INSTITUT FEMTO-ST

UMR CNRS 6174

The proof of the FLC consensus algorithm (V2)

Fouad Hanna — Lionel Droz-Bartholet — Jean-Christophe Lapayre

Rapport Technique **no** RTDISC2014-1

DÉPARTEMENT DISC – March 26, 2015

The proof of the FLC consensus algorithm (V2)

Fouad Hanna , Lionel Droz-Bartholet , Jean-Christophe Lapayre

Département DISC

CARTOON

March 26, 2015 (16 pages)

Abstract: It is well known that consensus algorithms are fundamental building blocks for fault tolerant distributed systems. In this technical report we present the correctness proof of our new leader-based consensus algorithm (FLC algorithm) for the crash-stop failure model. Our algorithm uses the leader oracle Ω and adapts a decentralized communication pattern.

Key-words: Fault tolerance, consensus, asynchronous distributed systems, unreliable failure detectors, leader oracle Ω.

FEMTO-ST Institute, DISC research department
UFR Sciences - Route de Gray - F-25030 BESANÇON cedex FRANCE
Tél : (33 3) 81 66 64 00 – Fax : (33 3) 81 66 64 23 – e-mail : brigitte.bataillard@univ-fcomte.fr

La preuve de l'algorithme de consensus FLC (V2)

Résumé : Il est bien connu que les algorithmes de consensus sont des éléments fondamentaux pour les systèmes distribués tolérant aux pannes. Dans ce rapport technique, nous présentons la preuve de notre nouvel algorithme de consensus à base de leader (l'algorithme FLC). Notre algorithme utilise l'oracle leader Ω, adapte un schéma de communication décentralisé et considère un modèle de panne crash-stop.

Mots-clés : Tolérance aux pannes, consensus, systèmes distribués asynchronnes, détecteurs de pannes non fiables, l'oracke leader Ω.

FEMTO-ST Institute, DISC research department
UFR Sciences - Route de Gray - F-25030 BESANÇON cedex FRANCE
Tél : (33 3) 81 66 64 00 – Fax : (33 3) 81 66 64 23 – e-mail : brigitte.bataillard@univ-fcomte.fr

1 Proof of the FLC algorithm

We present here the correctness proof of our algorithm. The proof goes through three properties: Termination, Agreement and validity. To simplify the presentation of the proof we use the word vote to indicate the $LeaderAck(r)$ message as well as the integrated leader vote in the EST message. Accordingly, when we say that a process is elected to be leader, this means that either it has received a majority of $LeaderAck(r)$ messages during the current round (this is the case of the first round of the first consensus to be executed) or that it has received a majority of integrated leader votes during the previous round. The election using $LeaderAck(r)$ messages is used only for the first round of the first consensus (i.e. when $r_i = 0 \wedge c = 0$). The integrated leader votes are used to elect a leader when $r_i > 0$ (i.e. for every round after the first one) regardless of the consensus number c. Finally, for the first round of every consensus after the first one (i.e. when $r_i = 0 \wedge c > 0$), processes will designate the leader of the last consensus $lastConsLeader$ as the leader of the current one and thus no voting mechanism is used during this round. In all three cases, we say that a leader is designated and the variable $leaderChosen_i$ will be set to true. The algorithm is presented in figure 1.

Axiom 1. *The communication network does not lose, alter nor duplicate messages.*

Axiom 2. *Eventual leadership property of the leader oracle* Ω*: There is a time after which all correct processes always trust the same correct process.*

Axiom 3. *A process does not start consensus* $c + 1$ *before terminating consensus* c.

Lemma 1. *A correct process does not block infinitely during a round.*

Proof. Let r be a round during which a process p_i blocks forever, we show that this is impossible. A process can block only at lines 14 and 17. We start by considering line 14. At line 14 we have two possibilities after the leader designation phase (phase 1) (Figure 1 lines 7-13):

- A *leader* was designated: In this case, depending on the consensus number c, either one of the processes has been elected to be leader or the leader of the previous consensus is not crashed and has been designated to be leader of the round. If the designated leader process is process p_i then it will have $leaderChosen_i = true$. As a result, the waiting condition at line 14 will be broken. Depending on the values of the

```
1:  procedure MainTask ($c$, $lastConsLeader$)                                  ▷ The body of the algorithm
2:      $r_i \leftarrow -1$; $electedLeader_i \leftarrow false$; $est_i \leftarrow [v_i, i]$;          ▷ $v_i \neq \perp$
3:      while $true$ do
4:          $r_i \leftarrow r_i + 1$; $estFromLeader_i \leftarrow [\perp, i]$; $leaderChosen_i \leftarrow false$;
5:          $ph1End_i \leftarrow false$; $leaderChanged_i \leftarrow false$; $nbLeaderVotes_i \leftarrow 0$; $nbNextRoundLeaderVotes_i \leftarrow 0$;
6:          $nbEST_i \leftarrow 0$; $nbNullEST_i \leftarrow 0$; $nbNonNullEST_i \leftarrow 0$;
7:          if $r_i = 0 \wedge c = 0$ then                                   ▷ First round of the first consensus run
8:              $leader_i \leftarrow getOmegaLeader()$;
9:              send $LeaderAck(r_i)$ to $leader_i$;
10:         else if $(r_i = 0 \wedge lastConsLeader = i) \bigvee (r_i > 0 \wedge electedLeader_i = true)$ then
11:             $estFromLeader_i \leftarrow est_i$; $leader_i \leftarrow i$; $leaderChosen_i \leftarrow true$;
12:         end if
13:         $ph1End_i \leftarrow true$;
14:         wait until $(leaderChosen_i = true \bigvee leaderChanged_i = true \bigvee nbNullEST_i > 0)$;
15:                                                 ▷ At this point we have $estFromLeader_i = est$ or $\perp$
16:         $\forall j$: send $EST(r_i, leader_i, estFromLeader_i, getOmegaLeader())$ to $j$;
17:         wait until $(nbEST_i \geq \lceil \frac{n+1}{2} \rceil)$;                  ▷ Wait to receive a majority of $EST$ messages
18:         if $nbNullEST_i \geq \lceil \frac{n+1}{2} \rceil$ then skip;                        ▷ Skip to the next round
19:         else if $nbNonNullEST_i \geq \lceil \frac{n+1}{2} \rceil$ then
20:             $lastConsLeader \leftarrow est[1]$; $\forall j \neq i$: send $Decide(est)$ to $j$; exit$(est[0], lastConsLeader)$;
21:         else $est_i \leftarrow est$;
22:         end if
23:         if $nbNextRoundLeaderVotes_i \geq \lceil \frac{n+1}{2} \rceil$ then          ▷ Process $i$ is elected leader for the next round
24:             $electedLeader_i \leftarrow true$;
25:         else $electedLeader_i \leftarrow false$;
26:         end if
27:     end while
28: end procedure

29: procedure NewOmegaLeader($j$)                                      ▷ Called by $\Omega$ when the $leader$ changes
30:     if $ph1End_i = true \wedge leaderChosen_i = false$ then
31:         $leader_i \leftarrow j$; $leaderChanged_i \leftarrow true$;
32:     end if
33: end procedure

34: procedure DeliverMessage($msg$)                                      ▷ Called when a message is received
35:     if $msg = LeaderAck(r_i)$ then
36:         if $leaderChosen_i = false \wedge r_i = 0 \wedge lastConsLeader = -1$ then
37:             $nbLeaderVotes_i + +$;
38:             if $nbLeaderVotes_i \geq \lceil \frac{n+1}{2} \rceil$ then                             ▷ Process $i$ is elected leader
39:                 $estFromLeader_i \leftarrow est_i$; $leader_i \leftarrow i$; $leaderChosen_i \leftarrow true$;
40:             end if
41:         end if
42:     else if $msg = Decide(est)$ then
43:         $lastConsLeader \leftarrow est[1]$; $\forall j \neq i$: send $Decide(est)$ to $j$; exit$(est[0], lastConsLeader)$;
44:     else if $msg = EST(r_i, leader, est, nextRoundLeader)$ then
45:         if $est[0] \neq \perp$ then
46:             $nbNonNullEST_i + +$;
47:             if $leaderChosen_i = false$ then
48:                 $estFromLeader_i \leftarrow est$; $leader_i \leftarrow leader$; $leaderChosen_i \leftarrow true$;
49:             end if
50:         else $nbNullEST_i + +$;
51:         end if
52:         if $nextRoundLeader = i$ then $nbNextRoundLeaderVotes_i + +$;
53:         end if
54:         $nbEST_i + +$;
55:     end if
56: end procedure
```

Figure 1: The new consensus algorithm

consensus number c and the round number r, the path corresponding to this case is one of the following (Figure **??**): $1 \rightarrow 2 \rightarrow 6$ when $r > 0$, $1 \rightarrow 3 \rightarrow 7$ when $r = 0 \wedge c > 0$ and $1 \rightarrow 4 \rightarrow 5 \rightarrow 11$

when $r = 0 \wedge c = 0$. However if process p_i is not the designated *leader* of round r then based on axiom 1 it will eventually receive an $EST(r_i, leader, v \neq \perp, nextRoundLeader)$ message from the designated *leader* of the round. As a result process p_i will execute line 48 which will set the variable $leaderChosen_i = true$ and consequently break the waiting condition at line 14. In this case, process p_i follows one of the following paths (Figure **??**): $1 \rightarrow 2 \rightarrow 10$ when $r > 0$, $1 \rightarrow 3 \rightarrow 10$ when $r = 0 \wedge c > 0$ and $1 \rightarrow 4 \rightarrow 5 \rightarrow 10$ when $r = 0 \wedge c = 0$.

- A *leader* could not be designated: This can happen either because no process succeeded to accumulate a majority of votes or because the designated leader process (i.e. the elected process or the leader of the last consensus) is crashed. In the latter case, suppose that process p_x is the crashed *leader*. Then, the function $NewOmegaLeader(j)$ of process p_i (with $i, j \neq x$) will be eventually called by the leader oracle Ω due to its eventual leadership property (axiom 2). As a result, the variable $leaderChanged_i$ will be set to $true$ and the waiting condition at line 14 will be broken. Depending on the values of the consensus number c and the round number r, the path corresponding to this case is one of the following (Figure **??**): $1 \rightarrow 2 \rightarrow 8$ when $r > 0$, $1 \rightarrow 3 \rightarrow 8$ when $r = 0 \wedge c > 0$ and $1 \rightarrow 4 \rightarrow 5 \rightarrow 8$ when $r = 0 \wedge c = 0$. In the first case, where no process succeeded to accumulate a majority of votes, no process will become *leader* and therefore process p_i will be blocked at line 14. Due to the eventual leadership property (axiom 2), the leader oracle Ω at each process will eventually give the same correct process (let it be j) to be considered *leader* of the round. Two cases can happen: (1) process p_i voted for a process p_x where $x \neq j$ then the function $NewOmegaLeader(j)$ will be called and the variable $leaderChanged_i$ will be set to $true$. As a result, the waiting condition at line 14 will be broken and process p_i will proceed by sending the null value $\perp$ to all other processes at line 16. (2) process p_i has voted for the eventual leader j then the function $NewOmegaLeader(j)$ will not be called because the eventual leadership property (axiom 2) of the leader oracle Ω is already satisfied. Let us suppose the worst case where there are only a majority of correct processes. In this case, there must be at least a process p_x, with $x \neq i$, voted for a process different from j. This situation leads process p_x to be in case (1). Consequently, p_x will send a null estimate to all other processes (message $EST(r_i, leader, \perp, nextRoundLeader)$). By axiom 1, process p_i will receive the null estimate from p_x and will break the waiting condition at line 14 accordingly (it will have $nbNullEST_i > 0$). In this situation, process p_i follows one of the following paths in the execution diagram presented in Figure **??**: $1 \rightarrow 2 \rightarrow 9$ when $r > 0$, $1 \rightarrow 3 \rightarrow 9$ when $r = 0 \wedge c > 0$ and $1 \rightarrow 4 \rightarrow 5 \rightarrow 9$ when $r = 0 \wedge c = 0$.

This means that no correct process blocks forever at line 14. As a result, all correct processes will execute line 16 by sending a $EST(r_i, leader, v, nextRoundLeader)$ message to all other processes. By axiom 1 and as there is always a majority of correct processes in the system then each correct process will receive at least a majority of $EST(r_i, leader, v, nextRoundLeader)$ messages (i.e. $nbEST_i \geq \lceil \frac{n+1}{2} \rceil$). Consequently, no correct process blocks forever at line 17. □

Lemma 2. *For the first consensus, i.e. when $c = 0$: (a) there is at most one leader per round and (b) no two processes decide differently regardless of the round number.*

Proof. We start by proving the first part (a) which will be used to prove the second part (b).

(a) From the algorithm (figure 1) we notice that during the first consensus $c = 0$, a leader can be designated only by using votes: either the $LeaderAck(r)$ message when $r_i = 0$ or the integrated leader vote in the EST message when $r_i > 0$. Therefore, a process p_i becomes *leader* of the round if it receives a majority of votes. According to the algorithm, each process diffuses at most one time the EST message (line 16) and sends at most one $LeaderAck(r)$ message during round $r_i = 0$ (line 9). Consequently, a process votes only one time per round. The proof is by contradiction. We now suppose the existence of two *leaders* during round r: p_i and p_j. This means that p_i has received votes to become leader from a majority of processes Q_i and that p_j has also received votes to become leader from a majority of processes Q_j. As $|\, Q_i \,| \geq \lceil \frac{n+1}{2} \rceil$ and $|\, Q_j \,| \geq \lceil \frac{n+1}{2} \rceil$ then $Q_i \cap Q_j \neq \emptyset$. It follows that $\exists p_x$ such that $p_x \in Q_i \cap Q_j$. This means that the process p_x sent 2 different votes during round r. This leads us to a contradiction with the fact that a process can send at most one vote per round. Therefore, it is impossible to have more than one *leader* per round during the first consensus $c = 0$. **(b)** Processes can decide either during the same round or during different rounds. Therefore, we start by proving that if two processes decide (at line 20), during the same round r, then they decide the same value. The proof is by contradiction. Suppose that process p_i and process p_j decide at line 20 two different values v_1 and v_2 respectively. Knowing that a process can send at most one $EST(r, leader, v, nextRoundLeader)$ message per round, then there must be two leaders for the round r (the first proposed the value v_1 while the second proposed the value v_2). This contradicts the first part of this lemma (a) which says that for the first consensus there cannot be more than one leader per round. As a result, $v_1 = v_2$ and the two processes p_i and p_j decided the same value. Now we show that even if two processes decide during two different rounds, then they will also decide the same value. Let r_1 be the first round during which a process p_i decides a value v_1 at line 20. Let us assume that another process decides a different value v_m during round $r_m > r_1$. To prove that $v_1 = v_m$ we show that the local estimate value of every process p_j that progresses to round $r_1 + 1$ is equal to v_1 and thus no other value can be decided in a future round. For a process to finish a round (or to decide) it has to receive the $EST(r, leader, v, nextRoundLeader)$ message from a majority of correct processes. Let Q_i and Q_j be the set of processes from which p_i and p_j has received the $EST(r_1, leader, v, nextRoundLeader)$ message during round r_1. Since Q_i and Q_j are both majority sets then $Q_i \cap Q_j \neq \emptyset$ and $\exists p_x$ such that $p_x \in Q_i \cap Q_j$. As p_i decided the value v_1 during round r_1 at line 20, then p_i has received v_1 from all the processes of Q_i and consequently from p_x. As a direct result of (a) we know that no other value than v_1 could be exchanged during round r_1 because we cannot have more than one *leader* per round and only a leader process can propose a value. This means that p_j also received the value v_1 from p_x during round r_1 and has consequently executed line 46 making the variable $nbNonNullEST_j > 0$. At any moment of the execution we have $nbEST_j = nbNullEST_j + nbNonNullEST_j$ and at the end of round r_1 we have $nbNonNullEST_j > 0$ then $nbNullEST_j < \lceil \frac{n+1}{2} \rceil$. From the algorithm we notice that a process can proceed to the next round by either executing line 18 or line 21. As p_j has $nbNullEST_j < \lceil \frac{n+1}{2} \rceil$ and at the same time $nbNonNullEST_j < \lceil \frac{n+1}{2} \rceil$ (because p_j does not decide during round r_1 by assumption) then it will execute line 21. As a result, process p_j proceeds to round $r_1 + 1$ with its local estimate $est_j = v_1$ (path $12 \rightarrow 13 \rightarrow 16$ or path $12 \rightarrow 13 \rightarrow 17$ in Figure **??**). This means that any process that proceeds to round $r_1 + 1$ will have its local estimate equal to v_1 and therefore the only value that can be decided during a round $r_m > r_1$ is v_1 because no other value is present in the system. By returning to our assumption we have that the value v_m was decided during a round $r_m > r_1$ and therefore $v_1 = v_m$. □

Lemma 3. *For any consensus, (a) there is at most one leader per round and (b) no two processes decide differently regardless of the round number.*

Proof. The proof is by induction on the consensus number c.

- When $c = 0$: This was proven by lemma 2.
- We suppose the lemma is correct for a given consensus number $c > 0$ (an arbitrary number for c). This means that for consensus c, we suppose

that there is at most one leader per round and no two processes decide differently regardless of the round number.

- Now we prove the lemma for the consensus number $c+1$. We start by proving the first part (a) which is then used to prove the second part (b).

  (a) From the algorithm (Figure 1) we notice that during any consensus $c > 0$ a leader can be designated using one of two methods depending on the round number r (line 10, Figure 1): (1) For the first round (i.e. when $r_i = 0$), the leader of the previous consensus will be designated to be leader of the round and (2) for any $r_i > 0$, the leader is elected using the integrated leader votes in the EST messages. For (2) the proof is identical to lemma 2-a because we use the same leader election mechanism which requires a majority of votes for a process to become leader. The proof for (1) is done by contradiction. We now suppose the existence of two *leaders* during round $r_i = 0$ of consensus $c+1$: p_i and p_j. This means that $lastConsLeader = i$ for p_i and that $lastConsLeader = j$ for p_j. From axiom 3 we have that consensus c which precedes consensus $c+1$ had been terminated before starting consensus $c+1$ (i.e. consensus c is finished and a decision is made). From the algorithm we notice that the variable $lastConsLeader$ for consensus $c+1$ is returned when a decision is made during consensus c and represents the identifier of the process that originally proposed the decision value (line 20 or 43, Figure 1). Also form the algorithm we notice that an estimation est is composed of the value of the estimation and the identifier of the process that proposed it. The process identifier is attached to the estimation value at the beginning of the consensus (line 2, Figure 1) and never changes during the execution of the consensus. By returning to our assumption, we have that $lastConsLeader = i$ for p_i and that $lastConsLeader = j$ for p_j at the end of consensus c. This means that the estimation est that was made as a decision at process p_i is different from the est that was made as a decision at process p_j and this contradicts the fact that no two processes decide differently regardless of the round number during consensus c (assumption made for the proof by induction on c). Therefore, only one leader can exist for the first round (i.e. when $r_i = 0$). Consequently, for any round number r during consensus $c+1$, at most one leader per round. This concludes the proof of part (a) of the lemma.

  (b) The proof is very similar to that of lemma 2-b because the only difference between the first consensus ($c = 0$) and any other consensus ($c > 0$) is in the way the leader of the first round is designated while all the other parts of the algorithms are the same. Processes can decide either during the same round or during different rounds. Therefore, we start by proving that if two processes decide (at line 20), during the same round r, then they decide the same value. The proof is by contradiction. Suppose that process p_i and process p_j decide at line 20 two different values v_1 and v_2 respectively. Knowing that a process can send at most one $EST(r, leader, v, nextRoundLeader)$ message per round, then there must be two leaders for the round r (the first proposed the value v_1 while the second proposed the value v_2). This contradicts the first part of this lemma (a) which says that for any consensus there cannot be more than one leader per round. As a result, $v_1 = v_2$ and the two processes p_i and p_j decided the same value. Now we show that even if two processes decide during two different rounds, then they will also decide the same value. Let r_1 be the first round during which a process p_i decides a value v_1 at line 20. Let us assume that another process decides a different value v_m during round $r_m > r_1$. To prove that $v_1 = v_m$ we show that the local estimate value of every process p_j that progresses to round $r_1 + 1$ is equal to v_1 and thus no other value can be decided in a future round. For a process to finish a round (or to decide) it has to receive the $EST(r, leader, v, nextRoundLeader)$ message from a majority of correct processes. Let Q_i and Q_j be the set of processes from which p_i and p_j has received the $EST(r_1, leader, v, nextRoundLeader)$ message during round r_1. Since Q_i and Q_j are both majority sets then $Q_i \cap Q_j \neq \emptyset$ and $\exists p_x$ such that $p_x \in Q_i \cap Q_j$. As p_i decided the value v_1 during round r_1 at line 20, then p_i has received v_1 from all the processes of Q_i and consequently from p_x. As a direct result of (a) we know that no other value than v_1 could be exchanged during round r_1 because we cannot have more than one *leader* per round and only a leader process can propose a value. This means that p_j also received the value v_1 from p_x during round r_1 and has consequently executed line 46 making the variable $nbNonNullEST_j > 0$. At any moment of the execution we have $nbEST_j = nbNullEST_j + nbNonNullEST_j$ and at the end of round r_1 we have $nbNonNullEST_j > 0$ then $nbNullEST_j < \lceil \frac{n+1}{2} \rceil$. From the algorithm we notice that a process can proceed to the next round by either executing line 18 or line 21. As p_j has $nbNullEST_j < \lceil \frac{n+1}{2} \rceil$ and at the same time $nbNonNullEST_j < \lceil \frac{n+1}{2} \rceil$ (because p_j does not decide during round r_1 by assumption) then it will execute line 21. As a result, process p_j proceeds to round $r_1 + 1$ with its local estimate $est_j = v_1$ (path $12 \rightarrow 13 \rightarrow 16$ or path $12 \rightarrow 13 \rightarrow 17$ in Figure **??**).

This means that any process that proceeds to round $r_1 + 1$ will have its local estimate equal to v_1 and therefore the only value that can be decided during a round $r_m > r_1$ is v_1 because no other value is present in the system. By returning to our assumption we have that the value v_m was decided during a round $r_m > r_1$ and therefore $v_1 = v_m$. This concludes the proof for part (b) of this lemma.

Since both the basis and the inductive step have been performed, by induction, the lemma holds for any consensus number c. □

1.1 Termination

Theorem 1. *Every correct process eventually decides.*

Proof. When a process decides it sends a decision message ($\mathit{Decide}(v)$) to all other processes at line 20. Therefore, by axiom 1, any other correct process will receive this message and will decide accordingly at line 43.

A leader can be designated either by voting or by designating the leader of the last consensus as leader of the round when $r_i = 0 \wedge c > 0$. In the latter case and if the leader of the last consensus, let it be p_l, is not crashed then it will proceed to the second phase of the algorithm. In the first case where the leader has to be elected using votes ($\mathit{LeaderAck}(r)$ messages or integrated leader votes) then we have from lemma 1 that no correct process blocks forever during a round. Based on this lemma and the eventual leadership property of the leader oracle Ω (axiom 2), there exist a round r and a correct process p_l such that each correct process p_i has $\mathit{leader}_i = l$ (i.e. the call of $\mathit{getOmegaLeader}()$ at lines 8 and 16 returns the same correct process p_l to be elected leader at all correct processes). This means that each correct process will send a vote to p_l. As a result and since there is always a majority of correct processes in the system and by axiom 1, the process p_l will receive a majority of votes (i.e. $\mathit{nbLeaderVotes}_l \geq \lceil \frac{n+1}{2} \rceil$) and becomes *leader* of the round.

It follows that the *leader* p_l sends its local estimate of the decision value est_l to all other processes at line 16. Now every correct process waiting at line 14 will receive this non null estimate est_l and will proceed by resending it to all other processes. There are not any null values $\perp$ that are being exchanged among the processes during this round r because the eventual leadership property (axiom 2) makes all correct processes trust the same correct process p_l to be *leader* of the round. According to lemma 2, only one leader per round exists (process p_l) and therefore est_l is the only non null estimate that can be received by any process. Consequently, est_l is the only value that can be sent at line 16. Thus, every correct process will receive a majority of the non null estimate est_l and will decide accordingly at line 16. □

1.2 Agreement

The proof of this property is covered by the part (b) of lemma 3 which says that for any consensus, no two processes decide differently regardless of the round number. Consequently, all correct processes decide the same value.

1.3 Validity

The proof of this property is quite straightforward. From the algorithm, it is obvious that if a process decides then it decides a value that was proposed by a *leader* process and any process p_i that becomes *leader* during a round r will execute line 11 or line 39 affecting its local estimate to the variable $estFromLeader_i$ which will be sent to all other processes at line 16.

FEMTO-ST Institute

FEMTO-ST INSTITUTE, headquarters
32 avenue de l'Observatoire - F-25044 BESANÇON Cedex FRANCE
Tél : (33 3) 81 85 39 99 – Fax : (33 3) 81 85 39 68 – e-mail : contact@femto-st.fr

FEMTO-ST - AS2M : TEMIS, 24 rue Alain Savary, F-25000 Besançon
FEMTO-ST - DISC : UFR Sciences - Route de Gray - F-25030 Besançon cedex France
FEMTO-ST - ENERGIE : Parc Technologique, 2 Av. Jean Moulin, Rue des entrepreneurs, F-90000 Belfort France
FEMTO-ST - MEC'APPLI : 24, chemin de l'épitaphe - F-25000 Besançon France
FEMTO-ST - MN2S : 32, rue de l'Observatoire - F-25044 Besançon cedex France
FEMTO-ST - OPTIQUE : UFR Sciences - Route de Gray - F-25030 Besançon cedex France
FEMTO-ST - TEMPS-FREQUENCE : 26, Chemin de l'Epitaphe - F-25030 Besançon cedex France

http://femto-st.fr