Laboratoire 3

Implémentation d'horloges logiques pour la synchronisation

Équipe :

GENTIL Adrien 1499297
KIRCHNER Dimitri 1502458
KRANTZ Xavier 1495708

Introduction

Les applications et systèmes répartis possèdent des caractéristiques propres comme la décomposition en **composantes réparties sur des matériels différents, autonomes et concurrents**. Cette décomposition expose un **problème complexe qui est la notion commune du temps**. Un des défis dans la conception de systèmes répartis est donc la coordination des actions, des interactions et la synchronisation de ces composantes afin d'obtenir un système global opérationnel et performant. Pour illustrer ce problème nous nous sommes exercés à l'implémentation d'un système de capteurs d'automobiles utilisant une horloge logique à base d'estampilles vectorielles.

I. Un diagramme de classes

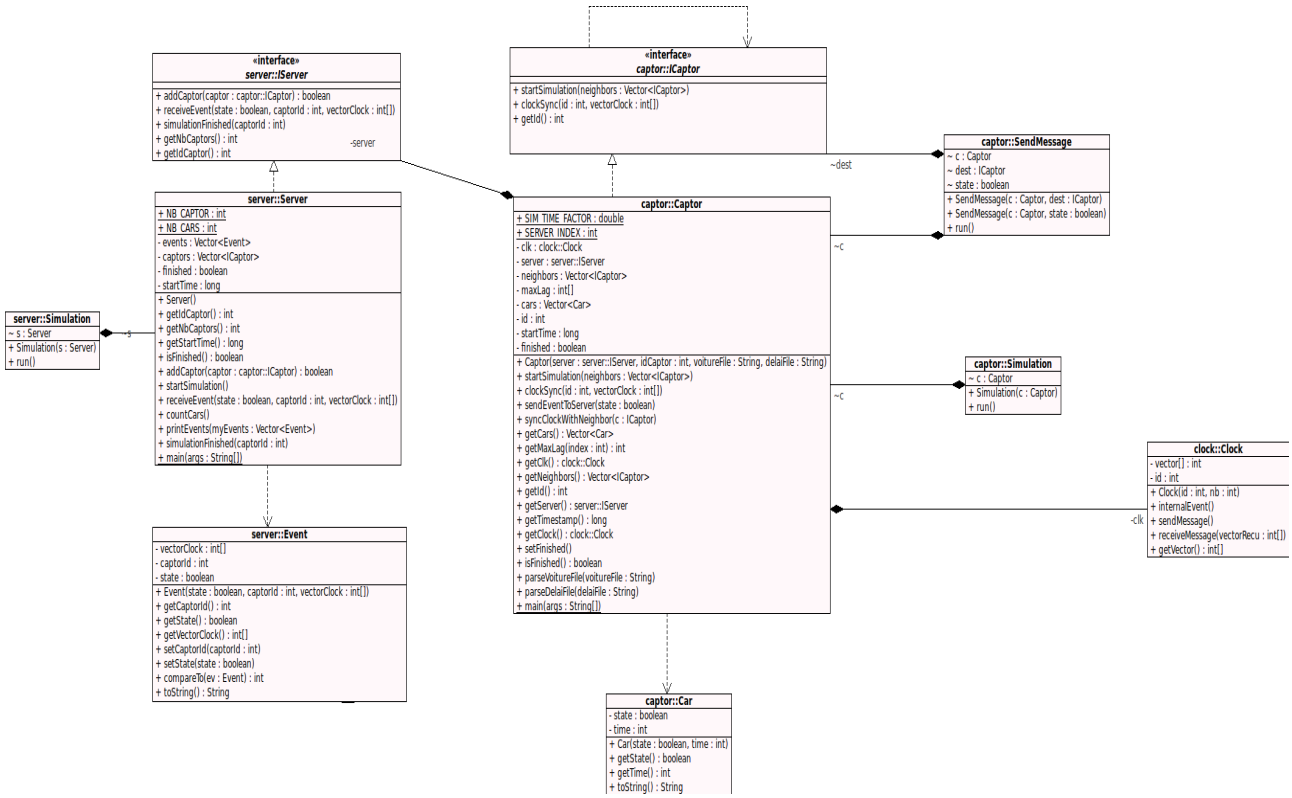


Diagramme de classes de l'application

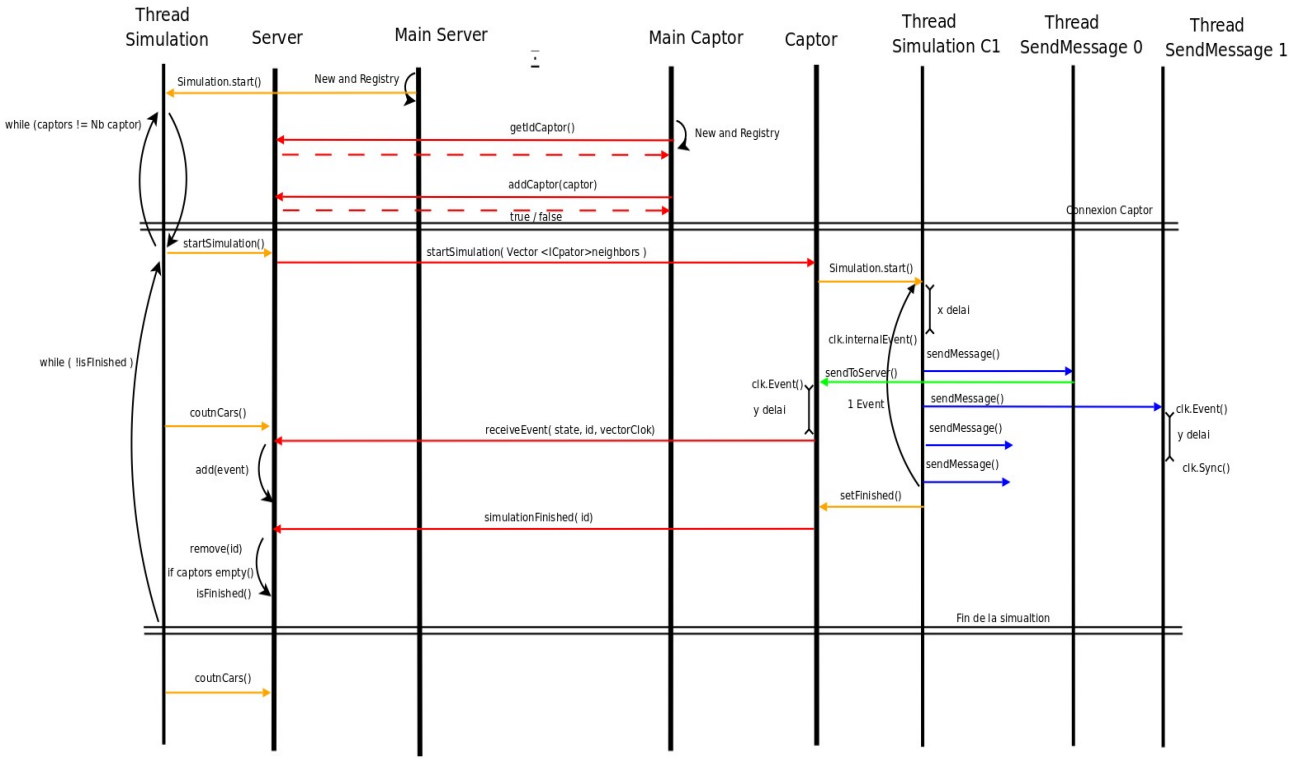


Diagramme de séquence

Description :

Pour comprendre la manière dont nous avons développé notre application, nous devons comprendre le fonctionnement d'un réseau de capteurs. Le principe repose sur **un serveur** centrale **qui reçoit des messages provenant de capteurs différents et qui ne sait pas à l'avance combien de messages il recevra**. Ainsi, nous avons décomposé l'exécution d'une simulation de capture en trois périodes principales :

- L'enregistrement des capteurs auprès du serveur,
- La réalisation des captures, l'envoie de messages, leur traitement de la part du serveur,
- Et la fin de la simulation avec l'affichage final des résultats.

Pour mettre en évidence le principe d'horloge logique et de synchronisation par vecteur d'horloge, les capteurs détectent et envoient leur évènement avec des délais de transmission différents les uns des autres. Afin de rendre ces **éléments indépendants** et non bloquants pour les différentes instances d'un capteur, **nous avons utilisé des "threads"**.

Le **thread de simulation du serveur** effectue une première boucle d'attente le temps que chaque capteur s'enregistre et obtienne un identifiant. Ensuite, il lance le thread de simulation de chaque capteurs. Pendant le temps des captures, le serveur effectue une boucle d'attente active et reçoit des évènements de chaque capteur (un vecteur d'horloge et une valeur de détection). A intervalle régulier, il trie et ordonne ces **évènements** pour effectuer un comptage des voitures détectées au temps logique t.

Le **thread simulation de chaque capteur** réalise la lecture d'un vecteur de captures (représentant la détection ou non d'une voiture) à intervalle de temps constant. Pendant toute la simulation, un capteur maintient à jour son vecteur d'horloges représenté par un objet "clock" et effectue l'envoie des messages au serveur et aux autres capteurs pour maintenir la synchronisation des évènements.

Les **threads sendMessages()** réalisent des appels indépendants non bloquants aux différentes méthodes d'envoi de messages sur le capteur en question. Ces dernière tire aléatoirement une durée de transmission comprise entre 0 et la valeur lue dans le fichier "Delais.txt" correspondante.

Vecteurs d'horloges :

Afin d'utiliser le concept d'horloge logique et de synchronisation des composantes de notre système, nous avons implémenté le principe de vecteur d'horloges. Les vecteurs d'horloges reposent sur **3 types d'évènements** :

- évènement interne : lecture d'une capture;
- réception d'un message : synchronisation avec les autres composantes;
- envoie d'un message : l'information accompagnée du vecteur d'horloge interne;

Pour ce faire, **nous avons réalisé un objet "clock" qui représente le vecteur d'horloge**. Ce dernier possède des méthodes qui correspondent aux différents évènements cités précédemment. La dimension de ce vecteur est égale au nombre de composantes concernées.

Lors d'un **évènement de type interne ou de l'émission d'un message**, le capteur incrémente la case du vecteur d'horloges correspondant à son identifiant (son horloge local).

Lors de la **réception d'un message**, le capteur incrémente son horloge et modifie les horloges des autres capteurs de son vecteur en prenant le maximum entre sa valeur interne et celle reçue .

En utilisant ensuite les **règles de comparaison** entre les vecteurs d'horloge, nous pouvons ordonner les évènement reçus par le serveur pour réaliser le comptage des voitures. Rappelons qu'une voiture est détectée si, à un même temps logique, les trois capteurs ont envoyé la même information de détection d'une voiture.

II. Questions

A. Donner la différence entre temps physique et temps logique. Le quel utilisons-nous dans notre application et pourquoi ?

Le temps dit physique est le temps que l'on habitude de voir : il apparait sur l'écran de notre ordinateur, sur notre montre, etc... Le problème est que celui-ci diffère selon les machines où nous sommes. En effet, nos ordinateurs calculent ce temps physique grâce à un cristal de quartz vibrant à une fréquence propre. Par conséquent, étant donne que les cristaux diffèrent d'un ordinateur à un autre, on a ce qu'on appelle « a clock skew », ie une **légère variation** entre nos horloges physiques. De plus, en admettant que les temps soient les mêmes sur chaque machine, les systèmes reparties nous obligent a relativiser le temps de « voyage » (limité par la vitesse du réseau), ainsi que le temps de traitement de l'information. C'est pour cette raison que Leslie Lamport a décidé d'introduire le temps dit **physique**. Le principe fondamental de celui-ci est la notion de causalité « **arrive avant** » : tout ce qui nous importe dans ce nouveau temps est de savoir si un événement est arrivé après ou avant un autre, et non le moment **précis** ou il s'est produit.

Dans les systèmes repartis en général, il est plus facile d'utiliser un temps **logique**, pour les raisons énoncées ci-dessus.

B. Quelle est la lacune de la méthode de Lamport? Est-ce que nous aurions pu utiliser la méthode de Lamport pour ce travail ?

La lacune de la méthode de Lamport est que le système peut affecter la même estampille a **plusieurs événements** n'étant pas sur le même processus. Il est alors impossible de savoir lequel de ceux-ci s'est déroulé en premier, ce qui peut être problématique dans certains cas... Cette méthode *ne* pourrait *pas* fonctionner pour nous. En effet, si un capteur envoie 3 vrais de suite au serveur, celui-ci croira alors qu'une voiture est présente est ouvrira donc la barrière. Avec les vecteurs d'horloges, nous pouvons vérifier que **chacun** des vecteurs a bien envoyé un Vrai avant d'ouvrir la barrière.

C. Expliquer en quoi consistent les défis de synchronisation de temps en général.

Le principal défi sur la synchronisation de temps dans les systèmes répartis est le maintient de la cohérence des données et la coordination d'actions. Cette notion est d'autant plus importante que l'application est jugée critique. On peut penser par exemple au contrôle aérien, aux échanges commerciaux, aux applications dites « temps réel ». Dans chacune de ces applications, on ne souhaite en aucun cas une défaillance de synchronisation, car le déroulement correcte est directement relié a celle-ci.

De plus, nous ne pouvons pas toujours utiliser une technologie comme le GPS pour les réseaux de senseurs. En effet, en plus d'un certain coût, ceux-ci peuvent être en dehors de leur couverture, et ne marcheraient alors plus (sans parler de la consommation d'énergie que cela demande).

Les défis de synchronisation se résument donc à avoir des capteurs à *basse consommation*, qui n'auraient pas de problème de *communication* et permettant ainsi des résultats **fiables**.

D. D'après les requis de synchronisation pour les réseaux de capteurs, expliquer pourquoi notre méthode risque de ne pas fonctionner.

Le problème de notre programme, est que le serveur ne peut pas déterminer en « temps réel » la présence d'une voiture. En effet, lorsqu'un évènement est reçu par le serveur, il le meut à la suite des autres évènements dans sa structure de données (vecteur). Ce n'est que lorsque le serveur procède au décompte de voitures que ce vecteur d'évènement est trié selon les propriétés des vecteurs d'horloges. Le principal problème vient du fait que **si le délai de transmission d'un message est plus long que d'autres, le message ne sera pas reçu dans son ordre logique, ne sera ré-ordonné que plus tard et viendra peut être modifier la séquence d'évènements**. Une technique possible afin de résoudre ce problème serait d'utiliser des horloges matricielles dans le but d'acquérir notre événement en temps réel.

E. Dire quel(s) méthode(s) de synchronisation de temps s'appliquerai(en)t le mieux à notre système de capteurs. Justifier la réponse.

La méthode de synchronisation s'adaptant le mieux à notre réseau serait la "Reference Broadcast Synchronisation". En effet, chaque noeud dans la portée de transmission de l'émetteur va enregistrer le temps local de récéption du message envoyé par l'émetteur, pour ensuite échanger son temps de récéption avec les autres récépteurs, afin d'estimer les écarts de temps entre eux. L'implémentation de cette nouvelle méthode ne varierait qu'en l'ajout d'un nouveau capteur qui agirait comme l'émetteur.

F. Trouver un article de synchronisation récent (2009) et pertinent à notre problème que nous pouvons suggérer comme solution.

Cet article explique comment synchroniser un ensemble de capteurs par un mécanisme similaire à la gravitation. Le projet est apparement en cours de développement.
http://www.springerlink.com/content/8g19pl3656403tlk/fulltext.pdf

III. Exécution

Tests de vérifications du système d'horloge logique à estampilles vectorielles :

Protocole :

Pour tester notre programme, notre procédure consistait à utiliser des fichiers "Delais" différents et "Voitures" de plus en plus important. Pour chaque fichier de délais, nous testions tous les fichiers de voitures et pour chaque fichier de voitures nous avons réalisé 10 exécutions de notre programme. Cette procédure a pour objectif la détections des effets de bords de la génération aléatoire des délais de transmissions.

Cas 1 :

Dans notre système, les 3 capteurs sont lancés en même temps. De ce fait, **lorsque les délais de transmissions des messages n'excèdent pas l'intervalle de temps de détection d'un évènement**, nous devrions avoir aucun problème dans la détection de voitures et le concept de vecteur d'horloge n'est pas forcément nécessaire. En effet, dans ce cas précis, nous nous soucions peut de savoir quel capteur a envoyer quel message, ce qui importe est d'obtenir des "blocs" de trois évènement "vrai" consécutifs.

Cas 2 :

Par contre si nous inscrivons des **valeurs maximales de délais supérieures au temps de détection** et que ces temps de transmission sont quelques fois calculés, nous pouvons détecter un problèmes. Comme l'illustre la Fig 1, si le temps de transmission de l'évènement "f" est plus court que le "e" seulement dans certains cas, les valeurs du vecteur d'horloge changent et cela perturbera l'ordre établit.

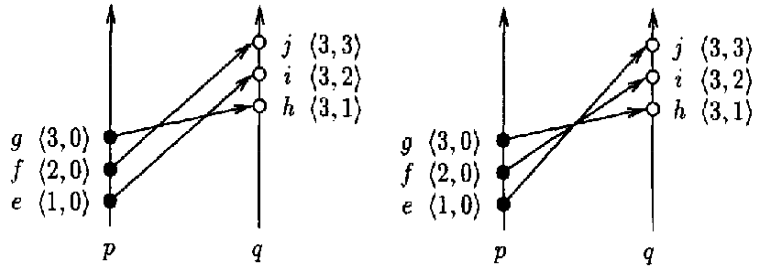


Fig 1 : A limitation of vector timestamps for reconstructing distributed computations. C.J. Fidge.

Dans ce cas qui peut intervenir dans la transmission des messages des capteurs au serveur ou entre les capteurs pour la synchronisation, le résultat final fait que le nombre de voitures détecté est faussé. Le phénomène est remarquable dans l'annexe 1.

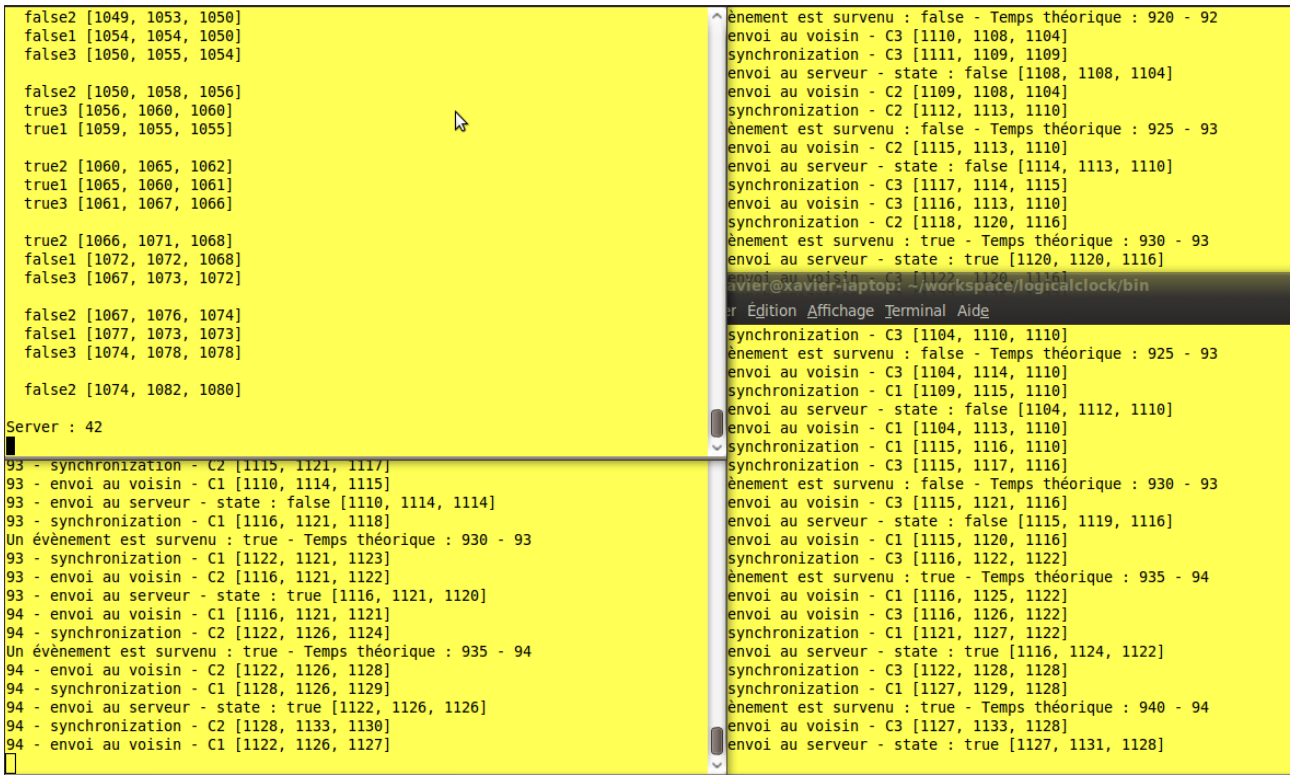
Problème de conception :

Sur cette même annexe, nous pouvons remarquer qu'il y a également une faiblesse au niveau du **trie des vecteurs d'horloges**. En effet, si deux horloges d'un site "i" ont une relation d'infériorité et celles du site "j" de supériorité, alors on ne peut déterminer une relation d'ordre entre ces deux vecteurs. Le mécanisme de trie les considère égaux et les laisses en place.

Autre problème :

De plus, nous pouvons ajouter que même si nous obtenons un nombre de voitures détectées égale au nombre réel de voitures présentes dans notre fichier de référence, nous ne pouvons être certains que ce sont bien ces voitures qui ont été détectées où si cela est du le ré-arrangement. En effet, une valeur "vrai" du temps logique "t+1" peut s'intervertir avec une valeur "vrai" ou "fausse" du temps "t". Ainsi si la séquence au temps "t" "était "VVF" elle devriendra "VVV" et sera détecté comme une voiture.

Exemple d'exécution :



Application en cours d'exécution.

Conclusion

Au cours de ce laboratoire, nous avons pu réfléchir au problème de temps et de synchronisation dans les systèmes répartis. En effet, comme nous avons pu le voir il existe plusieurs méthodes qui on toutes leur avantages et inconvénients en fonction des besoins de l'application. C'est ainsi que nous avons pu implémenter un réseau de capteurs utilisant les vecteurs d'horloges. Cependant, d'après nos tests, cette méthode ne semble pas la plus adaptée puisqu'il est possible d'obtenir des résultats différents pour plusieurs exécutions du programme. Le problème de temps universel et de synchronisation est complexe et est un véritable défi dans le fonctionnement des systèmes répartis.

Annexe 1 : Capture

```
Capteur 1 :
===========
Un évènement est survenu : true - Temps théorique : 780 - 79

79 - envoi au serveur - state : true [939, 935, 936]
79 - envoi au voisin - C3 [941, 935, 936]
79 - synchronization - C2 [942, 940, 937]
79 - envoi au voisin - C2 [940, 935, 936]
Un évènement est survenu : false - Temps théorique : 785 - 79
79 - envoi au serveur - state : false [944, 940, 937]
79 - envoi au voisin - C3 [946, 940, 937]
79 - envoi au voisin - C2 [945, 940, 937]
79 - synchronization - C3 [947, 941, 942]

On a perdu deux messages de synchronisation provenant de C3 qui seront
réceptionnées aux évènements futures.

Un évènement est survenu : false - Temps théorique : 790 - 80
80 - synchronization - C2 [952, 946, 943]
80 - synchronization - C3 [953, 947, 948]
80 - envoi au serveur - state : false [949, 941, 942]
80 - synchronization - C2 [954, 952, 949]
80 - envoi au voisin - C3 [951, 941, 942]
80 - envoi au voisin - C2 [950, 941, 942]
Un évènement est survenu : true - Temps théorique : 795 - 80
80 - envoi au voisin - C3 [958, 952, 949]
80 - synchronization - C3 [959, 953, 953]
80 - envoi au serveur - state : true [956, 952, 949]
81 - synchronization - C2 [960, 958, 954]
81 - synchronization - C3 [961, 959, 960]
81 - envoi au voisin - C2 [957, 952, 949]

Capteur 3 :
===========
Un évènement est survenu : true - Temps théorique : 780 - 79
79 - envoi au voisin - C2 [941, 941, 943]
79 - envoi au serveur - state : true [941, 941, 941]
79 - synchronization - C1 [946, 941, 944]
79 - synchronization - C2 [946, 947, 945]
79 - envoi au voisin - C1 [941, 941, 942]
Un évènement est survenu : false - Temps théorique : 785 - 79
79 - envoi au voisin - C2 [946, 947, 949]
80 - envoi au voisin - C1 [946, 947, 948]
80 - envoi au serveur - state : false [946, 947, 947]
80 - synchronization - C2 [946, 953, 950]
Un évènement est survenu : false - Temps théorique : 790 - 80
80 - synchronization - C1 [951, 953, 955]
80 - envoi au voisin - C2 [946, 953, 954]
80 - synchronization - C2 [951, 959, 956]
80 - synchronization - C1 [958, 959, 957]
80 - envoi au voisin - C1 [946, 953, 953]
80 - envoi au serveur - state : false [946, 953, 952]
```

```
Serveur :
=========
  false1 [944, 940, 937]
  true2 [941, 945, 943]
  false1 [949, 941, 942]

  false3 [946, 947, 947]
  false2 [946, 951, 949]
  false3 [946, 953, 952]
```

Nous pouvons observer que l'ordre des évènements "frontières" a changé. De plus, nous pouvons également voir que les deux évènements sont "incomparables" au sens vectoriel.

Annexe 2 : résultats de test

Délais inférieurs :

	C1	C2	C3
C1	0	2	1
C2	4	0	4
C3	3	3	0
S	2	2	4

Délais frontières :

	C1	C2	C3
C1	0	5	4
C2	5	0	1
C3	4	1	0
S	2	2	4

Délais supérieur :

	C1	C2	C3
C1	0	6	1
C2	2	0	9
C3	7	3	0
S	2	5	4

Délais inférieurs ordonnés :

	C1	C2	C3
C1	0	3	4
C2	2	0	4
C3	3	4	0
S	1	2	3

Le fichier Voitures3 contenait les 30 premières secondes de captures du fichier Voitures.
Le fichier Voitures2 contenait les 120 premières secondes de captures du fichier Voitures.

	Délais frontières					
	voitures3	Voitures3	Voitures2	Voitures2	Voitures	Voitures
référence	13	100	51	100	171	100
1	13	100	47	92,16	132	77,19
2	13	100	50	98,04	116	67,84
3	13	100	50	98,04	122	71,35
4	13	100	47	92,16	112	65,5
5	13	100	40	78,43	111	64,91
6	13	100	46	90,2	163	95,32
7	13	100	42	82,35	132	77,19
8	11	84,62	50	98,04	157	91,81
9	12	92,31	50	98,04	168	98,25
10	13	100	47	92,16	138	80,7
Moyenne	12,7	**97,69**	46,9	**91,96**	135,1	**79,01**
Erreurs	2	**2,31**	10	**8,04**	10	**20,99**

	Délais inférieurs					
	Voitures3	Voitures3	Voitures2	Voitures2	Voitures	Voitures
référence	13	100	51	100	171	100
1	13	100	44	86,27	170	99,42
2	13	100	51	100	144	84,21
3	13	100	51	100	171	100
4	13	100	51	100	75	43,86
5	13	100	51	100	153	89,47
6	13	100	51	100	169	98,83
7	13	100	51	100	166	97,08
8	13	100	51	100	171	100
9	13	100	43	84,31	97	56,73
10	13	100	51	100	140	81,87
Moyenne	13	**100**	49,5	**97,06**	145,6	**85,15**
Erreurs	0	**0**	2	**2,94**	8	**14,85**

	Délais supérieur – envoi au serveur					
	Voitures3	Voitures3	Voitures2	Voitures2	Voitures	Voitures
référence	13	100	51	100	171	100
1	13	100	38	74,51	170	99,42
2	13	100	51	100	154	90,06
3	12	92,31	42	82,35	138	80,7
4	13	100	51	100	116	67,84
5	13	100	48	94,12	127	74,27
6	13	100	51	100	116	67,84
7	12	92,31	32	62,75	169	98,83
8	13	100	24	47,06	170	99,42
9	12	92,31	40	78,43	171	100
10	13	100	50	98,04	117	68,42
Moyenne	12,7	**97,69**	42,7	**83,73**	144,8	**84,68**
Erreurs	3	**2,31**	7	**16,27**	9	**15,32**

	Délais inférieurs – ordonnés					
	Voitures3	Voitures3	Voitures2	Voitures2	Voitures	Voitures
référence	13	100	51	100	171	100
1	13	100	50	98,04	164	95,91
2	13	100	43	84,31	159	92,98
3	13	100	51	100	161	94,15
4	13	100	50	98,04	170	99,42
5	13	100	49	96,08	170	99,42
6	13	100	50	98,04	170	99,42
7	10	76,92	51	100	140	81,87
8	13	100	49	96,08	147	85,96
9	13	100	46	90,2	170	99,42
10	13	100	51	100	170	99,42
Moyenne	12,7	**97,69**	49	**96,08**	162,1	**94,8**
Erreurs	1	**2,31**	7	**3,92**	10	**5,2**