Advanced Network Security

3. Agreement and consensus I: concepts and protocols for crash failures

Jaap-Henk Hoepman

Digital Security (DS)
Radboud University Nijmegen, the Netherlands
@xotoxot // ⚬ jhh@cs.ru.nl // ⚙ www.cs.ru.nl/~jhh
Byzantine generals
Types of faults

- **Stopping / Crash**
  - Process stops unexpectedly and does nothing after that, forever

- **Omission**
  - Process skips a step it is supposed to perform
    - *e.g. sending a messages; this models message dropping on an edge (except that there is a limit on the number of affected edges...)*

- **Byzantine**
  - Process performs arbitrary actions, not specified by the protocol
    - *e.g. sending different messages to different recipients*
Byzantine failures are real

 Receivers have slightly different thresholds, so may receive different values
Decision problems

- **Private inputs** $C[p]$. *in*, **private decision outputs** $C[q]$. *decision*

- **Termination condition**
  - Deterministic termination
    - *Every correct process decides irrevocably, and stops/ knows it decided*
  - Probabilistic termination (convergence)
    - *Every correct process decides irrevocably with probability 1, and stops/ knows it decided*
  - Implicit termination (stabilisation)
    - *Every correct process decides, but never knows it decided (and may change decisions in the process); no such changes occur after a finite number of steps*

- **Consistency condition**
  - A global predicate over inputs and decision outputs
  - Problem specific
Solving decision problems

- **We assume a certain topology** \( G = (V,E) \), \( n = |V| \)
  - Typically a clique

- **We assume certain faulty behaviour**
  - E.g. crash failures only

- **We assume at most** \( f < n \) **processes are faulty**
  - Link failures are modelled as process failures
  - \( f \) expresses **robustness**; typically \( f < n/3 \) or \( f < n/2 \)
  - Sometimes we specify certain processes can/cannot fail

- **We assume recipient knows sender of messages (authenticity)**
  - Not signatures, but because of point-to-point direct connections

\( f \) is an **assumption** on number of faults. Real number of faults in an execution may be lower or equal (in which case algorithm is successful) or not (in which case it fails).
Decision problem: replicated server

- Suppose two (replicated) servers \( p, q \) hold the same data (input)

- Consistency condition:
  - All correct processes decide on this input

- Termination condition:
  - Deterministic

- Assumptions
  - Crash failures
  - At most one of the replicated servers fail

- Protocol for \( p, q \)
  \[
  \text{forall } r \neq q, p \\
  \text{do send } C[p].in \text{ to } r \\
  C[p].decision = C[p].in
  \]

- Protocol for other processes \( r \)
  \[
  \text{receive } v \\
  C[r].decision = v
  \]

Also sometimes written as \( \text{decide}(C[p].in) \)
Decision problem: replicated server

- What if (replicated) servers \( p, q \) hold different data?

- What if both replicated servers fail?

Protocol for \( p, q \)

\[
\text{forall } r \neq q, p \quad \text{do send } C[p].in \text{ to } r \\
C[p].decision = C[p].in
\]

Protocol for other processes \( r \)

\[
\text{receive } v \\
C[r].decision = v
\]
Decision problem: weak broadcast

- **One server** $p$ holds a bit
  - Either 0 or 1

- **Consistency condition:**
  - All correct processes decide on the same value
  - If $p$ does not crash, this should be $p$'s input

- **Termination condition:**
  - Stabilising

- **Assumptions**
  - Crash failures

**Protocol for $p$**

\[
C[p].\text{decision} = C[p].\text{in}
\]
\[
\text{if } C[p].\text{in} == 1
\]
\[
\text{then for all } r \neq p
\]
\[
\text{do send 1 to } r
\]

**Protocol for other processes $r$**

\[
C[r].\text{decision} = 0
\]
\[
\text{receive 1}
\]
\[
C[r].\text{decision} = 1
\]
\[
\text{for all } q \neq r
\]
\[
\text{do send 1 to } q
\]
Decision problem: weak broadcast

- What if $p$ crashes?
- Why is this not deterministically terminating?

Protocol for $p$

```
C[p].decision = C[p].in
if C[p].in == 1
then forall r ≠ p
do send 1 to r
```

Protocol for other processes $r$

```
C[r].decision = 0
receive 1
C[r].decision = 1
forall q ≠ r
do send 1 to q
```
The consensus problem
The consensus problem

- All processes have a binary input value
  - So it is different from a broadcast

- Consistency condition
  - All correct processes decide on the same value (*Agreement*)
  - If all processors have the same input value $b$, then all correct processors must decide $b$ (*Validity*)

- Termination condition
  - Deterministic
Aside: solving consensus with broadcast

**Atomic broadcast**
- Sender $p$ holds a bit
  - Either 0 or 1
- Consistency condition:
  - All correct processes decide on the same value (even when sender $p$ fails)
  - If $p$ does not fail, all correct processes decide on sender $p$'s input
- Termination condition: deterministic

**Consensus protocol for $p$**
Initialise vector $V[]$
$V[p] = C[p].in$ broadcast $C[p].in$
forall $r \neq p$
do receive $V[r]$
$C[p].decision = Majority \{V[r]\}$

**In other words: atomic broadcast and consensus are very similar**

**Remember: no link failures**
Consensus for crash failures

- **Assume at most** \( f < n \) **crash failures**

- **Synchronous protocol**
  - Computation proceeds in rounds
  - At start of round \( r \), all processors send all messages for round \( r \)
  - Before proceeding to round \( r + 1 \) all processors receive all round \( r \) messages
    - *If they arrive, they arrive in this round; otherwise they are lost forever*
Consensus: main approach

- Each processor $p$ builds the following tree $T_p$

Level 0

Level 1

Level 2

Level $r$

Level $r + 1$

Level $f + 1$

$v_{q_1,q_2,\ldots,q_k}^p$ means: $q_k$ told $p$, that $q_{k-1}$ told $q_k$, .... that $q_1$'s value is $v$

Initially all $\bot$

$v_e^p = C[p].in$

$\forall j \notin \sigma$, i.e. $n - |\sigma| = n - r$ children
Building the tree: protocol for p

Before round 1
- Initialise tree. Set all $v^p_\sigma = \bot$ and $v^p_\epsilon = C[p].in$

Round $r, 1 \leq r \leq f + 1$
- For all $\sigma$ with $|\sigma| = r - 1 \land p \notin \sigma$, send $v^p_\sigma$ to all processors $q$ (including $p$)
  ★ Call this message $m^q_{\sigma;p}$
- Receive all $m^p_{\sigma;x}$ addressed to $p$ and store in $v^p_{\sigma;x}$
  ★ By the protocol $x \notin \sigma$ so $p$ receives $n - (r - 1)$ such messages from each $x$
The protocol in action: round 1

\[ v_1^p = v_\epsilon^1 \]
\[ v_q^p = \perp \]
\[ v_n^p = v_\epsilon^n \]

Processor q crashes

\[ v_{q_1,q_2,\ldots,q_k}^p \] means: \( q_k \) told \( p \), that \( q_{k-1} \) told \( q_k \), ...

that \( q_1 \)'s value is \( v \)

Initially all \( \perp \)

\[ v_\epsilon^p = C[p].\text{in} \]
The protocol in action: round 2

Initially all $\bot$

$v_{\epsilon}^p = C[p].in$

For crash failures we have either $v_{q,\sigma}^p = \bot$
or $v_{q,\sigma}^p = v_{\epsilon}^q$

$z$ tells $p$ that $q$ told $z$ its value is $v$; So $q$ crashed after sending to $z$

If $z$ is honest, $z$ will tell this to all honest nodes
Deciding on a value

- Let $V_p = \{ v | v = v^p_\sigma \in T_p \land v \neq \perp \}$, i.e. the set of different values in $T_p$
- What are the possible values for $V_p$?
  - If inputs are binary, then $\{\}$, $\{0\}$, $\{1\}$, $\{0,1\}$
- If $|V_p| = 1$, i.e. $V_p = \{v\}$
  - $p$ decides on $v$
- Otherwise
  - $p$ decides on a default value $v_{def}$, say 0
Correctness

Lemma: suppose both processors $p$ and $q$ are correct (i.e. don’t fail). Then if $v \in V_p$ then $v \in V_q$

Proof

- If $v \in V_p$ then $v = v_\sigma^p$ for some $\sigma$ with $p \not\in \sigma$
  
  - If $p \in \sigma$, i.e. $\sigma = \alpha; p; \beta$ then $p$ sent $v = m_{\sigma;p}^p$ and hence $v = v_\alpha^p$ too, with $p \not\in \alpha$

- If $|\sigma| < f + 1$ then $p$ will sent $m_{\sigma;p}^q = v_\sigma^p = v$ to $q$ and then $v_{\sigma;p}^q = v$ and so $v \in V_q$

- If $|\sigma| = f + 1$ then there is a non faulty processor $z$ with $\sigma = \alpha; z; \beta$ such that $v_\alpha^z = v_\sigma^p$. Then at round $|\alpha| + 1$ processor $z$ sent $v = v_\alpha^z$ to $q$ as well (as the first processor in $\beta$). Again $v \in V_q$
Correctness

- By lemma previous slide, for any two correct processors we have agreement
  - If $|V_p| > 1$ then $|V_q| > 1$ so both decide on the same value $v_{def}$
  - If $|V_p| = 1$ then $V_p = V_q = \{v\}$ for some $v$ on which both decide

- If all processors start with the same value $v$, then all nodes in any tree equals $v$ or ⊥. Therefore $V_p = \{v\}$ for all correct $p$ who therefore decides on $v$