$ executes accept $(p, m, i)$ in round $j \geq i$, then all correct processes will execute accept $(p, m, i)$ by round $j + 1$. 
**AF: The Protocol - 1**

sender $s$ in round 0:
0: extract $m$

sender $s$ in round 1:
1: broadcast($s, m, 1$) 
Process $p$ in round $k, 1 \leq k \leq f+1$
2: if $p$ extracted $m$ in round $k-1$ and $p \neq$ sender then
4: broadcast($p, m, k$)
5: if $p$ has executed at least $k$ accept($q_i, m, j_i$) $1 \leq i \leq k$ in rounds 1 through $k$
   (where (i) $q_i$ distinct from each other and from $p$, (ii) one $q_i$ is $s$, and
   (iii) $1 \leq j_i \leq k$) and $p$ has not previously extracted $m$ then
6: extract $m$
7: if $k = f+1$ then
8: if in the entire execution $p$ has extracted exactly one $m$ then
9: deliver $m$
10: else deliver SF
11: halt
Termination

sender $s$ in round 0:
0: extract $m$
sender $s$ in round 1:
1: broadcast($s, m, 1$)

Process $p$ in round $k$, $1 \leq k \leq f+1$
2: if $p$ extracted $m$ in round $k-1$ and $p \neq$ sender then
4: broadcast($p, m, k$)
5: if $p$ has executed at least $k$ accept($q_i, m, j_i$) $1 \leq i \leq k$ in rounds 1 through $k$
   (where (i) $q_i$ distinct from each other and from $p$, (ii) one $q_i$ is $s$, and (iii) $1 \leq j_i \leq k$
   and $p$ has not previously extracted $m$ then
6: extract $m$
7: if $k = f+1$ then
8: if in the entire execution $p$ has extracted exactly one $m$ then
9: deliver $m$
10: else deliver SF
11: halt

In round $f+1$, every correct process delivers either $m$ or SF and then halts
sender $s$ in round 0:
0: extract $m$

sender $s$ in round 1:
1: broadcast($s, m, 1$)

Process $p$ in round $k, 1 \leq k \leq f + 1$
2: if $p$ extracted $m$ in round $k - 1$ and $p \neq sender$ then
4: broadcast($p, m, k$)
5: if $p$ has executed at least $k$ accept($q_i, m, j_i$) $1 \leq i \leq k$ in rounds 1 through $k$
   (where (i) $q_i$ distinct from each other and from $p$, (ii) one $q_i$ is $s$, and (iii) $1 \leq j_i \leq k$)
   and $p$ has not previously extracted $m$ then
6: extract $m$
7: if $k = f + 1$ then
8: if in the entire execution $p$ has extracted exactly one $m$ then
9: deliver $m$
10: else deliver SF
11: halt

Lemma
If a correct process extracts $m$, then every correct process eventually extracts $m$
Agreement – 1

sender $s$ in round 0:
0: extract $m$

sender $s$ in round 1:
1: broadcast $(s, m, 1)$

Process $p$ in round $k$, $1 \leq k \leq f + 1$
2: if $p$ extracted $m$ in round $k - 1$ and $p \neq$ sender then
4: broadcast $(p, m, k)$
5: if $p$ has executed at least $k$ accept $(q_i, m, j_i)$, $1 \leq i \leq k$ in rounds 1 through $k$
     (where (i) $q_i$ distinct from each other and from $p$, (ii) one $q_i$ is $s$, and (iii) $1 \leq j_i \leq k$ )
     and $p$ has not previously extracted $m$ then
6: extract $m$
7: if $k = f + 1$ then
8: if in the entire execution $p$ has extracted exactly one $m$ then
9: deliver $m$
10: else deliver SF
11: halt

Lemma
If a correct process extracts $m$, then every correct process eventually extracts $m$
Agreement - 1

Proof

Let $r$ be the earliest round in which some correct process extracts $m$. Let that process be $p$.

\begin{itemize}
  \item if $r = 0$, then $p = s$ and $p$ will execute $\text{broadcast}(s,m,1)$ in round 1. By \text{CORRECTNESS}, all correct processes will execute $\text{accept}(s,m,1)$ in round 1 and extract $m$.
  \item if $r > 0$, the sender is faulty. Since $p$ has extracted $m$ in round $r$, $p$ has accepted at least $r$ triples with properties (i), (ii), and (iii) by round $r$
    \begin{itemize}
      \item $r \leq f$ By \text{RELAY}, all correct processes will have accepted those $r$ triples by round $r + 1$
      \item $p$ will execute $\text{broadcast}(p,m,r + 1)$ in round $r + 1$
      \item By \text{CORRECTNESS}, any correct process other than $p, q_1, q_2, \ldots, q_r$ will have accepted $r + 1$ triples ($q_k,m,j_k), 1 \leq j_k \leq r + 1$, by round $r + 1$
      \item $q_1, q_2, \ldots, q_r, p$ are all distinct
      \item every correct process other than $q_1, q_2, \ldots, q_r, p$ will extract $m$
      \item $p$ has already extracted $m$; what about $q_1, q_2, \ldots, q_r$?
    \end{itemize}
\end{itemize}

Lemma

If a correct process extracts $m$, then every correct process eventually extracts $m$.
Agreement - 2

sender $s$ in round 0:
0: extract $m$
sender $s$ in round 1:
1: broadcast($s, m, 1$)

Process $p$ in round $k, 1 \leq k \leq f+1$
2: if $p$ extracted $m$ in round $k-1$ and $p \neq$ sender then
4: broadcast($p, m, k$)
5: if $p$ has executed at least $k$ accept($q_i, m, j_i$), $1 \leq i \leq k$ in rounds 1 through $k$
   (where (i) $q_i$ distinct from each other and from $p$, (ii) one $q_i$ is $s$, and (iii) $1 \leq j_i \leq k$)
   and $p$ has not previously extracted $m$ then
6: extract $m$
7: if $k = f+1$ then
8: if in the entire execution $p$ has extracted exactly one $m$ then
9: deliver $m$
10: else deliver SF
11: halt

Claim: $q_1, q_2, \ldots, q_r$ are all faulty

> Suppose $q_k$ were correct
> $p$ has accepted($q_k, m, j_k$) in round $j_k \leq r$
> By UNFORGEABILITY, $q_k$ executed broadcast($q_k, m, j_k$) in round $j_k$
> $q_k$ extracted $m$ in round $j_{k-1} < r$

CONTRADICTION

$\Box$ Case 2: $r = f+1$

$\Box$ Since there are at most $f$ faulty processes, some process $q_l$ in $q_1, q_2, \ldots, q_{f+1}$ is correct

$\Box$ By UNFORGEABILITY, $q_l$ executed broadcast($q_l, m, j_l$) in round $j_l \leq r$
$\Box$ $q_l$ has extracted $m$ in round $j_{l-1} < f + 1$

CONTRADICTION
Validity

A correct sender executes broadcast\((s, m, 1)\) in round 1.

By CORRECTNESS, all correct processes execute accept\((s, m, 1)\) in round 1 and extract \(m\).

In order to extract a different message \(m'\), a process must execute accept\((s, m', 1)\) in some round \(i \leq f + 1\).

By UNFORGEABILITY, and because \(s\) is correct, no correct process can extract \(m' \neq m\).

All correct processes will deliver \(m\).
Implementing broadcast and accept

- A process that wants to broadcast $m$, does so through a series of witnesses
  - Sends $m$ to all
  - Each correct process becomes a witness by relaying $m$ to all
- If a process receives enough witness confirmations, it accepts $m$
Can we rely on witnesses?

- Only if not too many faulty processes!
- Otherwise, a set of faulty processes could fool a correct process by acting as witnesses of a message that was never broadcast
- How large can be $f$ with respect to $n$?
Byzantine Generals

One General G, a set of Lieutenants $L_i$

General can order Attack (A) or Retreat (R)

General may be a traitor; so may be some of the Lieutenants

* * *

I. If G is trustworthy, every trustworthy $L_i$ must follow G’s orders

II. Every trustworthy $L_i$ must follow same battleplan
The plot thickens...

One traitor
The plot thickens...

One traitor

$G$

$L_1$

$L_2$
The plot thickens...

One traitor
The plot thickens...

One traitor
The plot thickens...

One traitor
The plot thickens...

One traitor
The plot thickens...

One traitor
The plot thickens...

One traitor
The plot thickens...

One traitor
The plot thickens...

One traitor
The plot thickens...

One traitor
The plot thickens...

One traitor
The plot thickens...

One traitor
The plot thickens...

One traitor
The plot thickens...

One traitor
The plot thickens...

One traitor
The plot thickens...

One traitor
The plot thickens...

One traitor
The plot thickens...

One traitor
A Lower Bound

Theorem

There is no algorithm that solves TRB for Byzantine failures if \( n \leq 3f \)

Back to the protocol...

To broadcast a message in round $r$, $p$ sends $(init, p, m, r)$ to all

A confirmation has the form $(echo, p, m, r)$

A witness sends $(echo, p, m, r)$ if either:
- it receives $(init, p, m, r)$ from $p$ directly
- or it receives confirmations for $(p, m, r)$ from at least $f + 1$ processes (at least one correct witness)

A process accepts $(p, m, r)$ if it has received $n - f$ confirmations (as many as possible...)

Protocol proceeds in rounds. Each round has 2 phases
Implementation of broadcast and accept

**Phase** \(2r - 1\)
1: \(p\) sends \((init, p, m, r)\) to all

**Phase** \(2r\)
2: if \(q\) received \((init, p, m, r)\) in phase \(2r - 1\) then
3: \(q\) sends \((echo, p, m, r)\) to all /* \(q\) becomes a witness */
4: if \(q\) receives \((echo, p, m, r)\) from at least \(n - f\) distinct processes in phase \(2r\) then
5: \(q\) accepts \((p, m, r)\)

**Phase** \(j > 2r\)
6: if \(q\) has received \((echo, p, m, r)\) from at least \(f + 1\) distinct processes in phases \((2r, 2r + 1, \ldots, j - 1)\) then
7: \(q\) sends \((echo, p, m, r)\) to all processes /* \(q\) becomes a witness */
8: if \(q\) has received \((echo, p, m, r)\) from at least \(n - f\) processes in phases \((2r, 2r + 1, \ldots, j)\) then
9: \(q\) accepts \((p, m, r)\)
Implementation of broadcast and accept

**Phase** $2r-1$

1: $p$ sends $(init, p, m, r)$ to all

**Phase** $2r$

2: if $q$ received $(init, p, m, r)$ in phase $2r-1$ then

3: $q$ sends $(echo, p, m, r)$ to all /* $q$ becomes a witness */

4: if $q$ receives $(echo, p, m, r)$ from at least $n-f$ distinct processes in phase $2r$ then

5: $q$ accepts $(p, m, r)$

**Phase** $j > 2r$

6: if $q$ has received $(echo, p, m, r)$ from at least $f+1$ distinct processes in phases $(2r, 2r+1, \ldots, j-1)$ then

7: $q$ sends $(echo, p, m, r)$ to all processes /* $q$ becomes a witness */

8: if $q$ has received $(echo, p, m, r)$ from at least $n-f$ processes in phases $(2r, 2r+1, \ldots, j)$ then

9: $q$ accepts $(p, m, r)$

Is termination a problem?
The implementation is correct

Theorem

If $n > 3f$, the given implementation of broadcast$(p, m, r)$ and accept$(p, m, r)$ satisfies Unforgeability, Correctness, and Relay

Assumption

Channels are authenticated
Correctness

If a correct process $p$ executes broadcast$(p, m, r)$ in round $r$, then all correct processes will execute accept$(p, m, r)$ in round $r$. 
Correctness

If a correct process \( p \) executes broadcast\((p, m, r)\) in round \( r \), then all correct processes will execute accept\((p, m, r)\) in round \( r \)

If \( p \) is correct then

- \( p \) sends \((init, p, m, r)\) to all in round \( r \) (phase \( 2r - 1 \))
- by Validity of the underlying send and receive, every correct process receives \((init, p, m, r)\) in phase
- every correct process becomes a witness
- every correct process sends \((echo, p, m, r)\) in phase \( 2r \)
- since there are at least \( n - f \) correct processes, every correct process receives at least \( n - f \) echoes in phase \( 2r \)
- every correct process executes accept\((p, m, r)\) in phase \( 2r \) (in round \( r \))
Unforgeability - 1

If a correct process \( q \) executes \( \text{accept}(p, m, r) \) in round \( j \geq r \), and \( p \) is correct, then \( p \) did in fact execute broadcast\((p, m, r)\) in round \( r \)

- Suppose \( q \) executes \( \text{accept}(p, m, r) \) in round \( j \)
- \( q \) received \((\text{echo}, p, m, r)\) from at least \( n - f \) distinct processes by phase \( k \), where \( k = 2j - 1 \) or \( k = 2j \)
- Let \( k' \) be the earliest phase in which some correct process \( q' \) becomes a witness to \((p, m, r)\)
Unforgeability - 1

If a correct process $q$ executes $\text{accept}(p, m, r)$ in round $j \geq r$, and $p$ is correct, then $p$ did in fact execute $\text{broadcast}(p, m, r)$ in round $r$.

- Suppose $q$ executes $\text{accept}(p, m, r)$ in round $j$.
- $q$ received $(\text{echo}, p, m, r)$ from at least $n - f$ distinct processes by phase $k$, where $k = 2j - 1$ or $k = 2j$.
- Let $k'$ be the earliest phase in which some correct process $q'$ becomes a witness to $(p, m, r)$.

Case 1: $k' = 2r - 1$

- $q'$ received $(\text{init}, p, m, r)$ from $p$.
- Since $p$ is correct, it follows that $p$ did execute $\text{broadcast}(p, m, r)$ in round $r$.

Case 2: $k' > 2r - 1$

- $q'$ has become a witness by receiving $(\text{echo}, p, m, r)$ from $f + 1$ distinct processes.
- At most $f$ are faulty; one is correct.
- This process was a witness to $(p, m, r)$ before phase $k'$.

CONTRADICTION

The first correct process receives $(\text{init}, p, m, r)$ from $p$!
Unforgeability -2

For $q$ to accept, some correct process must become witness.

Earliest correct witness $q'$ becomes so in phase $2r - 1$, and only if $p$ did indeed executed broadcast $(p, m, r)$.

Any correct process that becomes a witness later can only do so if a correct process is already a witness.

For any correct process to become a witness, $p$ must have executed broadcast $(p, m, r)$.
If a correct process $q$ executes $\text{accept}(p, m, r)$ in round $j \geq r$, then all correct processes will execute $\text{accept}(p, m, r)$ by round $j + 1$. 
Suppose correct q executes $\text{accept}(p, m, r)$ in round $j$ (phase $k = 2j - 1$ or $k = 2j$).

$q$ received at least $n - f$ $(\text{echo}, p, m, r)$ from distinct processes by phase $k$.

At least $n - 2f$ of them are correct.

All correct procs received $(\text{echo}, p, m, r)$ from at least $n - 2f$ correct processes by phase $k$.

From $n > 3f$, it follows that $n - 2f \geq f + 1$.

Then, all correct processes become witnesses by phase $k$.

All correct processes send $(\text{echo}, p, m, r)$ by phase $k + 1$.

Since there are at least $n - f$ correct processes, all correct processes will accept $(p, m, r)$ by phase $k + 1$ (round $2j$ or $2j + 1$).

If a correct process $q$ executes $\text{accept}(p, m, r)$ in round $j \geq r$, then all correct processes will execute $\text{accept}(p, m, r)$ by round $j + 1$. 

Relay
Taking a step back...

- Specified Consensus and TRB
- In the synchronous model:
  - solved Consensus and TRB for General Omission failures
  - proved lower bound on rounds required by TRB
  - solved TRB for AFMA
  - proved lower bound on replication for solving TRB with AF
  - solved TRB with AF
Ordered Broadcasts for Benign Failures
FIFO Order

If a process broadcasts a message $m$ before it broadcasts a message $m'$, then no correct process delivers $m'$ unless it has previously delivered $m$.

Uniform FIFO Order

If a process broadcasts a message $m$ before it broadcasts a message $m'$, then no process (correct or faulty) delivers $m'$ unless it has previously delivered $m$.
Causal Order

If the broadcast of a message $m$ causally precedes the broadcast of a message $m'$, then no correct process delivers $m'$ unless it has previously delivered $m$.

Uniform Causal Order

If the broadcast of a message $m$ causally precedes the broadcast of a message $m'$, then no process (correct or faulty) delivers $m'$ unless it has previously delivered $m$. 
From FIFO to Causal

Local Order

If a process broadcasts a message m and a process delivers m before broadcasting m', then no correct process delivers m' unless it previously delivered m.

Causal Order = FIFO Order + Local Order
Total Order

If correct processes \( p \) and \( q \) both deliver messages \( m \) and \( m' \), then \( p \) delivers \( m \) before \( m' \) if and only if \( q \) delivers \( m \) before \( m' \).

Uniform Total Order

If correct or faulty processes \( p \) and \( q \) both deliver messages \( m \) and \( m' \), then \( p \) delivers \( m \) before \( m' \) if and only if \( q \) delivers \( m \) before \( m' \).
A Modular Approach to Broadcast Protocols

(Hadzilakos & Toueg)

Reliable Broadcast

FIFO Broadcast

FIFO Atomic Broadcast

Causal Broadcast

Total Order

Causal Atomic Broadcast

FIFO Order

Causal Order

FIFO Order

Causal Order

FIFO Order