Failure Detectors

Seif Haridi
haridi@kth.se
Modeling Timing Assumptions

- Tedious to model eventual synchrony (partial synchrony)
- Timing assumptions mostly needed to detect failures
  - Heartbeats, timeouts, etc...

- Use **failure detectors** to encapsulate timing assumptions
  - Black box giving **suspicions** regarding process failures
  - Accuracy of suspicions depends on model strength
Implementation of Failure Detectors

Typical Implementation

- Periodically exchange *heartbeat* messages
- *Timeout* based on *worst case* message round trip
- If timeout, then *suspect* process
- If received message from suspected node, *revise suspicion* and increase time-out
Completeness and Accuracy

- Two important types of requirements
  - 1. Completeness requirements
    - Requirements regarding actually crashed nodes
      - When do they have to be detected?
  - 2. Accuracy requirements
    - Requirements regarding actually alive nodes
      - When are they allowed to be suspected?
Completeness and Accuracy

- In asynchronous system
  - Is it possible to achieve completeness?
    - Yes, suspect all processes

- Is it possible to achieve accuracy?
  - Yes, refrain from suspecting any process!

- Is it possible to achieve both?
  - NO!

- Failure detectors are feasible only in synchronous and partially synchronous systems
Requirements: Completeness

- **Strong Completeness**
  - Every crashed process is *eventually* detected by all *correct* processes

- There exists a time after which all crashed processes are detected by all correct processes
  - We only study failure detectors with this property

- Is it realistic? [d]
Requirements: Completeness

● **Weak Completeness**
  - Every crashed process is *eventually* detected by some *correct* process

● There exists a time after which all crashed nodes are detected by some correct nodes
  - Possibly detected by *different* correct nodes
Requirements: Accuracy

- **Strong Accuracy**
  - No correct process is ever suspected
- For all process p and q,
  - p does not suspect q, unless q has crashed
- Is it realistic? [d]
  - Strong assumption, requires synchrony
  - I.e. no premature timeouts
Requirements: Accuracy

- **Weak Accuracy**
  - There exists a correct process which is never suspected by any process

- There exists a correct node P
  - All nodes will never suspect P

- Still strong assumption
  - One node is always “well-connected”
Requirements: Accuracy

- **Eventual Strong Accuracy**
  - After some finite time the FD provides *strong accuracy*

- **Eventual Weak Accuracy**
  - After some finite time the detector provides *weak accuracy*

- After some time, the requirements are fulfilled
  - Prior to that, any behavior is possible!

- Quite weak assumptions [d]
  - When can eventual weak accuracy be achieved?
Failure Detectors Classes
Four Main Established Detectors

- Four detectors with strong completeness
  - Perfect Detector (P)
    - Strong Accuracy
  - Strong Detector (S)
    - Weak Accuracy

Synchronous Systems

- Eventually Perfect Detector (◊P)
  - Eventual Strong Accuracy
- Eventually Strong Detector (◊S)
  - Eventual Weak Accuracy

Partially Synchronous Systems
Four Less Interesting Detectors

- Four detectors with **weak completeness**
  - Detector Q
    - Strong Accuracy
  - Weak Detector (W)
    - Weak Accuracy

  \[\{\text{Synchronous Systems}\}\]

- Eventually Detector Q (◊Q)
  - Eventual Strong Accuracy
- Eventually Weak Detector (◊W)
  - Eventual Weak Accuracy

  \[\{\text{Partially Synchronous Systems}\}\]

S. Haridi, KTHx ID2203.1x
Prefect Failure
Detector P
Interface of Perfect Failure Detector

- **Module:**
  - Name: PerfectFailureDetector, instance P

- **Events:**
  - Indication (out): \( \langle P, \text{Crash} \mid p_i \rangle \)
    - Notifies that process \( p_i \) has crashed

- **Properties:**
  - \textit{PFD1 (strong completeness)}
  - \textit{PFD2 (strong accuracy)}
Properties of P

• Properties:
  • \textit{PFD1 (strong completeness)}
    • Eventually every process that \textit{crashes} is permanently detected by every correct process
  
  \textit{(liveness)}
  • \textit{PFD2 (strong accuracy)}
    • If a node \textit{p} is detected by any node, then \textit{p} has crashed
  
  \textit{(safety)}
  • Safety or Liveness?
Implementing P in Synchrony

- Assume synchronous system
  - Max transmission delay between 0 and $\delta$ time units

- Each process every $\gamma$ time units
  - Send <heartbeat> to all processes

- Each process waits $\gamma + \delta$ time units
  - If did not get <heartbeat> from $p_i$
    - Detect <crash | $p_i$>
Correctness of P

- **PFD1 (strong completeness)**
  - A crashed process doesn’t send `<heartbeat>`
  - Eventually every process will notice the absence of `<heartbeat>`
Correctness of P

- **PFD2 (strong accuracy)**
  - Assuming local computation is negligible
  - Maximum time between 2 heartbeats
    - $\gamma + \delta$ time units
  - If alive, all process will receive hb in time
    - No inaccuracy
Eventually Prefect Failure Detector ♦ P
Interface of ◇P

- **Module:**
  - Name: EventuallyPerfectFailureDetector, instance ◇P

- **Events:**
  - **Indication:** ◇P, suspect | p_i
    - Notifies that process p_i is suspected to have crashed
  - **Indication:** ◇P, restore | p_i
    - Notifies that process p_i is not suspected anymore

- **Properties:**
  - PFD1 (*strong completeness*)
  - PFD2 (*eventual strong accuracy*). Eventually, no correct process is suspected by any correct process
Implementing \( P \)

- Assume partially synchronous system
  - Eventually some bounds exists

- Each process every \( \gamma \) time units
  - Send <heartbeat> to all processes

- Each process waits \( T \) time units
  - If did not get <heartbeat> from \( p_i \)
    - Indicate <suspect | \( p_i \)> if \( p_i \) is not in suspected set
    - Put \( p_i \) in suspected set
  
  - If get HB from \( p_i \), and \( p_i \) is in suspected
    - Indicate <restore | \( p_i \)> and remove \( p_i \) from suspected
    - Increase timeout \( T \)
Correctness of $\Diamond P$

- **EPFD1 (strong completeness)**
  - Same as before

- **EPFD2 (eventual strong accuracy)**
  - Each time $p$ is inaccurately suspected by a correct $q$
    - Timeout $T$ is increased at $q$
    - Eventually system becomes synchronous, and $T$ becomes larger than the unknown bound $\delta$ ($T > \gamma + \delta$)
    - $q$ will receive HB on time, and never suspect $p$ again
Leader Election
Leader Election versus Failure Detection

- Failure detection captures failure behavior
  - Detect *failed* processes

- Leader election (LE) also captures failure behavior
  - Detect *correct* processes (a single *and* same for all)

- Formally, *leader election is a FD*
  - Always suspects all processes except one (leader)
  - Ensures some properties regarding that process
Leader Election vs. Failure Detection

We will define two leader election abstraction and algorithms

- Leader election (LE) which “matches” P
- Eventual leader election ($\Omega$) which “matches” $\diamondsuit P$
Matching LE and P

- **P’s properties**
  - \( P \) always eventually detects failures (strong completeness)
  - \( P \) never suspects correct nodes (strong accuracy)

- **Completeness of LE**
  - Informally: eventually ditch **failed leaders**
  - Formally: **eventually** every correct process trusts **some** correct node

- **Accuracy of LE**
  - Informally: never ditch a correct leader
  - Formally: No two **correct** processes trust different **correct** nodes
    - Is this really accuracy? [d]
    - Yes! Assume two processes trust different correct processes
      - One of them must eventually switch, i.e. leaving a correct node
LE desirable properties

- LE always eventually detects failures
  - Eventually every correct process trusts some correct node
- LE is always accurate
  - No two correct processes trust different correct processes
- But the above two permit the following
  - But $P_1$ is “inaccurately” leaving a correct leader

```
p_1 -> elect p_3  
    |    
    v
  elect p_3

p_2
  |  
  v 
  elect p_3

p_3
  |  
  v 
  ✗
```
LE desirable properties

- To avoid “inaccuracy” we add
  - Local Accuracy:
    - If a process is elected leader by $p_i$, all previously elected leaders by $p_i$ have crashed

Not allowed, as $p_1$ is correct

```
<table>
<thead>
<tr>
<th></th>
<th>elect $p_3$</th>
<th>elect $p_1$</th>
<th>elect $p_2$</th>
<th>elect $p_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_2$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p_3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

S. Haridi, KTHx ID2203.1x
Interface of Leader Election

- **Module:**
  - Name: LeaderElection (le)

- **Events:**
  - **Indication:** \(\text{leLeader} \mid p_i\)
    - Indicate that leader is node \(p_i\)

- **Properties:**
  - **LE1 (eventual completeness).** Eventually every correct process trusts some correct process
  
  - **LE2 (agreement).** No two correct processes trust different correct processes
  
  - **LE3 (local accuracy).** If a process is elected leader by \(p_i\), all previously elected leaders by \(p_i\) have crashed
Implementing LE

- Globally rank all processes
  - E.g. rank ordering $\text{rank}(p_1) > \text{rank}(p_2) > \text{rank}(p_3) > \ldots$
- $\text{maxrank}(S)$
  - The process $p \in S$, with the largest rank
Implementing LE

- LeaderElection, instance le
- Uses:
  - PerfectFailureDetector, instance P
- upon event \langle le, Init \rangle do
  - suspected := \emptyset
  - leader := \bot
- upon event \langle P, Crash |p \rangle do
  - suspected := suspected \cup \{p\}
- upon leader \neq \text{maxrank}(\Pi \setminus \text{suspected}) do
  - leader := \text{maxrank}(\Pi \setminus \text{suspected})
  - trigger \langle le, Leader | leader \rangle
Eventual Leader Election $\Omega$
Matching $\Omega$ and $\Diamond P$

- $\Diamond P$ weakens $P$ by only providing eventual accuracy
- Weaken LE to $\Omega$ by only guaranteeing eventual agreement

**LE Properties:**

- **LE1** *(eventual completeness).* Eventually every correct node trusts some correct node
- **LE2** *(agreement).* No two correct nodes trust different correct nodes
- **LE3** *(local accuracy).* If a node is elected leader by $p_i$, all previously elected leaders by $p_i$ have crashed
Interface of Eventual Leader Election

• **Module:**
  - Name: EventualLeaderElection ($\Omega$)

• **Events:**
  - Indication (out): $\langle \Omega$, Trust $\mid p_i \rangle$
    - Notify that $p_i$ is trusted to be leader

• **Properties:**
  - $\textit{ELD1 (eventual completeness).}$ Eventually every correct node trusts some correct node
  - $\textit{ELD2 (eventual agreement).}$ Eventually no two correct nodes trust different correct node
Eventual Leader Detection $\Omega$

- In crash-stop process abstraction
  - $\Omega$ is obtained directly from $\Diamond P$

- Each process trusts the process with highest rank among all processes not suspected by $\Diamond P$

- Eventually, exactly one correct process will be trusted by all correct processes
Implementing $\Omega$

- EventualLeaderElection, instance $\Omega$
- **Uses:** EventuallyPerfectFailureDetector, instance $◊P$
- **upon event** $\langle \Omega, \text{Init} \rangle$ **do**
  - suspected := $\emptyset$; leader := $\bot$
- **upon event** $\langle ◊P, \text{Suspect} \mid p \rangle$ **do**
  - suspected := suspected $\cup \{p\}$
- **upon event** $\langle ◊P, \text{Restore} \mid p \rangle$ **do**
  - suspected := suspected $\setminus \{p\}$
- **upon** leader $\neq \text{maxrank}(\Pi \setminus \text{suspected})$ **do**
  - leader := maxrank($\Pi \setminus \text{suspected}$)
  - **trigger** $\langle \Omega, \text{Trust} \mid \text{leader} \rangle$
Ω for Crash Recovery

- Can we elect a recovered process? [d]
  - Not if it keeps crash-recovering infinitely often!
- Basic idea
  - Count number of times you’ve crashed (epoch)
  - Distribute your epoch periodically to all nodes
  - Elect leader with lowest (epoch, rank(node))
- Implementation
  - Similar to ◊P and Ω for crash-stop
  - Piggyback epoch with heartbeats
  - Store epoch, upon recovery load epoch and increment
Reductions
Reductions

- We say $X \leq Y$ if
  - $X$ can be solved given a solution of $Y$
  - Read $X$ is reducible to $Y$
  - Informally, problem $X$ is easier or as hard as $Y$
Preorders, partial orders...

- A relation \( \preceq \) is a preorder on a set \( A \) if for any \( x,y,z \) in \( A \)
  - \( x \preceq x \) (reflexivity)
  - \( x \preceq y \) and \( y \preceq z \) implies \( x \preceq z \) (transitivity)

- Difference between preorder and partial order
  - Partial order is a preorder with anti-symmetry
    - \( x \leq y \) and \( y \leq x \) implies \( x = y \)
  - For preorder two different objects \( x \) and \( y \) can be symmetric
    - It is possible that \( x \preceq y \) and \( y \preceq x \) for two different \( x \) and \( y \), \( (x \neq y) \)
Reducibility $\preceq$ is a preorder

- $\preceq$ is a preorder
  - **Reflexivity.** $X \preceq X$
    - $X$ can be solved given a solution to $X$
  - **Transitivity.** $X \preceq Y$ and $Y \preceq Z$ implies $X \preceq Z$
    - Since $Y \preceq Z$, use implementation of $Z$ to implement $Y$
      use that implementation of $Y$ to implement $X$
    - Hence we implemented $X$ from $Z$’s implementation

- $\preceq$ is not anti-symmetric, thus not a partial order
  - Two different $X$ and $Y$ can be equivalent
    - Distinct problems $X$ and $Y$ can be solved from the other’s solution
Shortcut definitions

- We write $X \cong Y$ if
  - $X \preceq Y$ and $Y \preceq X$
  - Problem $X$ is equivalent to $Y$

- We write $X \prec Y$ if
  - $X \preceq Y$ and not $X \cong Y$
  - or equivalently, $X \preceq Y$ and not $Y \preceq X$
  - Problem $X$ is strictly weaker than $Y$, or
  - Problem $Y$ is strictly stronger than $X$
Example

- It is true that \( \Diamond P \leq P \)
  - Given P, we can implement \( \Diamond P \)
    - We just return P’s suspicions.
    - P always satisfies \( \Diamond P \)’s properties
- In fact, \( \Diamond P < P \) in the asynchronous model
  - Because not \( P \leq \Diamond P \) is true
- Reductions common in computability theory
  - If \( X \leq Y \), and if we know X is impossible to solve
    - Then Y is impossible to solve too
  - If \( \Diamond P \leq P \), and some problem Z can be solved with \( \Diamond P \)
    - Then Z can also be solved with P
Weakest FD for a problem?

- Often P is used to solve problem X
  - But P is not very practical (needs synchrony)
  - Is X a “practically” solvable problem?
    - Can we implement X with ◊P?
    - Sometimes a weaker FD than P will not solve X
      - Proven using reductions
Weakest FD for a problem

- Common proof to show $P$ is weakest FD for $X$
  - Prove that $P \preceq X$
  - I.e. $P$ can be solved given $X$
- If $P \preceq X$ then $◊P < X$
  - Because we know $◊P < P$ and $P \simeq X$, i.e. $◊P < P \simeq X$
    - If we can solve $X$ with $◊P$, then
    - we can solve $P$ with $◊P$, which is a contradiction
How are the detectors related
Trivial Reductions

- Strongly complete
  - ♦P ≤ P
    - P is always strongly accurate, thus also eventually strongly accurate
  - ♦S ≤ S
    - S is always weakly accurate, thus also eventually weakly accurate
  - S ≤ P
    - P is always strongly accurate, thus also always weakly accurate
  - ♦S ≤ ♦P
    - ♦P is always eventually strongly accurate, thus also always eventually weakly accurate
Trivial Reductions (2)

- Weakly complete
  - ◊Q ≤ Q
    - Q is always strongly accurate, thus also eventually strongly accurate
  - ◊W ≤ W
    - W is always weakly accurate, thus also eventually weakly accurate
  - W ≤ Q
    - Q is always strongly accurate, thus also always weakly accurate
  - ◊W ≤ ◊Q
    - ◊Q is always eventually strongly accurate, thus also always eventually weakly accurate
Completeness “Irrelevant”

Weak completeness **trivially reducible** to strong

Strong completeness **reducible** to weak

- i.e. can get strong completeness from weak
  \[ P \preceq Q, \ S \preceq W, \ \diamond P \preceq \diamond Q, \ \diamond S \preceq \diamond W, \]

- They’re **equivalent**!
  \[ P \simeq Q, \ S \simeq W, \ \diamond P \simeq \diamond Q, \ \diamond S \simeq \diamond W \]

<table>
<thead>
<tr>
<th>Completeness</th>
<th>Strong</th>
<th>Weak</th>
<th>Eventual Strong</th>
<th>Eventual Weak</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong</td>
<td>P</td>
<td>S</td>
<td>\diamond P</td>
<td>\diamond S</td>
</tr>
<tr>
<td>Weak</td>
<td>Q</td>
<td>W</td>
<td>\diamond Q</td>
<td>\diamond W</td>
</tr>
</tbody>
</table>

S. Haridi, KTHx ID2203.1x
Proving Irrelevance of Completeness

- Weak completeness ensures
  - every crash is eventually detected by some correct node

- Simple idea
  - Every process $q$ broadcast suspicions $\text{Susp}$ periodically
  - upon event receive $<S,q>$
    - $\text{Susp} := (\text{Susp} \cup S) - \{q\}$
  - Every crash is eventually detected by all correct $p$
  - Can this violate some accuracy properties?

Also works like a heartbeat
Maintaining Accuracy

- Strong and Weak Accuracy aren’t violated

Strong accuracy
- No one is ever inaccurate
- Our reduction never spreads inaccurate suspicions

Weak accuracy
- Everyone is accurate about at least one process p
  - No one will spread inaccurate information about p
Maintaining Eventual Accuracy

- Eventual Strong and Eventual Weak Accuracy aren’t violated

- Proof is almost same as previous page
  - Eventually all faulty processes crash
  - Inaccurate suspicions undone
  - Will get heartbeat from correct nodes and revise (\(-\{q\}\))
Relation between FDs

\[ \diamond P \rightarrow P \rightarrow Q \rightarrow W \rightarrow W \rightarrow S \rightarrow \diamond S \]

Equivalent reducible to
Omega also a FD

• Can we implement ◊S with Ω? [d]
  • I.e. is it true that ◊S ≤ Ω
  • Suspect all nodes except the leader given by Ω
    • Eventual Completeness
      ▪ All nodes are suspected except the leader (which is correct)
    • Eventual Weak Accuracy
      ▪ Eventually, one correct node (leader) is not suspected by anyone
  • Thus, ◊S ≤ Ω
$\Omega$ equivalent to $\diamond S$ (and $\diamond W$)

- We showed $\diamond S \preceq \Omega$, it turns out we also have $\Omega \preceq \diamond S$
  - I.e. $\Omega \simeq \diamond S$

- The famous CHT (Chandra, Hadzilucas, Toueg) result
  - If consensus implementable with detector $D$
    Then Omega can be implemented using $D$
  - I.e. if $\text{Consensus} \preceq D$, then $\Omega \preceq D$
    - Since $\diamond S$ can be used to solve consensus, we have $\Omega \preceq D$
    - Implies $\diamond W$ is weakest detector to solve consensus
Relation between FDs (2)

\[ P \iff Q \iff W \iff S \iff \Omega \]

equivalent reducible to
Combining Abstractions
Combining Abstractions

- **Fail-stop** (synchronous)
  - Crash-stop process model
  - Perfect links + Perfect failure detector (P)

- **Fail-silent** (asynchronous)
  - Crash-stop process model
  - Perfect links

- **Fail-noisy** (partially synchronous)
  - Crash-stop process model
  - Perfect links + Eventually Perfect failure detector (◊P)

- **Fail-recovery**
  - Crash-recovery process model
  - Stubborn links + ...
The rest of course

- Assume crash-stop system with a perfect failure detector (fail-stop)
  - Give algorithms

- Try to make a weaker assumption
  - Revisit the algorithms